wbia-utool

Release latest

May 21, 2023

Contents:

CHAPTER 1

utool package

1.1 Subpackages

1.1.1 utool._internal package

1.1.1.1 Submodules

1.1.1.2 utool._internal.meta_util_arg module

utool._internal.meta_util_arg.**get_argflag**(*flag*)

utool._internal.meta_util_arg.**get_argval**(*argstr*, *type_=None*, *default=None*)

Returns a value of an argument specified on the command line after some flag

CommandLine: python -c "import utool; print([(type(x), x) for x in [utool.get_argval('–quest')]])" –quest="holy grail" python -c "import utool; print([(type(x), x) for x in [utool.get_argval('–quest')]])" –quest="42" python -c "import utool; print([(type(x), x) for x in [utool.get_argval('–quest')]])" –quest=42 python -c "import utool; print([(type(x), x) for x in [utool.get_argval('–quest')]])" –quest 42 python -c "import utool; print([(type(x), x) for x in [utool.get_argval('–quest', float)]])" –quest 42

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_arg import *  # NOQA
>>> import sys
>>> sys.argv.extend(['--spam', 'eggs', '--quest=holy grail', '--ans=42'])
>>> get_argval('--spam', type_=str, default=None)
eggs
>>> get_argval('--quest', type_=str, default=None)
holy grail
>>> get_argval('--ans', type_=int, default=None)
42
```

1.1.1.3 utool._internal.meta_util_cache module

utool._internal.meta_util_cache.**global_cache_read**(*key*, *appname=None*, ***kwargs*)

1.1.1.4 utool._internal.meta_util_constants module

1.1.1.5 utool._internal.meta_util_cplat module

utool._internal.meta_util_cplat.**get_app_resource_dir**(**args*, ***kwargs*)
Returns a writable directory for an application

Parameters

- **appname** – the name of the application
- **args** – any other subdirectories may be specified

utool._internal.meta_util_cplat.**get_resource_dir**()
Returns a directory which should be writable for any application

1.1.1.6 utool._internal.meta_util_dbg module

utool._internal.meta_util_dbg.**get_caller_lineno**(*N=0*, *strict=True*)

utool._internal.meta_util_dbg.**get_caller_name**(*N=0*, *strict=True*)
Standalone version of get_caller_name

utool._internal.meta_util_dbg.**get_stack_frame**(*N=0*, *strict=True*)

1.1.1.7 utool._internal.meta_util_git module

1.1.1.8 utool._internal.meta_util_iter module

utool._internal.meta_util_iter.**ensure_iterable**(*obj*)

Parameters **obj** (*scalar or iterable*) –

Returns obj if it was iterable otherwise [obj]

Return type it3erable

CommandLine: python -m utool._internal.meta_util_iter –test-ensure_iterable

Timeit: %timeit ut.ensure_iterable([1]) %timeit ut.ensure_iterable(1) %timeit ut.ensure_iterable(np.array(1)) %timeit ut.ensure_iterable([1]) %timeit [1]

Example

```
>>> # DISABLE_DOCTEST
>>> from utool._internal.meta_util_iter import *  # NOQA
>>> # build test data
>>> obj_list = [3, [3], '3', (3,), [3,4,5]]
>>> # execute function
>>> result = [ensure_iterable(obj) for obj in obj_list]
>>> # verify results
>>> result = str(result)
```

(continues on next page)

(continued from previous page)

```
>>> print(result)
[[3], [3], ['3'], (3,), [3, 4, 5]]
```

utool._internal.meta_util_iter.**isiterable**(*obj*)

Returns if the object can be iterated over and is NOT a string # TODO: implement isscalar similar to numpy

Parameters **obj** (*scalar or iterable*) –

Returns

Return type bool

CommandLine: python -m utool._internal.meta_util_iter –test-isiterable

Example

```
>>> # DISABLE_DOCTEST
>>> from utool._internal.meta_util_iter import *  # NOQA
>>> # build test data
>>> obj_list = [3, [3], '3', (3,), [3,4,5]]
>>> # execute function
>>> result = [isiterable(obj) for obj in obj_list]
>>> # verify results
>>> print(result)
[False, True, False, True, True]
```

utool._internal.meta_util_iter.**isscalar**(*obj*)

1.1.1.9 utool._internal.meta_util_path module

utool._internal.meta_util_path.**ensuredir**(*dpath*)

utool._internal.meta_util_path.**truepath**(*path*)

Normalizes and returns absolute path with so specs

utool._internal.meta_util_path.**unixjoin**(**args*)

Like os.path.join, but uses forward slashes on win32

utool._internal.meta_util_path.**unixpath**(*path*)

TODO: rename to unix_truepath Corrects fundamental problems with windows paths.~

1.1.1.10 utool._internal.meta_util_six module

utool._internal.meta_util_six.**ensure_unicode**(*str_*)

Todo: rob gp “isinstance(.*\bstr\b)”

---

utool._internal.meta_util_six.**get_funccode**(*func*)

utool._internal.meta_util_six.**get_funcdoc**(*func*)

utool._internal.meta_util_six.**get_funcglobals**(*func*)

utool._internal.meta_util_six.**get_funcname**(*func*)

utool._internal.meta_util_six.**set_funcdoc**(*func*, *newdoc*)

utool._internal.meta_util_six.**set_funcname**(*func*, *newname*)

1.1.1.11 utool._internal.py2_syntax_funcs module

utool._internal.py2_syntax_funcs.**ignores_exc_tb**(**args*, ***kwargs*)

PYTHON 2 ONLY VERSION – needs to be in its own file for syntactic reasons

ignore_exc_tb decorates a function and remove both itself and the function from any exception traceback that occurs.

This is useful to decorate other trivial decorators which are polluting your stacktrace.

1.1.1.12 utool._internal.randomwrap module

This is a minimal Python client for Mads Haahr's random number generator at www.random.org

This tiny set of functions only implements a subset of the HTTP interface available. In particular it only uses the 'live' # random number generator, and doesn't offer the option of using the alternative 'stored' random # number sets. However, it should be obvious how to extend it by sending requests with different parameters. # The web service code is modelled on Mark Pilgrim's "Dive into Python" tutorial at http://www.diveintopython.org/http_web_services # This client by George Dunbar, University of Warwick (Copyright George Dunbar, 2008) # It is distributed under the Gnu General Public License.

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

See <http://www.gnu.org/licenses/> for a copy of the GNU General Public License. For use that falls outside this license, please contact me.

To use in a python script or at the interactive prompt (randomwrapy.py has to be in the Python search path, of course):

from randomwrapy import *

rnumlistwithoutreplacement(0, 12) # returns a list of the integers 0 - 12 inclusive, in a random order

rnumlistwithreplacement(12, 5) # returns 12 integers from the range [0, 5]

rnumlistwithreplacement(12, 5, 2) # returns 12 integers from the range [2, 5]

rrandom() # returns a random float in the range [0, 1]

reportquota() # tells you how many bits you have available; visit www.random.org/quota for more information

Arguments where given are (must be) numbers, of course. There is almost no error checking in these scripts! For example, if the web site is down, Python will simply raise an exception and report the http error code. See worldrandom.py for an alternative implementation that goes a little further with error checking.

utool._internal.randomwrap.**build_request_parameterNR**(*min*, *max*)

utool._internal.randomwrap.**build_request_parameterWR**(*howmany*, *min*, *max*)

utool._internal.randomwrap.**checkquota**()

utool._internal.randomwrap.**reportquota**()

utool._internal.randomwrap.**rnumlistwithoutreplacement**(*min*, *max*)
Returns a randomly ordered list of the integers between min and max

utool._internal.randomwrap.**rnumlistwithreplacement**(*howmany*, *max*, *min=0*)
Returns a list of howmany integers with a maximum value = max. The minimum value defaults to zero.

utool._internal.randomwrap.**rrandom**()
Get the next random number in the range [0.0, 1.0]. Returns a float.

1.1.1.13 utool._internal.util_importer module

NEEDS CLEANUP SO IT EITHER DOES THE IMPORTS OR GENERATES THE FILE

python -c "import utool"

utool._internal.util_importer.**dynamic_import**(*modname*, *import_tuples*, *developing=True*, *ignore_froms=[]*, *dump=False*, *ignore_startswith=[]*, *ignore_endswith=[]*, *ignore_list=[]*, *check_not_imported=True*, *return_initstr=False*, *verbose=False*)

MAIN ENTRY POINT

Dynamically import listed util libraries and their attributes. Create reload_subs function.

Using __import__ like this is typically not considered good style However, it is better than import * and this will generate the good file text that can be used when the module is 'frozen"

Returns

init_inject_str - by default all imports are executed in this function and only the remainig code needed to be executed is returned to define the reload logic.

str, str: init_inject_str, init_str - if return_initstr is True then also returns init_str defining the from imports.

Return type str

Ignore: ignore_startswith = [] ignore_endswith = [] check_not_imported = True verbose = True

utool._internal.util_importer.**make_import_tuples**(*module_path*, *exclude_modnames=[]*)
Infer the import_tuples from a module_path

utool._internal.util_importer.**make_initstr**(*modname*, *import_tuples*, *verbose=False*)
Just creates the string representation. Does no importing.

1.1.1.14 utool._internal.win32_send_keys module

Check that SendInput can work the way we want it to

The tips and tricks at http://www.pinvoke.net/default.aspx/user32.sendinput is useful!

exception utool._internal.win32_send_keys.**KeySequenceError**
Bases: Exception

Exception raised when a key sequence string has a syntax error

utool._internal.win32_send_keys.**SendKeys**(*keys*, *pause=0.05*, *with_spaces=False*, *with_tabs=False*, *with_newlines=False*, *turn_off_numlock=True*)
Parse the keys and type them

1.1.1.15 Module contents

Need to hack to make work for internal modules and reload subs properly

Regen Command: cd /home/joncrall/code/utool/utool/_internal makeinit.py -x win32_send_keys

1.1.2 utool.experimental package

1.1.2.1 Submodules

1.1.2.2 utool.experimental.bytecode_optimizations module

1.1.2.3 utool.experimental.dynamic_connectivity module

class utool.experimental.dynamic_connectivity.**BinaryNode**(*value*, *parent=None*, *left=None*, *right=None*)

Bases: object

left

right

set_child(*other*, *dir_*)

class utool.experimental.dynamic_connectivity.**DummyEulerTourForest**(*nodes=None*)

Bases: object

maintain a forests of euler tour trees

This is a bad implementation, but will let us use the DynConnGraph api

add_edge(*u*, *v*)

add_node(*node*)

components()

find_root(*node*)

has_edge(*u*, *v*)

remove_edge(*u*, *v*)

subtree(*node*)

to_networkx()

class utool.experimental.dynamic_connectivity.**DynConnGraph**(*graph*)

Bases: object

Stores a spanning forest with Euler Tour Trees

References

https://courses.csail.mit.edu/6.851/spring14/lectures/L20.pdf https://courses.csail.mit.edu/6.851/spring14/lectures/L20.html http://cs.stackexchange.com/questions/33595/what-is-the-most-efficient-algorithm-and-data-structure-for-maintaining-connecte

https://www.cs.princeton.edu/courses/archive/fall03/cs528/handouts/Poly%20logarithmic.pdf

DEFINES ET-Trees http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.192.8615&rep=rep1&type=pdf

https://www.cs.princeton.edu/courses/archive/fall03/cs528/handouts/Poly%20logarithmic.pdf http://courses.csail.mit.edu/6.851/spring12/scribe/L20.pdf http://courses.csail.mit.edu/6.854/16/Projects/B/dynamic-graphs-survey.pdf http://dl.acm.org/citation.cfm?id=502095 http://delivery.acm.org/10.1145/510000/502095/p723-holm.pdf?ip=128.213.17.14&id=502095&acc=ACTIVE%20SERVICE&key=7777116298C9657D%2EAF047EA360787914%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35&CFID=905563744&CFTOKEN=37809688&__acm__=1488222284_4ae91dd7a761430ee714f0c69c17b772

Notes

Paper uses level 0 at top, but video lecture uses floor(log(n)) as top. All edges start at level floor(log(n)). The level of each edge will change over time, but cannot decrease below zero. Let G[i] = subgraph graph at level i. (contains only edges of level i or greater) Let F[i] be Euler tour forest to correspond with G[i]. G[log(n)] = full graph Invariant 1 Every connected component of G_i has at most 2^i vertices. Invariant 2 F[0] $\supseteq$ F[1] $\supseteq$ F[2] $\supseteq$ ... $\supseteq$ F[log(n)].

In other words: F[i] = F[log(n)] $\subseteq$ G_i, and F[log(n)] is the minimum spanning forest of G_{log(n)}, where the weight of an edge is its level. F[0] is a maximum spanning forest if using 0 as top level

CommandLine: python -m utool.experimental.dynamic_connectivity DynConnGraph –show

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.experimental.dynamic_connectivity import *  # NOQA
>>> import networkx as nx
>>> import utool as ut
>>> import wbia.plottool as pt
>>> graph = nx.Graph([
>>>     (0, 1), (0, 2), (0, 3), (1, 3), (2, 4), (3, 4), (2, 3),
>>>     (5, 6), (5, 7), (5, 8), (6, 8), (7, 9), (8, 9), (7, 8),
>>> ])
>>> graph = nx.generators.cycle_graph(5)
>>> self = DynConnGraph(graph)
>>> pt.qtensure()
>>> pt.show_nx(self.graph, fnum=1)
>>> self.show_internals(fnum=2)
>>> self.remove_edge(1, 2)
>>> self.show_internals(fnum=3)
>>> self.remove_edge(3, 4)
>>> self.show_internals(fnum=4)
>>> ut.show_if_requested()
```

add_edge(*u*, *v*)
O(log(n))

is_connected(*u*, *v*)
Check if vertices u and v are connected. Query top level forest F[0] to see if u and v are in the same tree. This can be done by checking F_{log(n)} if Findroot(u) = Findroot(v). This costs O(log(n) / log(log(n))) using B-tree based Euler-Tour trees. but this trades off with a O(log^2(n)/log(log(n))) update This is O(log(n)) otherwise

remove_edge(*u*, *v*)
Using notation where 0 is top level

Intuitively speaking, when the level of a nontree edge is increased, it is because we have discovered that its end points are close enough in F to fit in a smaller tree on a higher level.

show_internals(*fnum=None*)

class utool.experimental.dynamic_connectivity.**EulerTourForest**

Bases: object

add_edge(*u*, *v*)

self = EulerTourForest() self.add_node(1) self.add_node(2) u, v = 1, 2

add_node(*node*)

find_root(*node*)

has_node(*node*)

reroot(*old*, *new*)

class utool.experimental.dynamic_connectivity.**EulerTourList**(*iterable*, *load=1000*)

Bases: object

load-list representation of an Euler tour inspired by sortedcontainers

this doesnt work for the same reason keyed bintrees dont work

the indexing needs to be implicit, but this has explicit indexes

append(*value*)

join(*other*)

Parameters **other** –

CommandLine: python -m sortedcontainers.sortedlist join2

Example

```
>>> from utool.experimental.dynamic_connectivity import *  # NOQA
>>> self = EulerTourList([1, 2, 3, 2, 4, 2, 1], load=3)
>>> other = EulerTourList([0, 5, 9, 5, 0], load=3)
>>> result = self.join(other)
>>> print(result)
```

split(*pos*, *idx*)

update(*iterable*)

class utool.experimental.dynamic_connectivity.**EulerTourTree**

Bases: object

CommandLine: python -m utool.experimental.dynamic_connectivity EulerTourTree –show

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.experimental.dynamic_connectivity import *  # NOQA
>>> #mst = nx.balanced_tree(2, 2)
>>> edges = [
>>>     ('R', 'A'), ('R', 'B'),
>>>     ('B', 'C'), ('C', 'D'), ('C', 'E'),
>>>     ('B', 'F'), ('B', 'G'),
```

(continues on next page)

(continued from previous page)

```
>>> ]
>>> mst = nx.Graph(edges)
>>> self = EulerTourTree.from_tree(mst)
>>> import wbia.plottool as pt
>>> pt.qt4ensure()
>>> fnum = 1
>>> pnum_ = pt.make_pnum_nextgen(1, 3)
>>> pt.show_nx(mst, pnum=pnum_(), fnum=fnum)
>>> pt.show_nx(self.to_networkx(), pnum=pnum_(), fnum=fnum)
>>> pt.show_nx(self.tour_tree.to_networkx(labels=['key', 'value']), pnum=pnum_(),␣
↪fnum=fnum)
>>> print(self.tour)
>>> print(self.first_lookup)
>>> print(self.last_lookup)
>>> ut.show_if_requested()
```

cut(*a*, *b*, *bstjoin=False*)
cuts edge (a, b) into two parts because this is a tree

a, b = (2, 5) print(self.first_lookup[a] > self.first_lookup[b]) tree = self.tour_tree list(tree.item_slice(k1, k2))

find_root(*node*)

classmethod from_tour(*tour*, *fast=False*, *start=0*)

classmethod from_tree(*mst*, *fast=True*, *start=0*)

join(*other*)

reroot(*s*)
s = 3 s = 'B'

Let os denote any occurrence of s. Splice out the first part of the sequence ending with the occurrence before os, remove its first occurrence (or), and tack this on to the end of the sequence which now begins with os. Add a new occurrence os to the end.

to_networkx()

class utool.experimental.dynamic_connectivity.**TestETT**
Bases: object

raise NotImplementedError()

hg clone https://bitbucket.org/mozman/bintrees

git clone git@github.com:Erotemic/bintrees.git

References

https://courses.csail.mit.edu/6.851/spring07/scribe/lec05.pdf http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.192.8615&rep=rep1&type=pdf http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.208.2351&rep=rep1&type=pdf https://en.wikipedia.org/wiki/Euler_tour_technique

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.experimental.dynamic_connectivity import *  # NOQA
>>> edges = [(1, 2), (1, 6), (1, 5), (2, 3), (2, 4)]
>>> edges = [
>>>     ('R', 'A'), ('R', 'B'),
>>>     ('B', 'C'), ('C', 'D'), ('C', 'E'),
>>>     ('B', 'F'), ('B', 'G'),
>>> ]
>>> mst = nx.Graph(edges)
>>> #mst = nx.balanced_tree(2, 11)
>>> self = TestETT.from_tree(mst)
>>> import wbia.plottool as pt
>>> pt.qt4ensure()
>>> pt.show_nx(mst)
```

```
>>> mst = nx.balanced_tree(2, 4)
```

delete_edge_bst_version(*a*, *b*, *bstjoin=False*)

a, b = (2, 5) print(self.first_lookup[a] > self.first_lookup[b]) tree = self.tour_tree list(tree.item_slice(k1, k2))

delete_edge_list_version(*a*, *b*)

classmethod from_tour(*tour*, *version='bst'*, *fast=True*)

classmethod from_tree(*mst*, *version='bst'*, *fast=True*)

```
>>> # DISABLE_DOCTEST
>>> from utool.experimental.dynamic_connectivity import *  # NOQA
>>> mst = nx.balanced_tree(2, 4)
>>> self = TestETT.from_tree(mst)
>>> import wbia.plottool as pt
>>> pt.qt4ensure()
>>> pt.show_nx(self.to_networkx(), pnum=(2, 1, 1), fnum=1)
```

```
>>> a, b = 2, 5
>>> other = self.delete_edge_bst_version(a, b)
>>> pt.show_nx(other.to_networkx(), pnum=(2, 1, 1), fnum=2)
```

join_trees(*t1*, *t2*, *e*)

reroot(*s*)

s = 3 s = 'B'

Let os denote any occurrence of s. Splice out the first part of the sequence ending with the occurrence before os, remove its first occurrence (or), and tack this on to the end of the sequence which now begins with os. Add a new occurrence os to the end.

to_networkx()

utool.experimental.dynamic_connectivity.**comparison**()

CommandLine: python -m utool.experimental.dynamic_connectivity comparison –profile python -m utool.experimental.dynamic_connectivity comparison

utool.experimental.dynamic_connectivity.**euler_tour_dfs**(*G*, *source=None*)

adaptation of networkx dfs

1.1.2.4 utool.experimental.euler_tour_tree_avl module

class utool.experimental.euler_tour_tree_avl.**EulerTourTree**(*iterable=None*, *root=None*)

Bases: *utool.util_dev.NiceRepr*

TODO: generalize out the binary tree sequence part

CommandLine: python -m utool.experimental.euler_tour_tree_avl EulerTourTree

References

Randomized Dynamic Graph Algorithms with Polylogarithmic Time per Operation Henzinger and King 1995 http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.192.8615&rep=rep1&type=pdf

Ignore:

```
>>> # DISABLE_DOCTEST
>>> from utool.experimental.euler_tour_tree_avl import *  # NOQA
>>> ETT = EulerTourTree
>>> self = ETT(['a', 'b', 'c', 'b', 'd', 'b', 'a'])
>>> self._assert_nodes()
>>> other = ETT(['E', 'F', 'G', 'F', 'E'])
>>> other2 = ETT(['E', 'F', 'G', 'F', 'E'])
>>> new = self + other + other2
>>> print(self)
>>> print(other)
>>> print(self + other)
>>> print(new)
>>> print(new + self + self)
>>> self.print_tree()
>>> #other.print_tree()
>>> #self.print_tree()
```

Ignore:

```
>>> # DISABLE_DOCTEST
>>> import networkx as nx
>>> from utool.experimental.euler_tour_tree_avl import *  # NOQA
>>> edges = [
>>>     ('R', 'A'), ('R', 'B'),
>>>     ('B', 'C'), ('C', 'D'), ('C', 'E'),
>>>     ('B', 'F'), ('B', 'G'),
>>> ]
>>> tour = euler_tour(nx.Graph(edges))
>>> print(tour)
>>> self = EulerTourTree(tour)
>>> print(self)
>>> assert list(self) == tour
```

copy()

get_ascii_tree()

get_node(*index*)

join(*other*)

min_elem()

print_tree()

repr_tree
reconstruct represented tree as a DiGraph to preserve the current rootedness

reroot(*first_node*, *last_node*)

Notes

Pick any occurrence of the new root r. Split the tour into A and B, where B is the part of the tour before r. Delete the first node of A and append r. Concatenate B and A.

To change the root of T from r to s: Let os denote any occurrence of s. Splice out the first part of the sequence ending with the occurrence before or, remove its first occurrence (or), and tack this on to the end of the sequence which now begins with os. Add a new occurrence os to the end.

CommandLine: python -m utool.experimental.euler_tour_tree_avl reroot

Ignore:

```
>>> # DISABLE_DOCTEST
>>> import networkx as nx
>>> from utool.experimental.euler_tour_tree_avl import *  # NOQA
>>> edges = [
>>>     ('R', 'A'), ('R', 'B'),
>>>     ('B', 'C'), ('C', 'D'), ('C', 'E'),
>>>     ('B', 'F'), ('B', 'G'),
>>> ]
>>> edges = list(nx.balanced_tree(2, 2).edges())
>>> tour = euler_tour(nx.Graph(edges))
>>> self = EulerTourTree(tour)
>>> print('old_tour = %r' % (self,))
>>> nodes = list(self._traverse_nodes())
>>> self.first_lookup = {node.value: node for node in nodes[::-1]}
>>> self.last_lookup = {node.value: node for node in nodes}
>>> new_root_val = list(self)[445 % (len(tour) - 1)]
>>> new_root_val = 5
>>> print('new_root_val = %r' % (new_root_val,))
>>> first_node = self.first_lookup[new_root_val]
>>> last_node = self.last_lookup[new_root_val]
>>> self.reroot(first_node, last_node)
>>> print('new_tour = %r' % (self,))
>>> ut.quit_if_noshow()
>>> ut.show_if_requested()
```

show_nx(*labels=['value'], edge_labels=False, fnum=None*)

to_networkx(*labels=None*, *edge_labels=False*)
Get a networkx representation of the binary search tree.

values()

class utool.experimental.euler_tour_tree_avl.**Node**(*key=None*, *value=None*)
Bases: *utool.util_dev.NiceRepr*

Internal object, represents a tree node.

free()
Remove all references.

kids

set_child(*direction*, *other*)

val

xdata
compatibility with the C node_t struct

utool.experimental.euler_tour_tree_avl.**ascii_tree**(*root*, *name=None*)

utool.experimental.euler_tour_tree_avl.**avl_insert_dir**(*root*, *new_node*, *direction=1*)
Inserts a single node all the way to the left (direction=1) or right (direction=1)

CommandLine: python -m utool.experimental.euler_tour_tree_avl avl_insert_dir –show python -m utool.experimental.euler_tour_tree_avl avl_insert_dir

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.experimental.euler_tour_tree_avl import *  # NOQA
>>> import utool as ut
>>> root = Node(value='A')
>>> new_node = Node(value='B')
>>> new_root = avl_insert_dir(root, new_node, direction=1)
>>> new_root = avl_insert_dir(root, Node(value='Z'), direction=1)
>>> EulerTourTree(root=new_root)._assert_nodes()
>>> for v in ut.chr_range(5, base='C'):
>>>     new_root = avl_insert_dir(new_root, Node(value=v), direction=1)
>>>     self = EulerTourTree(root=new_root)
>>>     self._assert_nodes()
>>> new = EulerTourTree(root=new_root)
>>> print(new)
>>> ut.quit_if_noshow()
>>> ut.qtensure()
>>> new.show_nx(edge_labels=True)
>>> ut.show_if_requested()
>>> #ascii_tree(root)
>>> #print(result)
```

utool.experimental.euler_tour_tree_avl.**avl_join**(*t1*, *t2*, *node*)
Joins two trees *t1* and *t1* with an intermediate key-value pair

CommandLine: python -m utool.experimental.euler_tour_tree_avl avl_join

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.experimental.euler_tour_tree_avl import *  # NOQA
>>> self = EulerTourTree(['a', 'b', 'c', 'b', 'd', 'b', 'a'])
>>> other = EulerTourTree(['E', 'F', 'G', 'F', 'E'])
>>> node = Node(value='Q')
>>> root = avl_join(self.root, other.root, node)
>>> new = EulerTourTree(root=root)
>>> print('new = %r' % (new,))
>>> ut.quit_if_noshow()
>>> self.print_tree()
>>> other.print_tree()
>>> new.print_tree()
```

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.experimental.euler_tour_tree_avl import *  # NOQA
>>> self = EulerTourTree(['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K'])
>>> other = EulerTourTree(['X'])
>>> node = Node(value='Q')
>>> root = avl_join(self.root, other.root, node)
>>> new = EulerTourTree(root=root)
>>> print('new = %r' % (new,))
>>> ut.quit_if_noshow()
>>> ut.qtensure()
>>> #self.show_nx(fnum=1)
>>> #other.show_nx(fnum=2)
>>> new.show_nx()
```

Running Time: O(abs(r(t1) - r(t2))) O(abs(height(t1) - height(t2)))

utool.experimental.euler_tour_tree_avl.**avl_join2**(*t1*, *t2*)
join two trees without any intermediate key

Returns new_root

Return type *Node*

O(log(n) + log(m)) = O(r(t1) + r(t2))

For AVL-Trees the rank r(t1) = height(t1) - 1

utool.experimental.euler_tour_tree_avl.**avl_join_dir_recursive**(*t1*, *t2*, *node*, *direction*)
Recursive version of join_left and join_right TODO: make this iterative using a stack

utool.experimental.euler_tour_tree_avl.**avl_new_top**(*t1*, *t2*, *top*, *direction=0*)

if direction == 0: (t1, t2) is (left, right)

if direction == 1: (t1, t2) is (right, left)

utool.experimental.euler_tour_tree_avl.**avl_release_kids**(*node*)
splits a node from its kids maintaining parent pointers

utool.experimental.euler_tour_tree_avl.**avl_release_parent**(*node*)
removes the parent of a child

utool.experimental.euler_tour_tree_avl.**avl_rotate_double**(*root*, *direction*)
Double rotation, either 0 (left) or 1 (right).

utool.experimental.euler_tour_tree_avl.**avl_rotate_single**(*root*, *direction*)
Single rotation, either 0 (left) or 1 (right).

utool.experimental.euler_tour_tree_avl.**avl_split**(*root*, *node*)
O(log(n))

Parameters

- **root** (Node) – tree root
- **node** (Node) – node to split at

Returns

(tl, tr, node) tl contains all keys in the tree less than node tr contains all keys in the tree greater than node node is the node we split out

Return type puple

CommandLine: python -m utool.experimental.euler_tour_tree_avl avl_split

Ignore:

```
>>> from utool.experimental.euler_tour_tree_avl import *  # NOQA
>>> self = EulerTourTree(ut.chr_range(10))
>>> self.print_tree()
>>> node = self.get_node(5)
>>> part1, part2, bnode = avl_split(self.root, node)
>>> ascii_tree(part1)
>>> ascii_tree(part2)
>>> ascii_tree(bnode)
```

Ignore:

```
>>> from utool.experimental.euler_tour_tree_avl import *  # NOQA
>>> test_avl_split(verbose=2)
```

utool.experimental.euler_tour_tree_avl.**avl_split_first**(*root*)

Removes the minimum element from the tree

Returns new_root, first_node

Return type tuple

O(log(n)) = O(height(root))

utool.experimental.euler_tour_tree_avl.**avl_split_last**(*root*)

Removes the maximum element from the tree

Returns new_root, last_node

Return type tuple

O(log(n)) = O(height(root))

utool.experimental.euler_tour_tree_avl.**avl_split_old**(*root*, *key*)

utool.experimental.euler_tour_tree_avl.**backtrace_root**(*node*)

Ignore:

```
>>> from utool.experimental.euler_tour_tree_avl import *  # NOQA
>>> self = EulerTourTree(range(10))
>>> self._assert_nodes()
>>> root = self.root
>>> node = self.get_node(5)
>>> self.print_tree()
>>> print('node = %r' % (node,))
>>> rpath = backtrace_root(node)
>>> print('rpath = %r' % (rpath,))
```

utool.experimental.euler_tour_tree_avl.**euler_tour**(*G*, *node=None*, *seen=None*, *visited=None*)

definition from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.192.8615&rep=rep1&type=pdf

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.experimental.euler_tour_tree_avl import *  # NOQA
>>> edges = [
>>>     ('R', 'A'), ('R', 'B'),
>>>     ('B', 'C'), ('C', 'D'), ('C', 'E'),
>>>     ('B', 'F'), ('B', 'G'),
>>> ]
>>> G = nx.Graph(edges)
>>> node = list(G.nodes())[0]
>>> et1 = euler_tour(G, node)
>>> et2 = euler_tour_dfs(G, node)
```

utool.experimental.euler_tour_tree_avl.**euler_tour_dfs**(*G*, *source=None*)
adaptation of networkx dfs

utool.experimental.euler_tour_tree_avl.**height**(*node*)

utool.experimental.euler_tour_tree_avl.**test_avl_split**(*verbose=1*)

1.1.2.5 utool.experimental.pandas_highlight module

utool.experimental.pandas_highlight.**monkey_to_str_columns**(*self*, *latex=False*)

utool.experimental.pandas_highlight.**pandas_repr**(*df*)

utool.experimental.pandas_highlight.**to_string_monkey**(*df*, *highlight_cols=None*, *latex=False*)
monkey patch to pandas to highlight the maximum value in specified cols of a row

Ignore:

```
>>> from utool.experimental.pandas_highlight import *
>>> import pandas as pd
>>> df = pd.DataFrame(
>>>     np.array([[ 0.9,         0.86886931,  0.86842073,  0.9       ],
>>>               [ 0.34196218,  0.34289191,  0.34206377,  0.34252863],
>>>               [ 0.34827074,  0.34827074,  0.34827074,  0.34827074],
>>>               [ 0.76979453,  0.77214855,  0.77547518,  0.38850962]]),
>>>     columns=['sum(fgweights)', 'sum(weighted_ratio)', 'len(matches)',
↪'score_lnbnn_1vM'],
>>>     index=['match_state(match-v-rest)', 'match_state(nomatch-v-rest)',
↪'match_state(notcomp-v-rest)', 'photobomb_state']
>>> )
>>> highlight_cols = 'all'
>>> print(to_string_monkey(df, highlight_cols))
>>> print(to_string_monkey(df, highlight_cols, latex=True))
```

ut.editfile(pd.io.formats.printing.adjoin)

1.1.2.6 Module contents

1.1.3 utool.tests package

1.1.3.1 Submodules

1.1.3.2 utool.tests._oldtest_decor module

class utool.tests._oldtest_decor.**BoringTestClass**
Bases: object

getter_eggs(*eggs*)

getter_spam(*spam_input_*, **spamargs*, ***spamkwds*)

method1(**args*, ***kwargs*)

method2(*a*, *b*, *c=True*, *d='343'*, **args_*, ***kwargs_*)

method3()

setter(*input_*, *values*)

setter_special(*input_*, *values*, ***kwargs*)

utool.tests._oldtest_decor.**decor**(*func*)

utool.tests._oldtest_decor.**func1**(*arg1*, *arg2*)

utool.tests._oldtest_decor.**func2**(*arg1*, *arg2*, *arg4=5*, **args*, ***kwargs*)

utool.tests._oldtest_decor.**func3**(*arg1*, *arg2*, *arg4=5*, ***kwargs*)

utool.tests._oldtest_decor.**func4**(**args*)

utool.tests._oldtest_decor.**func5**(***kwargs*)

utool.tests._oldtest_decor.**func6**(**args*, ***kwargs*)

utool.tests._oldtest_decor.**main**()

Ignore:

```
>>> # ENABLE_DOCTEST
>>> from utool.tests.test_decor import *  # NOQA
>>> result = main()
>>> print(result)
```

utool.tests._oldtest_decor.**print_argspec**(*func*)

utool.tests._oldtest_decor.**test_decorator_module**()

1.1.3.3 utool.tests._oldtest_hash module

REMEMBER In Python 3 think: text or data. str.encode: text -> data bytes.decode: data -> text

utool.tests._oldtest_hash.**test_augment_uuid**()

utool.tests._oldtest_hash.**test_byteslike**()

utool.tests._oldtest_hash.**test_file_hash**()

utool.tests._oldtest_hash.**test_hashstr**()

utool.tests._oldtest_hash.**test_hashstr_components**()

1.1.3.4 utool.tests._oldtest_logging module

utool.tests._oldtest_logging.**func1**()

utool.tests._oldtest_logging.**func2**()

utool.tests._oldtest_logging.**remove_timestamp**(*string*)

utool.tests._oldtest_logging.**test**()

1.1.3.5 utool.tests._oldtest_reloading module

utool.tests._oldtest_reloading.**docstr_test1**()

utool.tests._oldtest_reloading.**docstr_test2**()

utool.tests._oldtest_reloading.**print_docstr**()

utool.tests._oldtest_reloading.**print_ids**()

utool.tests._oldtest_reloading.**reloading_test1**()

utool.tests._oldtest_reloading.**reloading_test2**()

1.1.3.6 utool.tests.run_tests module

utool.tests.run_tests.**convert_tests_from_utool_to_nose**(*module_list*)

utool.tests.run_tests.**run_tests**()

1.1.3.7 Module contents

1.2 Submodules

1.3 utool.DynamicStruct module

class utool.DynamicStruct.**DynStruct**(*child_exclude_list=[]*, *copy_dict=None*, *copy_class=None*)

Bases: *utool.Printable.AbstractPrintable*

dynamically add and remove members

add_dict(*dyn_dict*)
Adds a dictionary to the prefs

copy()

deepcopy(***kwargs*)

execstr(*local_name*)
returns a string which when evaluated will add the stored variables to the current namespace

localname is the name of the variable in the current scope * use locals().update(dyn.to_dict()) instead

flat_dict(*dyn_dict={}*, *only_public=True*)

to_dict()
Converts dynstruct to a dictionary.

update(***kwargs*)

1.4 utool.Preferences module

FIXME: This class is very old, convoluted, and coupled. It really needs to be rewritten efficiently. the __setattr__ __getattr__ stuff needs to be redone, and DynStruct probably needs to go away.

class utool.Preferences.**Pref**(*default=<class 'utool.DynamicStruct.DynStruct'>*, *doc='empty docs'*, *hidden=False*, *choices=None*, *depeq=None*, *fpath=''*, *name='root'*, *parent=None*)
Bases: *utool.DynamicStruct.DynStruct*

Structure for Creating Preferences. Caveats: When using a value call with ['valname'] to be safe Features: * Can be saved and loaded. * Can be nested * Dynamically add/remove

asdict(*split_structs_bit=False*)
Converts prefeters to a dictionary. Children Pref can be optionally separated

change_combo_val(*new_val*)
Checks to see if a selection is a valid index or choice of a combo preference

createQWidget()

customPrintableType(*name*)

ensure_attr(*attr*, *default*)

full_name()
returns name all the way up the tree

get_fpath()

get_printable(*type_bit=True*, *print_exclude_aug=[]*)

get_type()

items()
Wow this class is messed up. I had to overwrite items when moving to python3, just because I haden't called it yet

iteritems()
Wow this class is messed up. I had to overwrite items when moving to python3, just because I haden't called it yet

load()
Read pref dict stored on disk. Overwriting current values.

pref_update(*key*, *new_val*)
Changes a preference value and saves it to disk

qt_col_count()

qt_get_child(*row*)

qt_get_data(*column*)

qt_get_parent()

qt_is_editable()

qt_parents_index_of_me()

qt_row_count()

qt_set_leaf_data(*qvar*)

save()
Saves prefs to disk in dict format

to_dict(*split_structs_bit=False*)
Converts prefeters to a dictionary. Children Pref can be optionally separated

toggle(*key*)
Toggles a boolean key

update(***kwargs*)

value()

class utool.Preferences.**PrefChoice**(*choices*, *default*)
Bases: *utool.DynamicStruct.DynStruct*

change_val(*new_val*)

combo_val()

get_tuple()

class utool.Preferences.**PrefInternal**(*name*, *doc*, *default*, *hidden*, *fpath*, *depeq*, *choices*)
Bases: *utool.DynamicStruct.DynStruct*

freeze_type()

get_type()

class utool.Preferences.**PrefTree**(*parent*)
Bases: *utool.DynamicStruct.DynStruct*

utool.Preferences.**test_Preferences**()

CommandLine: python -m utool.Preferences –test-test_Preferences –show –verbpref

Example

```
>>> # DISABLE_DOCTEST
>>> # xdoctest: +REQUIRES(module:wbia.guitool)
>>> # FIXME depends on guitool_ibei
>>> from utool.Preferences import *  # NOQA
>>> import utool as ut
>>> import wbia.guitool
>>> wbia.guitool.ensure_qtapp()
>>> root = test_Preferences()
>>> ut.quit_if_noshow()
>>> widget = root.createQWidget()
>>> #widget.show()
>>> wbia.guitool.qtapp_loop(widget)
```

1.5 utool.Printable module

DEPRICATE

class utool.Printable.**AbstractPrintable**(*child_print_exclude=[]*)
Bases: object

A base class that prints its attributes instead of the memory address

format_printable(*type_bit=False*, *indstr=' * '*)

get_printable(*type_bit=True*, *print_exclude_aug=[]*, *val_bit=True*, *max_valstr=-1*, *justlength=False*)

printme()

printme2(*type_bit=True*, *print_exclude_aug=[]*, *val_bit=True*, *max_valstr=-1*, *justlength=True*)

printme3()

utool.Printable.**npArrInfo**(*arr*)
OLD update and refactor

utool.Printable.**printableType**(*val*, *name=None*, *parent=None*)
Tries to make a nice type string for a value. Can also pass in a Printable parent object

utool.Printable.**printableVal**(*val*, *type_bit=True*, *justlength=False*)
Very old way of doing pretty printing. Need to update and refactor. DEPRICATE

1.6 utool.__main__ module

utool.__main__.**main**()

1.7 utool.oldalg module

utool.oldalg.**bayesnet**()

References

https://class.coursera.org/pgm-003/lecture/17 http://www.cs.ubc.ca/~murphyk/Bayes/bnintro.html http://www3.cs.stonybrook.edu/~sael/teaching/cse537/Slides/chapter14d_BP.pdf http://www.cse.unsw.edu.au/~cs9417ml/Bayes/Pages/PearlPropagation.html https://github.com/pgmpy/pgmpy.git http://pgmpy.readthedocs.org/en/latest/ http://nipy.bic.berkeley.edu:5000/download/11

utool.oldalg.**bayesnet_examples**()

1.8 utool.sandbox module

class utool.sandbox.**LazyModule**(*modname*)
Bases: object

Waits to import the module until it is actually used

1.9 utool.util_alg module

utool.util_alg.**absdiff**(*x*, *y*)

utool.util_alg.**almost_allsame**(*vals*)

utool.util_alg.**almost_eq**(*arr1*, *arr2*, *thresh=1e-11*, *ret_error=False*)
checks if floating point number are equal to a threshold

utool.util_alg.**apply_grouping**(*items*, *groupxs*)
applies grouping from group_indicies non-optimized version

Parameters

- **items** (*list*) – items to group
- **groupxs** (*list of list of ints*) – grouped lists of indicies

SeeAlso: vt.apply_grouping - optimized numpy version ut.group_indices

CommandLine: python -m utool.util_alg –exec-apply_grouping –show

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_alg import *  # NOQA
>>> import utool as ut
>>> idx2_groupid = [2, 1, 2, 1, 2, 1, 2, 3, 3, 3, 3]
>>> items         = [1, 8, 5, 5, 8, 6, 7, 5, 3, 0, 9]
>>> (keys, groupxs) = ut.group_indices(idx2_groupid)
>>> grouped_items = ut.apply_grouping(items, groupxs)
>>> result = ut.repr2(grouped_items)
>>> print(result)
[[8, 5, 6], [1, 5, 8, 7], [5, 3, 0, 9]]
```

utool.util_alg.**bayes_rule**(*b_given_a*, *prob_a*, *prob_b*)
P(A | B) = frac{ P(B | A) P(A) }{ P(B) }

Parameters

- **b_given_a** (*ndarray or float*) –
- **prob_a** (*ndarray or float*) –
- **prob_b** (*ndarray or float*) –

Returns a_given_b

Return type ndarray or float

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_alg import *  # NOQA
>>> b_given_a = .1
>>> prob_a = .3
>>> prob_b = .4
>>> a_given_b = bayes_rule(b_given_a, prob_a, prob_b)
```

(continues on next page)

(continued from previous page)

```
>>> result = a_given_b
>>> print(result)
0.075
```

utool.util_alg.**choose**(*n*, *k*)
N choose k

binomial combination (without replacement) scipy.special.binom

utool.util_alg.**colwise_diag_idxs**(*size*, *num=2*)
dont trust this implementation or this function name

Parameters **size** (*int*) –

Returns upper_diag_idxs

Return type

?

CommandLine: python -m utool.util_alg –exec-colwise_diag_idxs –size=5 –num=2 python -m utool.util_alg –exec-colwise_diag_idxs –size=3 –num=3

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_alg import *  # NOQA
>>> size = ut.get_argval('--size', default=5)
>>> num = ut.get_argval('--num', default=2)
>>> mat = np.zeros([size] * num, dtype=np.int)
>>> upper_diag_idxs = colwise_diag_idxs(size, num)
>>> poses = np.array(upper_diag_idxs)
>>> idxs = np.ravel_multi_index(poses.T, mat.shape)
>>> print('poses.T =\n%s' % (ut.repr2(poses.T),))
>>> mat[tuple(poses.T)] = np.arange(1, len(poses) + 1)
>>> print(mat)
poses.T =
np.array([[0, 0, 1, 0, 1, 2, 0, 1, 2, 3],
          [1, 2, 2, 3, 3, 3, 4, 4, 4, 4]])
```

utool.util_alg.**compare_groups**(*true_groups*, *pred_groups*)
Finds how predictions need to be modified to match the true grouping.

Notes

pred_merges - the merges needed that would need to be done for the pred_groups to match true_groups.

pred_hybrid - the hybrid split/merges needed that would need to be done for the pred_groups to match true_groups.

Ignore:

```
>>> # ENABLE_DOCTEST
>>> from utool.util_alg import *  # NOQA
>>> import utool as ut
>>> true_groups = [
```

(continues on next page)

(continued from previous page)

```
>>>    [20, 21], [22, 23], [1, 2], [12, 13, 14], [4], [5, 6, 3], [7, 8],
>>>    [9, 10, 11], [31, 32, 33, 34, 35],    [41, 42, 43, 44], [45], [50]
>>> ]
>>> pred_groups = [
>>>     [20, 21, 22, 23], [1, 2], [12], [13, 14], [3, 4], [5, 6,11],
>>>     [7], [8, 9], [10], [31, 32], [33, 34, 35], [41, 42, 43, 44, 45]
>>> ]
>>> comparisons = ut.compare_groups(true_groups, pred_groups)
>>> print(comparisons)
>>> result = ut.repr4(comparisons)
>>> print(result)
{
    'common': {{1, 2}},
    'pred_hybrid': {{10}, {3, 4}, {5, 6, 11}, {7}, {8, 9}},
    'pred_merges': [{{12}, {13, 14}}, {{31, 32}, {33, 34, 35}}],
    'pred_splits': [{20, 21, 22, 23}, {41, 42, 43, 44, 45}],
    'true_hybrid': {{3, 5, 6}, {4}, {50}, {7, 8}, {9, 10, 11}},
    'true_merges': [{12, 13, 14}, {31, 32, 33, 34, 35}],
    'true_splits': [{{20, 21}, {22, 23}}, {{41, 42, 43, 44}, {45}}],
}
```

utool.util_alg.**cumsum**(*item_list*, *initial=0*)

python cumsum

Parameters

- **item_list** (*list*) – list of numbers or items supporting addition
- **initial** (*value*) – initial zero value

Returns list of accumulated values

Return type list

References

stackoverflow.com/questions/9258602/elegant-pythonic-cumsum

CommandLine: python -m utool.util_alg cumsum

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_alg import *  # NOQA
>>> import utool as ut
>>> item_list = [1, 2, 3, 4, 5]
>>> initial = 0
>>> result = cumsum(item_list, initial)
>>> assert result == [1, 3, 6, 10, 15]
>>> print(result)
>>> item_list = zip([1, 2, 3, 4, 5])
>>> initial = tuple()
>>> result2 = cumsum(item_list, initial)
>>> assert result2 == [(1,), (1, 2), (1, 2, 3), (1, 2, 3, 4), (1, 2, 3, 4, 5)]
>>> print(result2)
```

utool.util_alg.**deg_to_rad**(*degree*)

utool.util_alg.**diagonalized_iter**(*size*)

TODO: generalize to more than 2 dimensions to be more like itertools.product.

CommandLine: python -m utool.util_alg –exec-diagonalized_iter python -m utool.util_alg –exec-diagonalized_iter –size=5

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_alg import *  # NOQA
>>> import utool as ut
>>> size = ut.get_argval('--size', default=4)
>>> iter_ = diagonalized_iter(size)
>>> mat = [[None] * size for _ in range(size)]
>>> for count, (r, c) in enumerate(iter_):
>>>     mat[r][c] = count
>>> result = ut.repr2(mat, nl=1, packed=True)
>>> print(result)
[[0, 2, 5, 9],
 [1, 4, 8, 12],
 [3, 7, 11, 14],
 [6, 10, 13, 15],]
```

utool.util_alg.**edit_distance**(*string1*, *string2*)

Edit distance algorithm. String1 and string2 can be either strings or lists of strings

pip install python-Levenshtein

Parameters

- **string1** (*str or list*) –
- **string2** (*str or list*) –

CommandLine: python -m utool.util_alg edit_distance –show

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_alg import *  # NOQA
>>> import utool as ut
>>> string1 = 'hello world'
>>> string2 = ['goodbye world', 'rofl', 'hello', 'world', 'lowo']
>>> edit_distance(['hello', 'one'], ['goodbye', 'two'])
>>> edit_distance('hello', ['goodbye', 'two'])
>>> edit_distance(['hello', 'one'], 'goodbye')
>>> edit_distance('hello', 'goodbye')
>>> distmat = edit_distance(string1, string2)
>>> result = ('distmat = %s' % (ut.repr2(distmat),))
>>> print(result)
>>> [7, 9, 6, 6, 7]
```

utool.util_alg.**enumerate_primes**(*max_prime=4100*)

utool.util_alg.**euclidean_dist**(*vecs1*, *vec2*, *dtype=None*)

utool.util_alg.**expensive_task_gen**(*num=8700*)

Runs a task that takes some time

Parameters **num** (*int*) – (default = 8700)

CommandLine: python -m utool.util_alg expensive_task_gen –show

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_alg import *  # NOQA
>>> import utool as ut
>>> #num = 8700
>>> num = 40000
>>> with ut.Timer('expanesive task'):
>>>     time_list = list(ut.expensive_task_gen(num))
>>> print(sum(time_list))
>>> ut.quit_if_noshow()
>>> import wbia.plottool as pt
>>> #pt.plot(time_list)
>>> from scipy.optimize import curve_fit
>>> def func(x, a, b, c, d):
>>>     return a * np.exp(-c * x) + d
>>> #a*x**3 + b*x**2 +c*x + d
>>> y = np.array(time_list)
>>> y = np.array(ut.cumsum(y))
>>> x = np.arange(len(y))
>>> #popt, pcov = curve_fit(func, x, y, p0=(1, 1e-6, 1))
>>> #print('pcov = %r' % (pcov,))
>>> #print('popt = %r' % (popt,))
>>> # http://stackoverflow.com/questions/3433486/-curve-fitting-in-python
>>> pt.plt.plot(x[::num//50], y[::num//50], 'rx', label='measured data')
>>> #x2 = np.arange(len(y) * 2)
>>> #pt.plt.plot(x2, func(x2, *popt), 'b', label="Fitted Curve") #same as line␣
↪above \/
>>> #pt.plt.legend(loc='upper left')
>>> ut.show_if_requested()
```

utool.util_alg.**factors**(*n*)

Computes all the integer factors of the number *n*

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_alg import *  # NOQA
>>> import utool as ut
>>> result = sorted(ut.factors(10))
>>> print(result)
[1, 2, 5, 10]
```

References

http://stackoverflow.com/questions/6800193/finding-all-the-factors

utool.util_alg.**fibonacci**(*n*)

Parameters **n** (*int*) –

Returns the n-th fibonacci number

Return type int

References

http://stackoverflow.com/questions/15047116/iterative-alg-fib

CommandLine: python -m utool.util_alg fibonacci_iterative

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_alg import *  # NOQA
>>> import utool as ut
>>> with ut.Timer('fib iter'):
>>>     series = [fibonacci_iterative(n) for n in range(20)]
>>> result = ('series = %s' % (str(series[0:10]),))
>>> print(result)
series = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

utool.util_alg.**fibonacci_approx**(*n*)

approximate value (due to numerical errors) of fib(n) using closed form expression

Parameters **n** (*int*) –

Returns the n-th fib number

Return type int

CommandLine: python -m utool.util_alg fibonacci_approx

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_alg import *  # NOQA
>>> import utool as ut
>>> series = [int(fibonacci_approx(n)) for n in range(20)]
>>> result = ('series = %s' % (str(series[0:10]),))
>>> print(result)
```

utool.util_alg.**fibonacci_iterative**(*n*)

Parameters **n** (*int*) –

Returns the n-th fibonacci number

Return type int

References

http://stackoverflow.com/questions/15047116/iterative-alg-fib

CommandLine: python -m utool.util_alg fibonacci_iterative

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_alg import *  # NOQA
>>> import utool as ut
>>> with ut.Timer('fib iter'):
>>>     series = [fibonacci_iterative(n) for n in range(20)]
>>> result = ('series = %s' % (str(series[0:10]),))
>>> print(result)
series = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

utool.util_alg.**fibonacci_recursive**(*n*)

CommandLine: python -m utool.util_alg –test-fibonacci_recursive

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_alg import *  # NOQA
>>> import utool as ut
>>> with ut.Timer('fib rec'):
>>>     series = [fibonacci_recursive(n) for n in range(20)]
>>> result = ('series = %s' % (str(series[0:10]),))
>>> print(result)
series = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

utool.util_alg.**find_group_consistencies**(*groups1*, *groups2*)

Returns a measure of group consistency

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_alg import *  # NOQA
>>> groups1 = [[1, 2, 3], [4], [5, 6]]
>>> groups2 = [[1, 2], [4], [5, 6]]
>>> common_groups = find_group_consistencies(groups1, groups2)
>>> result = ('common_groups = %r' % (common_groups,))
>>> print(result)
common_groups = [(5, 6), (4,)]
```

utool.util_alg.**find_group_differences**(*groups1*, *groups2*)

Returns a measure of how disimilar two groupings are

Parameters

- **groups1** (*list*) – true grouping of items
- **groups2** (*list*) – predicted grouping of items

CommandLine: python -m utool.util_alg find_group_differences

SeeAlso: vtool.group_indicies vtool.apply_grouping

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_alg import *  # NOQA
>>> groups1 = [[1, 2, 3], [4], [5, 6], [7, 8], [9, 10, 11]]
>>> groups2 = [[1, 2, 11], [3, 4], [5, 6], [7], [8, 9], [10]]
>>> total_error = find_group_differences(groups1, groups2)
>>> result = ('total_error = %r' % (total_error,))
>>> print(result)
total_error = 20
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_alg import *  # NOQA
>>> groups1 = [[1, 2, 3], [4], [5, 6]]
>>> groups2 = [[1, 2, 3], [4], [5, 6]]
>>> total_error = find_group_differences(groups1, groups2)
>>> result = ('total_error = %r' % (total_error,))
>>> print(result)
total_error = 0
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_alg import *  # NOQA
>>> groups1 = [[1, 2, 3], [4], [5, 6]]
>>> groups2 = [[1, 2], [4], [5, 6]]
>>> total_error = find_group_differences(groups1, groups2)
>>> result = ('total_error = %r' % (total_error,))
>>> print(result)
total_error = 4
```

Ignore: # Can this be done via sklearn label analysis? # maybe no... the labels assigned to each component are arbitrary # maybe if we label edges? likely too many labels. groups1 = [[1, 2, 3], [4], [5, 6], [7, 8], [9, 10, 11]] groups2 = [[1, 2, 11], [3, 4], [5, 6], [7], [8, 9], [10]]

utool.util_alg.**flatten_membership_mapping**(*uid_list*, *members_list*)

utool.util_alg.**generate_primes**(*stop=None*, *start_guess=2*)

utool.util_alg.**get_nth_bell_number**(*n*)

Returns the (num_items - 1)-th Bell number using recursion. The Bell numbers count the number of partitions of a set.

Parameters **n** (*int*) – number of items in a set

Returns

Return type int

References

http://adorio-research.org/wordpress/?p=11460

CommandLine: python -m utool.util_alg –exec-bell –show

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_alg import *  # NOQA
>>> n = 3
>>> result = get_nth_bell_number(n)
>>> print(result)
5
```

utool.util_alg.**get_nth_prime**(*n*, *max_prime=4100*, *safe=True*)
hacky but still brute force algorithm for finding nth prime for small tests

utool.util_alg.**get_nth_prime_bruteforce**(*n*, *start_guess=2*, *start_num_primes=0*)

Parameters **n** (*int*) – the n-th prime (n=2000 takes about a second)

CommandLine: python -m utool.util_alg get_nth_prime_bruteforce –show

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_alg import *  # NOQA
>>> import utool as ut
>>> n_list = []
>>> time_list = []
>>> for n in range(1, 2000 + 2, 500):
>>>     with ut.Timer(verbose=0) as t:
>>>         get_nth_prime_bruteforce(n)
>>>     time_list += [t.ellapsed]
>>>     n_list += [n]
>>> ut.quit_if_noshow()
>>> import wbia.plottool as pt
>>> pt.multi_plot(n_list, [time_list], xlabel='prime', ylabel='time')
>>> ut.show_if_requested()
```

utool.util_alg.**get_phi**()
Golden Ratio: phi = 1 / sqrt(5) / 2.0 = 1.61803398875

utool.util_alg.**get_phi_ratio1**()

utool.util_alg.**get_prime_index**(*prime*)

utool.util_alg.**greedy_max_inden_setcover**(*candidate_sets_dict*, *items*, *max_covers=None*)
greedy algorithm for maximum independent set cover

Covers items with sets from candidate sets. Could be made faster.

CommandLine: python -m utool.util_alg –test-greedy_max_inden_setcover

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_alg import *  # NOQA
>>> import utool as ut
>>> candidate_sets_dict = {'a': [5, 3], 'b': [2, 3, 5],
...                        'c': [4, 8], 'd': [7, 6, 2, 1]}
>>> items = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> max_covers = None
>>> tup = greedy_max_inden_setcover(candidate_sets_dict, items, max_covers)
>>> (uncovered_items, covered_items_list, accepted_keys) = tup
>>> result = ut.repr4((uncovered_items, sorted(list(accepted_keys))), nl=False)
>>> print(result)
([0, 9], ['a', 'c', 'd'])
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_alg import *  # NOQA
>>> import utool as ut
>>> candidate_sets_dict = {'a': [5, 3], 'b': [2, 3, 5],
...                        'c': [4, 8], 'd': [7, 6, 2, 1]}
>>> items = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> max_covers = 1
>>> tup = greedy_max_inden_setcover(candidate_sets_dict, items, max_covers)
>>> (uncovered_items, covered_items_list, accepted_keys) = tup
>>> result = ut.repr4((uncovered_items, sorted(list(accepted_keys))), nl=False)
>>> print(result)
([0, 3, 4, 5, 8, 9], ['d'])
```

utool.util_alg.**group_indices**(*groupid_list*)

groups indicies of each item in `groupid_list`

Parameters **groupid_list** (*list*) – list of group ids

SeeAlso: vt.group_indices - optimized numpy version ut.apply_grouping

CommandLine: python -m utool.util_alg –test-group_indices python3 -m utool.util_alg –test-group_indices

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_alg import *  # NOQA
>>> import utool as ut
>>> groupid_list = ['b', 1, 'b', 1, 'b', 1, 'b', 'c', 'c', 'c', 'c']
>>> (keys, groupxs) = ut.group_indices(groupid_list)
>>> result = ut.repr3((keys, groupxs), nobraces=1, nl=1)
>>> print(result)
[1, 'b', 'c'],
[[1, 3, 5], [0, 2, 4, 6], [7, 8, 9, 10]],
```

utool.util_alg.**grouping_delta**(*old*, *new*, *pure=True*)

Finds what happened to the old groups to form the new groups.

Parameters

- **old** (*set of frozensets*) – old grouping

- **new** (*set of frozensets*) – new grouping
- **pure** (*bool*) – hybrids are separated from pure merges and splits if pure is True, otherwise hybrid cases are grouped in merges and splits.

Returns

delta: dictionary of changes containing the merges, splits, unchanged, and hybrid cases. Except for unchanged, case a subdict with new and old keys. For splits / merges, one of these contains nested sequences to indicate what the split / merge is.

Return type dict

Todo: incorporate addition / deletion of elements?

Notes

merges - which old groups were merged into a single new group. splits - which old groups were split into multiple new groups. hybrid - which old groups had split/merge actions applied. unchanged - which old groups are the same as new groups.

Ignore:

```
>>> # ENABLE_DOCTEST
>>> from utool.util_alg import *  # NOQA
>>> import utool as ut
>>> old = [
>>>     [20, 21, 22, 23], [1, 2], [12], [13, 14], [3, 4], [5, 6,11],
>>>     [7], [8, 9], [10], [31, 32], [33, 34, 35], [41, 42, 43, 44, 45]
>>> ]
>>> new = [
>>>   [20, 21], [22, 23], [1, 2], [12, 13, 14], [4], [5, 6, 3], [7, 8],
>>>   [9, 10, 11], [31, 32, 33, 34, 35],   [41, 42, 43, 44], [45],
>>> ]
>>> delta = ut.grouping_delta(old, new)
>>> assert set(old[0]) in delta['splits']['old']
>>> assert set(new[3]) in delta['merges']['new']
>>> assert set(old[1]) in delta['unchanged']
>>> result = ut.repr4(delta, nl=2, nobr=True, sk=True)
>>> print(result)
unchanged: {
    {1, 2},
},
splits: {
    old: [{20, 21, 22, 23}, {41, 42, 43, 44, 45}],
    new: [{{20, 21}, {22, 23}}, {{41, 42, 43, 44}, {45}}],
},
merges: {
    old: [{{12}, {13, 14}}, {{31, 32}, {33, 34, 35}}],
    new: [{12, 13, 14}, {31, 32, 33, 34, 35}],
},
hybrid: {
    old: {{10}, {3, 4}, {5, 6, 11}, {7}, {8, 9}},
    new: {{3, 5, 6}, {4}, {7, 8}, {9, 10, 11}},
    splits: [{{7}}, {{11}, {5, 6}}, {{10}}, {{3}, {4}}, {{8}, {9}}],
    merges: [{{7}, {8}}, {{4}}, {{3}, {5, 6}}, {{10}, {11}, {9}}],
},
```

Ignore:

```
>>> # ENABLE_DOCTEST
>>> from utool.util_alg import *  # NOQA
>>> import utool as ut
>>> old = [
>>>     [1, 2, 3], [4], [5, 6, 7, 8, 9], [10, 11, 12]
>>> ]
>>> new = [
>>>     [1], [2], [3, 4], [5, 6, 7], [8, 9, 10, 11, 12]
>>> ]
>>> # every case here is hybrid
>>> pure_delta = ut.grouping_delta(old, new, pure=True)
>>> assert len(ut.flatten(pure_delta['merges'].values())) == 0
>>> assert len(ut.flatten(pure_delta['splits'].values())) == 0
>>> delta = ut.grouping_delta(old, new, pure=False)
>>> delta = ut.order_dict_by(delta, ['unchanged', 'splits', 'merges'])
>>> result = ut.repr4(delta, nl=2, sk=True)
>>> print(result)
{
    unchanged: {},
    splits: [
        [{2}, {3}, {1}],
        [{8, 9}, {5, 6, 7}],
    ],
    merges: [
        [{4}, {3}],
        [{8, 9}, {10, 11, 12}],
    ],
}
```

utool.util_alg.**grouping_delta_stats**(*old*, *new*)

Returns statistics about grouping changes

Parameters

- **old** (*set of frozenset*) – old grouping
- **new** (*set of frozenset*) – new grouping

Returns df: data frame of size statistics

Return type pd.DataFrame

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_alg import *  # NOQA
>>> import utool as ut
>>> old = [
>>>     [20, 21, 22, 23], [1, 2], [12], [13, 14], [3, 4], [5, 6,11],
>>>     [7], [8, 9], [10], [31, 32], [33, 34, 35], [41, 42, 43, 44, 45]
>>> ]
>>> new = [
>>>   [20, 21], [22, 23], [1, 2], [12, 13, 14], [4], [5, 6, 3], [7, 8],
>>>   [9, 10, 11], [31, 32, 33, 34, 35],   [41, 42, 43, 44], [45],
>>> ]
>>> df = ut.grouping_delta_stats(old, new)
>>> print(df)
```

utool.util_alg.**iapply_grouping**(*items*, *groupxs*)

Iterates over groups from group_indicies

Parameters

- **items** (*list*) – items to group
- **groupxs** (*list of list of ints*) – grouped lists of indicies

SeeAlso: vt.apply_grouping - optimized numpy version ut.group_indices

CommandLine: python -m utool.util_alg –exec-apply_grouping –show

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_alg import *  # NOQA
>>> import utool as ut
>>> idx2_groupid = [2, 1, 2, 1, 2, 1, 2, 3, 3, 3, 3]
>>> items         = [1, 8, 5, 5, 8, 6, 7, 5, 3, 0, 9]
>>> (keys, groupxs) = ut.group_indices(idx2_groupid)
>>> grouped_items = list(ut.iapply_grouping(items, groupxs))
>>> result = ut.repr2(grouped_items)
>>> print(result)
[[8, 5, 6], [1, 5, 8, 7], [5, 3, 0, 9]]
```

utool.util_alg.**inbounds**(*num*, *low*, *high*, *eq=False*)

Parameters

- **num** (*scalar or ndarray*) –
- **low** (*scalar or ndarray*) –
- **high** (*scalar or ndarray*) –
- **eq** (*bool*) –

Returns is_inbounds

Return type scalar or ndarray

CommandLine: python -m utool.util_alg –test-inbounds

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_alg import *  # NOQA
>>> import utool as ut
>>> num = np.array([[ 0.   ,  0.431,  0.279],
...                 [ 0.204,  0.352,  0.08 ],
...                 [ 0.107,  0.325,  0.179]])
>>> low  = .1
>>> high = .4
>>> eq = False
>>> is_inbounds = inbounds(num, low, high, eq)
>>> result = ut.repr2(is_inbounds, with_dtype=True)
>>> print(result)
```

(continues on next page)

(continued from previous page)

```
np.array([[False, False,  True],
          [ True,  True, False],
          [ True,  True,  True]], dtype=bool)
```

utool.util_alg.**is_prime**(*num*)

naive function for finding primes. Good for stress testing

References

http://thelivingpearl.com/2013/01/06/how-to-find-prime-numbers-in-python/

CommandLine: python -m utool.util_alg –test-is_prime

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_alg import *  # NOQA
>>> import utool as ut
>>> with ut.Timer('isprime'):
>>>     series = [is_prime(n) for n in range(30)]
>>> result = ('primes = %s' % (str(ut.list_where(series[0:10])),))
>>> print(result)
primes = [2, 3, 5, 7]
```

utool.util_alg.**item_hist**(*list_*)

counts the number of times each item appears in the dictionary

utool.util_alg.**knapsack**(*items*, *maxweight*, *method='recursive'*)

Solve the knapsack problem by finding the most valuable subsequence of *items* subject that weighs no more than *maxweight*.

Parameters

- **items** (*tuple*) – is a sequence of tuples *(value, weight, id_)*, where *value* is a number and *weight* is a non-negative integer, and *id_* is an item identifier.
- **maxweight** (*scalar*) – is a non-negative integer.

Returns

(total_value, items_subset) - a pair whose first element is the sum of values in the most valuable subsequence, and whose second element is the subsequence. Subset may be different depending on implementation (ie top-odwn recusrive vs bottom-up iterative)

Return type tuple

References

http://codereview.stackexchange.com/questions/20569/dynamic-programming-solution-to-knapsack-problem http://stackoverflow.com/questions/141779/solving-the-np-complete-problem-in-xkcd http://www.es.ele.tue.nl/education/5MC10/Solutions/knapsack.pdf

CommandLine: python -m utool.util_alg –test-knapsack

python -m utool.util_alg –test-knapsack:0 python -m utool.util_alg –exec-knapsack:1

Ignore:

```
>>> annots_per_view = 2
>>> maxweight = 2
>>> items = [
>>>     (0.7005208343554686, 0.7005208343554686, 0),
>>>     (0.669270834329427, 0.669270834329427, 1),
>>>     (0.669270834329427, 0.669270834329427, 2),
>>>     (0.7005208343554686, 0.7005208343554686, 3),
>>>     (0.7005208343554686, 0.7005208343554686, 4),
>>>     (0.669270834329427, 0.669270834329427, 5),
>>>     (0.669270834329427, 0.669270834329427, 6),
>>>     (0.669270834329427, 0.669270834329427, 7),
>>>     (0.669270834329427, 0.669270834329427, 8),
>>>     (0.669270834329427, 0.669270834329427, 9),
>>>     (0.669270834329427, 0.669270834329427, 10),
>>>     (0.669270834329427, 0.669270834329427, 11),
>>>     (0.669270834329427, 0.669270834329427, 12),
>>>     (0.669270834329427, 0.669270834329427, 13),
>>>     (0.669270834329427, 0.669270834329427, 14),
>>>     (0.669270834329427, 0.669270834329427, 15),
>>>     (0.669270834329427, 0.669270834329427, 16),
>>>     (0.669270834329427, 0.669270834329427, 17),
>>>     (0.7005208343554686, 0.7005208343554686, 18),
>>>     (0.7005208343554686, 0.7005208343554686, 19),
>>>     (0.669270834329427, 0.669270834329427, 20),
>>>     (0.7005208343554686, 0.7005208343554686, 21),
>>>     (0.669270834329427, 0.669270834329427, 22),
>>>     (0.669270834329427, 0.669270834329427, 23),
>>>     (0.669270834329427, 0.669270834329427, 24),
>>>     (0.669270834329427, 0.669270834329427, 25),
>>>     (0.669270834329427, 0.669270834329427, 26),
>>>     (0.669270834329427, 0.669270834329427, 27),
>>>     (0.669270834329427, 0.669270834329427, 28),
>>>     (0.7005208343554686, 0.7005208343554686, 29),
>>>     (0.669270834329427, 0.669270834329427, 30),
>>>     (0.669270834329427, 0.669270834329427, 31),
>>>     (0.669270834329427, 0.669270834329427, 32),
>>>     (0.669270834329427, 0.669270834329427, 33),
>>>     (0.7005208343554686, 0.7005208343554686, 34),
>>>     (0.669270834329427, 0.669270834329427, 35),
>>>     (0.669270834329427, 0.669270834329427, 36),
>>>     (0.669270834329427, 0.669270834329427, 37),
>>>     (0.7005208343554686, 0.7005208343554686, 38),
>>>     (0.669270834329427, 0.669270834329427, 39),
>>>     (0.669270834329427, 0.669270834329427, 40),
>>>     (0.7005208343554686, 0.7005208343554686, 41),
>>>     (0.669270834329427, 0.669270834329427, 42),
>>>     (0.669270834329427, 0.669270834329427, 43),
>>>     (0.669270834329427, 0.669270834329427, 44),
>>> ]
>>> values = ut.take_column(items, 0)
>>> weights = ut.take_column(items, 1)
>>> indices = ut.take_column(items, 2)
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_alg import *  # NOQA
>>> import utool as ut
>>> items = [(4, 12, 0), (2, 1, 1), (6, 4, 2), (1, 1, 3), (2, 2, 4)]
>>> maxweight = 15
>>> total_value, items_subset = knapsack(items, maxweight, method='recursive')
>>> total_value1, items_subset1 = knapsack(items, maxweight, method='iterative')
>>> result =  'total_value = %.2f\n' % (total_value,)
>>> result += 'items_subset = %r' % (items_subset,)
>>> ut.assert_eq(total_value1, total_value)
>>> ut.assert_eq(items_subset1, items_subset)
>>> print(result)
total_value = 11.00
items_subset = [(2, 1, 1), (6, 4, 2), (1, 1, 3), (2, 2, 4)]
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_alg import *  # NOQA
>>> import utool as ut
>>> # Solve https://xkcd.com/287/
>>> weights = [2.15, 2.75, 3.35, 3.55, 4.2, 5.8] * 2
>>> items = [(w, w, i) for i, w in enumerate(weights)]
>>> maxweight = 15.05
>>> total_value, items_subset = knapsack(items, maxweight, method='recursive')
>>> total_value1, items_subset1 = knapsack(items, maxweight, method='iterative')
>>> total_weight = sum([t[1] for t in items_subset])
>>> print('total_weight = %r' % (total_weight,))
>>> result =  'total_value = %.2f' % (total_value,)
>>> print('items_subset = %r' % (items_subset,))
>>> print('items_subset1 = %r' % (items_subset1,))
>>> #assert items_subset1 == items_subset, 'NOT EQ\n%r !=\n%r' % (items_subset1,␣
↪items_subset)
>>> print(result)
total_value = 15.05
```

Timeit:

```
>>> import utool as ut
>>> setup = ut.codeblock(
>>>     '''
>>>     import utool as ut
>>>     weights = [215, 275, 335, 355, 42, 58] * 40
>>>     items = [(w, w, i) for i, w in enumerate(weights)]
>>>     maxweight = 2505
>>>     #import numba
>>>     #knapsack_numba = numba.autojit(ut.knapsack_iterative)
>>>     #knapsack_numba = numba.autojit(ut.knapsack_iterative_numpy)
>>>     ''')
>>> # Test load time
>>> stmt_list1 = ut.codeblock(
>>>     '''
>>>     #ut.knapsack_recursive(items, maxweight)
```

(continues on next page)

(continued from previous page)

```
>>>     ut.knapsack_iterative(items, maxweight)
>>>     ut.knapsack_ilp(items, maxweight)
>>>     #knapsack_numba(items, maxweight)
>>>     #ut.knapsack_iterative_numpy(items, maxweight)
>>>     ''').split('\n')
>>> ut.util_dev.timeit_compare(stmt_list1, setup, int(5))
```

utool.util_alg.**knapsack_greedy**(*items*, *maxweight*)

non-optimal greedy version of knapsack algorithm does not sort input. Sort the input by largest value first if desired.

Parameters

- **items** (*tuple*) – is a sequence of tuples *(value, weight, id_)*, where *value* is a scalar and *weight* is a non-negative integer, and *id_* is an item identifier.
- **maxweight** (*scalar*) – is a non-negative integer.

CommandLine: python -m utool.util_alg –exec-knapsack_greedy

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_alg import *  # NOQA
>>> items = [(4, 12, 0), (2, 1, 1), (6, 4, 2), (1, 1, 3), (2, 2, 4)]
>>> maxweight = 15
>>> total_value, items_subset = knapsack_greedy(items, maxweight)
>>> result =  'total_value = %r\n' % (total_value,)
>>> result += 'items_subset = %r' % (items_subset,)
>>> print(result)
total_value = 7
items_subset = [(4, 12, 0), (2, 1, 1), (1, 1, 3)]
```

utool.util_alg.**knapsack_ilp**(*items*, *maxweight*, *verbose=False*)

solves knapsack using an integer linear program

CommandLine: python -m utool.util_alg knapsack_ilp

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_alg import *  # NOQA
>>> import utool as ut
>>> # Solve https://xkcd.com/287/
>>> weights = [2.15, 2.75, 3.35, 3.55, 4.2, 5.8, 6.55]
>>> values  = [2.15, 2.75, 3.35, 3.55, 4.2, 5.8, 6.55]
>>> indices = ['mixed fruit', 'french fries', 'side salad',
>>>            'hot wings', 'mozzarella sticks', 'sampler plate',
>>>            'barbecue']
>>> items = [(v, w, i) for v, w, i in zip(values, weights, indices)]
>>> #items += [(3.95, 3.95, 'mystery plate')]
>>> maxweight = 15.05
>>> verbose = True
```

(continues on next page)

(continued from previous page)

```
>>> total_value, items_subset = knapsack_ilp(items, maxweight, verbose)
>>> print('items_subset = %s' % (ut.repr3(items_subset, nl=1),))
```

utool.util_alg.**knapsack_iterative**(*items*, *maxweight*)

utool.util_alg.**knapsack_iterative_int**(*items*, *maxweight*)
Iterative knapsack method

Math: maximize sum_{i in T} v_i subject to sum_{i in T} w_i leq W

Notes

dpmat is the dynamic programming memoization matrix. dpmat[i, w] is the total value of the items with weight at most W T is idx_subset, the set of indicies in the optimal solution

CommandLine: python -m utool.util_alg –exec-knapsack_iterative_int –show

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_alg import *  # NOQA
>>> weights = [1, 3, 3, 5, 2, 1] * 2
>>> items = [(w, w, i) for i, w in enumerate(weights)]
>>> maxweight = 10
>>> items = [(.8, 700, 0)]
>>> maxweight = 2000
>>> print('maxweight = %r' % (maxweight,))
>>> print('items = %r' % (items,))
>>> total_value, items_subset = knapsack_iterative_int(items, maxweight)
>>> total_weight = sum([t[1] for t in items_subset])
>>> print('total_weight = %r' % (total_weight,))
>>> print('items_subset = %r' % (items_subset,))
>>> result =  'total_value = %.2f' % (total_value,)
>>> print(result)
total_value = 0.80
```

Ignore: DPMAT = [[dpmat[r][c] for c in range(maxweight)] for r in range(len(items))] KMAT = [[kmat[r][c] for c in range(maxweight)] for r in range(len(items))]

utool.util_alg.**knapsack_iterative_numpy**(*items*, *maxweight*)
Iterative knapsack method

maximize sum_{i in T} v_i subject to sum_{i in T} w_i leq W

Notes

dpmat is the dynamic programming memoization matrix. dpmat[i, w] is the total value of the items with weight at most W T is the set of indicies in the optimal solution

utool.util_alg.**knapsack_recursive**(*items*, *maxweight*)

utool.util_alg.**longest_common_substring**(*s1*, *s2*)

References

https://en.wikibooks.org/wiki/Algorithm_Implementation/Strings/Longest_common_substring#Python2

utool.util_alg.**max_size_max_distance_subset**(*items*, *min_thresh=0*, *Kstart=2*, *verbose=False*)

Parameters

- **items** –
- **min_thresh** (*int*) – (default = 0)
- **Kstart** (*int*) – (default = 2)

Returns prev_subset_idx

Return type

?

CommandLine: python -m utool.util_alg –exec-max_size_max_distance_subset

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_alg import *  # NOQA
>>> items = [1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> min_thresh = 3
>>> Kstart = 2
>>> verbose = True
>>> prev_subset_idx = max_size_max_distance_subset(items, min_thresh,
>>>                                                Kstart, verbose=verbose)
>>> result = ('prev_subset_idx = %s' % (str(prev_subset_idx),))
>>> print(result)
```

utool.util_alg.**maximin_distance_subset1d**(*items*, *K=None*, *min_thresh=None*, *verbose=False*)

Greedy algorithm, may be exact for 1d case. First, choose the first item, then choose the next item that is farthest away from all previously chosen items. Iterate.

CommandLine: python -m utool.util_alg –exec-maximin_distance_subset1d

Notes

Given a set of items V. Let $E = V times V$ be the the set of all item pairs.

The goal is to return the largest subset of item such that the distance between any pair of items in the subset is greater than some threshold.

Let t[u, v] be the distance between u and v.

Let x[u, v] = 1 if the annotation pair (u, v) is included.

Let y[u] = 1 if the annotation u is included.

Objective: maximize sum(y[u] for u in V)

subject to: # Annotations pairs are only included if their timedelta is less than # the threshold. x[u, v] = 0 if t[u, v] > thresh

If an edge is exclued at least one of its endpoints must be # excluded y[u] + y[v] - x[u, v] < 2

Example

```
>>> # DISABLE_DOCTEST
>>> import utool as ut
>>> from utool.util_alg import *  # NOQA
>>> #items = [1, 2, 3, 4, 5, 6, 7]
>>> items = [20, 1, 1, 9, 21, 6, 22]
>>> min_thresh = 5
>>> K = None
>>> result = maximin_distance_subset1d(items, K, min_thresh, verbose=True)
>>> print(result)
(array([1, 3, 6]), [1, 9, 22])
```

Example

```
>>> # DISABLE_DOCTEST
>>> import utool as ut
>>> from utool.util_alg import *  # NOQA
>>> #items = [1, 2, 3, 4, 5, 6, 7]
>>> items = [0, 1]
>>> min_thresh = 5
>>> K = None
>>> result = maximin_distance_subset1d(items, K, min_thresh, verbose=True)
>>> print(result)
```

utool.util_alg.**maximum_distance_subset**(*items*, *K*, *verbose=False*)

FIXME: I believe this does not work.

Returns a subset of size K from items with the maximum pairwise distance

References

stackoverflow.com/questions/12278528/subset-elements-furthest-apart-eachother
stackoverflow.com/questions/13079563/condensed-distance-matrix-pdist

Recurance: Let X[n,k] be the solution for selecting k elements from first n elements items. X[n, k] = max(max(X[m, k - 1] + (sum_{p in prev_solution} dist(o, p)) for o < n and o not in prev solution)) for m < n.

Example

```
>>> # DISABLE_DOCTEST
>>> import scipy.spatial.distance as spdist
>>> items = [1, 6, 20, 21, 22]
```

CommandLine: python -m utool.util_alg –exec-maximum_distance_subset

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_alg import *  # NOQA
>>> #items = [1, 2, 3, 4, 5, 6, 7]
```

(continues on next page)

(continued from previous page)

```
>>> items = [1, 6, 20, 21, 22]
>>> K = 3
>>> result = maximum_distance_subset(items, K)
>>> print(result)
(42.0, array([4, 3, 0]), array([22, 21,  1]))
```

utool.util_alg.**negative_minclamp_inplace**(*arr*)

utool.util_alg.**norm_zero_one**(*array*, *dim=None*)

normalizes a numpy array from 0 to 1 based in its extent

Parameters

- **array** (*ndarray*) –
- **dim** (*int*) –

Returns

Return type ndarray

CommandLine: python -m utool.util_alg –test-norm_zero_one

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_alg import *  # NOQA
>>> import utool as ut
>>> array = np.array([ 22, 1, 3, 2, 10, 42, ])
>>> dim = None
>>> array_norm = norm_zero_one(array, dim)
>>> result = ut.repr2(list(array_norm), precision=3)
>>> print(result)
[0.512, 0.000, 0.049, 0.024, 0.220, 1.000]
```

utool.util_alg.**normalize**(*array*, *dim=0*)

utool.util_alg.**num_partitions**(*num_items*)

utool.util_alg.**number_of_decimals**(*num*)

Parameters **num** (*float*) –

References

stackoverflow.com/questions/6189956/finding-decimal-places

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_alg import *  # NOQA
>>> num = 15.05
>>> result = number_of_decimals(num)
>>> print(result)
2
```

utool.util_alg.**prod**(*item_list*, *initial=1.0*)
product of all number in a list (like np.prod)

Parameters

- **item_list** (*list*) – list of numbers or items supporting mulitplicatiuon
- **initial** (*value*) – initial identity (default=1)

Returns Multiplied value

Return type float

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_alg import *  # NOQA
>>> initial = 1.0
>>> item_list = [1, 2, 3, 4, 5]
>>> assert prod(item_list, initial) == 120.0
>>> assert prod([]) == 1.0
>>> assert prod([5]) == 5.0
```

utool.util_alg.**product_nonsame**(*list1*, *list2*)
product of list1 and list2 where items are non equal

utool.util_alg.**product_nonsame_self**(*list_*)

utool.util_alg.**rad_to_deg**(*radians*)

utool.util_alg.**safe_div**(*a*, *b*)

utool.util_alg.**safe_pdist**(*arr*, **args*, ***kwargs*)

Kwargs: metric = ut.absdiff

SeeAlso: scipy.spatial.distance.pdist

TODO: move to vtool

utool.util_alg.**self_prodx**(*list_*)

utool.util_alg.**setcover_greedy**(*candidate_sets_dict*, *items=None*, *set_weights=None*, *item_values=None*, *max_weight=None*)
Greedy algorithm for various covering problems. approximation gaurentees depending on specifications like set_weights and item values

Set Cover: log(len(items) + 1) approximation algorithm Weighted Maximum Cover: 1 - 1/e == .632 approximation algorithm Generalized maximum coverage is not implemented

References

https://en.wikipedia.org/wiki/Maximum_coverage_problem

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_alg import *  # NOQA
>>> import utool as ut
>>> candidate_sets_dict = {
>>>     'a': [1, 2, 3, 8, 9, 0],
>>>     'b': [1, 2, 3, 4, 5],
>>>     'c': [4, 5, 7],
>>>     'd': [5, 6, 7],
>>>     'e': [6, 7, 8, 9, 0],
>>> }
>>> max_weight = None
>>> items = None
>>> set_weights = None
>>> item_values = None
>>> greedy_soln = ut.sort_dict(ut.setcover_greedy(candidate_sets_dict))
>>> exact_soln = ut.sort_dict(ut.setcover_ilp(candidate_sets_dict))
>>> print('greedy_soln = %r' % (greedy_soln,))
>>> print('exact_soln = %r' % (exact_soln,))
```

utool.util_alg.**setcover_ilp**(*candidate_sets_dict*, *items=None*, *set_weights=None*, *item_values=None*, *max_weight=None*, *verbose=False*)

Set cover / Weighted Maximum Cover exact algorithm

https://en.wikipedia.org/wiki/Maximum_coverage_problem

utool.util_alg.**solve_boolexpr**()

sudo pip install git+https://github.com/tpircher/quine-mccluskey.git sudo pip uninstall quine_mccluskey pip uninstall quine_mccluskey

pip install git+https://github.com/tpircher/quine-mccluskey.git

Parameters **varnames** –

Returns

Return type

?

CommandLine: python -m utool.util_alg solve_boolexpr –show

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_alg import *  # NOQA
>>> import utool as ut
>>> varnames = ['sa', 'said', 'aid']
>>> result = solve_boolexpr()
>>> print(result)
```

utool.util_alg.**square_pdist**(*arr*, **args*, ***kwargs*)

utool.util_alg.**standardize_boolexpr**(*boolexpr_*, *parens=False*)

Standardizes a boolean expression into an or-ing of and-ed variables

Parameters **boolexpr** (*str*) –

Returns final_expr

Return type str

CommandLine: sudo pip install git+https://github.com/tpircher/quine-mccluskey.git python -m utool.util_alg standardize_boolexpr –show

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_alg import *  # NOQA
>>> import utool as ut
>>> boolexpr_ = 'not force_opencv and (orient_ or is_gif)'
>>> result = standardize_boolexpr(boolexpr_, parens=True)
>>> print(result)
(orient_ and (not force_opencv)) or (is_gif and (not force_opencv))
```

utool.util_alg.**triangular_number**(*n*)

Latex: T_n = sum_{k=1}^{n} k = frac{n (n + 1)}{2} = binom{n + 1}{2}

References

en.wikipedia.org/wiki/Triangular_number

utool.util_alg.**ungroup**(*grouped_items*, *groupxs*, *maxval=None*, *fill=None*)

Ungroups items

Parameters

- **grouped_items** (*list*) –
- **groupxs** (*list*) –
- **maxval** (*int*) – (default = None)

Returns ungrouped_items

Return type list

SeeAlso: vt.invert_apply_grouping

CommandLine: python -m utool.util_alg ungroup_unique

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_alg import *  # NOQA
>>> import utool as ut
>>> grouped_items = [[1.1, 1.2], [2.1, 2.2], [3.1, 3.2]]
>>> groupxs = [[0, 2], [1, 5], [4, 3]]
>>> maxval = None
>>> ungrouped_items = ungroup(grouped_items, groupxs, maxval)
>>> result = ('ungrouped_items = %s' % (ut.repr2(ungrouped_items),))
>>> print(result)
ungrouped_items = [1.1, 2.1, 1.2, 3.2, 3.1, 2.2]
```

utool.util_alg.**ungroup_gen**(*grouped_items*, *groupxs*, *fill=None*)

Ungroups items returning a generator. Note that this is much slower than the list version and is not gaurenteed to have better memory usage.

Parameters

- **grouped_items** (*list*) –
- **groupxs** (*list*) –
- **maxval** (*int*) – (default = None)

Returns ungrouped_items

Return type list

SeeAlso: vt.invert_apply_grouping

CommandLine: python -m utool.util_alg ungroup_unique

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_alg import *  # NOQA
>>> import utool as ut
>>> grouped_items = [[1.1, 1.2], [2.1, 2.2], [3.1, 3.2]]
>>> groupxs = [[1, 2], [5, 6], [9, 3]]
>>> ungrouped_items1 = list(ungroup_gen(grouped_items, groupxs))
>>> ungrouped_items2 = ungroup(grouped_items, groupxs)
>>> assert ungrouped_items1 == ungrouped_items2
>>> grouped_items = [[1.1, 1.2], [2.1, 2.2], [3.1, 3.2]]
>>> groupxs = [[0, 2], [1, 5], [4, 3]]
>>> ungrouped_items1 = list(ungroup_gen(grouped_items, groupxs))
>>> ungrouped_items2 = ungroup(grouped_items, groupxs)
>>> assert ungrouped_items1 == ungrouped_items2
```

Ignore: labels = np.random.randint(0, 64, 10000) unique_labels, groupxs = ut.group_indices(labels) grouped_items = ut.apply_grouping(np.arange(len(labels)), groupxs) ungrouped_items1 = list(ungroup_gen(grouped_items, groupxs)) ungrouped_items2 = ungroup(grouped_items, groupxs) assert ungrouped_items2 == ungrouped_items1 %timeit list(ungroup_gen(grouped_items, groupxs)) %timeit ungroup(grouped_items, groupxs)

utool.util_alg.**ungroup_unique**(*unique_items*, *groupxs*, *maxval=None*)

Ungroups unique items to correspond to original non-unique list

Parameters

- **unique_items** (*list*) –
- **groupxs** (*list*) –
- **maxval** (*int*) – (default = None)

Returns ungrouped_items

Return type list

CommandLine: python -m utool.util_alg ungroup_unique

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_alg import *  # NOQA
>>> import utool as ut
>>> unique_items = [1, 2, 3]
>>> groupxs = [[0, 2], [1, 3], [4, 5]]
>>> maxval = None
>>> ungrouped_items = ungroup_unique(unique_items, groupxs, maxval)
>>> result = ('ungrouped_items = %s' % (ut.repr2(ungrouped_items),))
>>> print(result)
ungrouped_items = [1, 2, 1, 2, 3, 3]
```

utool.util_alg.**unixtime_hourdiff**(*x*, *y*)

Parameters

- **x** –
- **y** (*ndarray*) – labels

Returns

Return type

?

CommandLine: python -m utool.util_alg –exec-unixtime_hourdiff –show

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_alg import *  # NOQA
>>> x = np.nan
>>> y = 0
>>> result = unixtime_hourdiff(x, y)
>>> print(result)
>>> ut.quit_if_noshow()
>>> import wbia.plottool as pt
>>> ut.show_if_requested()
```

utool.util_alg.**upper_diag_self_prodx**(*list_*)

upper diagnoal of cartesian product of self and self. Weird name. fixme

Parameters **list** (*list*) –

Returns

Return type list

CommandLine: python -m utool.util_alg –exec-upper_diag_self_prodx

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_alg import *  # NOQA
>>> list_ = [1, 2, 3]
>>> result = upper_diag_self_prodx(list_)
>>> print(result)
[(1, 2), (1, 3), (2, 3)]
```

utool.util_alg.**xywh_to_tlbr**(*bbox*, *img_wh*)
converts xywh format to (tlx, tly, blx, bly)

1.10 utool.util_aliases module

Aliases harder to remember or very common standard module functions

1.11 utool.util_arg module

Handles command line parsing

class utool.util_arg.**ArgumentParser2**(*parser*)
Bases: object

Wrapper around argparse.ArgumentParser with convinence functions

add_arg(*switch*, **args*, ***kwargs*)

add_argument_group(**args*, ***kwargs*)

add_flag(*switch*, *default=False*, ***kwargs*)

add_float(*switch*, **args*, ***kwargs*)

add_int(*switch*, **args*, ***kwargs*)

add_intlist(*switch*, **args*, ***kwargs*)

add_ints(*switch*, **args*, ***kwargs*)

add_meta(*switch*, *type*, *default=None*, *help=''*, ***kwargs*)

add_str(*switch*, **args*, ***kwargs*)

add_strlist(*switch*, **args*, ***kwargs*)

add_strs(*switch*, **args*, ***kwargs*)

utool.util_arg.**argflag**(*argstr_*, *default=False*, *help_=''*, *return_specified=None*, *need_prefix=True*, *return_was_specified=False*, *argv=None*, *debug=None*, ***kwargs*)
Checks if the commandline has a flag or a corresponding noflag

Parameters

- **argstr** (*str, list, or tuple*) – the flag to look for
- **default** (*bool*) – dont use this (default = False)
- **help** (*str*) – a help string (default = '')
- **return_specified** (*bool*) – returns if flag was specified or not (default = False)

Returns (parsed_val, was_specified)

Return type tuple

Todo: depricate return_was_specified

CommandLine: python -m utool.util_arg –exec-get_argflag –noface –exec-mode python -m utool.util_arg –exec-get_argflag –foo –exec-mode python -m utool.util_arg –exec-get_argflag –no-foo –exec-mode python -m utool.util_arg –exec-get_argflag –foo=True –exec-mode python -m utool.util_arg –exec-get_argflag –foo=False –exec-mode

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_arg import *  # NOQA
>>> argstr_ = '--foo'
>>> default = False
>>> help_ = ''
>>> return_specified = True
>>> (parsed_val, was_specified) = get_argflag(argstr_, default, help_, return_
↪specified)
>>> result = ('(parsed_val, was_specified) = %s' % (str((parsed_val, was_
↪specified)),))
>>> print(result)
```

utool.util_arg.**argparse_dict**(*default_dict_*, *lbl=None*, *verbose=None*, *only_specified=False*, *force_keys={}*, *type_hint=None*, *alias_dict={}*)

Gets values for a dict based on the command line

Parameters

- **default_dict_** –
- **only_specified** (*bool*) – if True only returns keys that are specified on commandline. no defaults.

Returns a dictionary

Return type dict_

CommandLine: python -m utool.util_arg –test-argparse_dict python -m utool.util_arg –test-argparse_dict –foo=3 python -m utool.util_arg –test-argparse_dict –flag1 python -m utool.util_arg –test-argparse_dict –flag2 python -m utool.util_arg –test-argparse_dict –noflag2 python -m utool.util_arg –test-argparse_dict –thresh=43 python -m utool.util_arg –test-argparse_dict –bins=-10 python -m utool.util_arg –test-argparse_dict –bins=-10 –only-specified –helpx

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_arg import *  # NOQA
>>> import utool as ut
>>> # build test data
>>> default_dict_ = {
...     'bins': 8,
...     'foo': None,
...     'flag1': False,
...     'flag2': True,
...     'max': 0.2,
...     'neg': -5,
...     'thresh': -5.333,
... }
>>> # execute function
```

(continues on next page)

(continued from previous page)

```
>>> only_specified = ut.get_argflag('--only-specified')
>>> dict_ = argparse_dict(default_dict_, only_specified=only_specified)
>>> # verify results
>>> result = ut.repr4(dict_, sorted_=True)
>>> print(result)
```

utool.util_arg.**argv_flag_dec**(**argin*, ***kwargs*)

Decorators which control program flow based on sys.argv the decorated function does not execute without its corresponding flag

Kwargs: default, quiet, indent, default

ReturnKwargs: alias_flags

utool.util_arg.**argv_flag_dec_true**(*func*, ***kwargs*)

utool.util_arg.**argval**(*key*, *default=None*, *type=None*, *smartcast=True*, *return_exists=False*, *argv=None*)

alias for get_argval

Ignore:

```
>>> # ENABLE_DOCTEST
>>> import utool as ut
>>> import sys
>>> argv = ['--aids=[1,2,3]']
>>> value = ut.argval('--aids', default=[1, 2], argv=argv)
>>> assert isinstance(value, list)
>>> value2 = ut.argval('--aids', smartcast=False, argv=argv)
>>> assert isinstance(value2, str)
>>> value2 = ut.argval('--aids', smartcast=True, argv=argv)
>>> assert isinstance(value2, list)
```

utool.util_arg.**aug_sysargv**(*cmdstr*)

DEBUG FUNC modify argv to look like you ran a command

utool.util_arg.**autogen_argparse2**(*dpath_list*)

FUNCTION IS NOT FULLY IMPLEMENTED CURRENTLY ONLY RETURNS LIST OF FLAGS THAT THE PROGRAM SILENTLY TAKES

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_arg import *  # NOQA
>>> import utool as ut
>>> dpath_list = [
...     ut.truepath('~/code/utool/utool'),
...     ut.truepath('~/code/ibeis/ibeis'),
... ]
>>> flagtups_list = autogen_argparse2(dpath_list)
>>> flagtup_list_ = [ut.regex_replace('[()(\']','',tupstr) for tupstr in ut.
↪flatten(flagtups_list)]
>>> flagtup_list = ut.flatten([tupstr.split(',') for tupstr in flagtup_list_])
>>> flagtup_set = set([tupstr.strip() for tupstr in flagtup_list if tupstr.find('=
↪') == -1])
>>> print('\n'.join(flagtup_set))
```

utool.util_arg.**autogen_argparse_block**(*extra_args=[]*)

SHOULD TURN ANY REGISTERED ARGS INTO A A NEW PARSING CONFIG FILE FOR BETTER –help COMMANDS

import utool as ut __REGISTERED_ARGS__ = ut.util_arg.__REGISTERED_ARGS__

Parameters **extra_args** (*list*) – (default = [])

CommandLine: python -m utool.util_arg –test-autogen_argparse_block

Example

```
>>> # DISABLE_DOCTEST
>>> import utool as ut
>>> extra_args = []
>>> result = ut.autogen_argparse_block(extra_args)
>>> print(result)
```

utool.util_arg.**get_arg**(*argstr_*, *type_=None*, *default=None*, *help_=None*, *smartcast=True*, *return_specified=None*, *argv=None*, *verbose=None*, *debug=None*, *return_was_specified=False*, *pos=None*)

Returns a value of an argument specified on the command line after some flag

Parameters

- **argstr** (*str or tuple*) – string or tuple of strings denoting the command line values to parse
- **type** (*None*) – type of the variable to parse (default = None)
- **default** (*None*) – (default = None)
- **help** (*None*) – help for this argument (not fully integrated) (default = None)
- **smartcast** (*bool*) – tries to be smart about casting the parsed strings (default = True)
- **return_specified** (*bool*) – (default = False)
- **argv** (*None*) – override sys.argv with custom command line vector (default = None)
- **pos** (*int*) – if specified the argument can also be found in position *pos* of the command line varargs

---

Todo: depricate return_was_specified

CommandLine: python -m utool.util_arg –test-get_argval python -m utool.util_arg –exec-get_argval:0 python -m utool.util_arg –exec-get_argval:1 python -c “import utool; print([(type(x), x) for x in [utool.get_argval(’–quest’)]])” –quest=”holy grail” python -c “import utool; print([(type(x), x) for x in [utool.get_argval(’–quest’)]])” –quest=”42” python -c “import utool; print([(type(x), x) for x in [utool.get_argval(’–quest’)]])” –quest=42 python -c “import utool; print([(type(x), x) for x in [utool.get_argval(’–quest’)]])” –quest 42 python -c “import utool; print([(type(x), x) for x in [utool.get_argval(’–quest’, float)]])” –quest 42 python -c “import utool; print([(type(x), x) for x in [utool.get_argval((’–nAssign’), int)]])” –nAssign 42 python -c “import utool; print([(type(x), x) for x in [utool.get_argval((’–test’), str)]])” –test python -c “import utool; print([(type(x), x) for x in [utool.get_argval((’–test’), str)]])” –test “foobar is good” –youbar ok

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_arg import *  # NOQA
>>> import utool as ut
>>> import sys
>>> argv = ['--spam', 'eggs', '--quest=holy grail', '--ans=42', '--the-val=1,2,3']
>>> # specify a list of args and kwargs to get_argval
>>> argstr_kwargs_list = [
>>>     ('--spam',                    dict(type_=str, default=None, argv=argv)),
>>>     ('--quest',                   dict(type_=str, default=None, argv=argv)),
>>>     (('--ans', '--foo'),          dict(type_=int, default=None, argv=argv)),
>>>     (('--not-there', '--absent'), dict(argv=argv)),
>>>     ('--the_val',                 dict(type_=list, argv=argv)),
>>>     ('--the-val',                 dict(type_=list, argv=argv)),
>>> ]
>>> # Execute the command with for each of the test cases
>>> res_list = []
>>> argstr_list = ut.get_list_column(argstr_kwargs_list, 0)
>>> for argstr_, kwargs in argstr_kwargs_list:
>>>     res = get_argval(argstr_, **kwargs)
>>>     res_list.append(res)
>>> result = ut.repr2(ut.odict(zip(argstr_list, res_list)), nl=1)
>>> result = result.replace('u\'', '\'')  # hack
>>> print(result)
{
    '--spam': 'eggs',
    '--quest': 'holy grail',
    ('--ans', '--foo'): 42,
    ('--not-there', '--absent'): None,
    '--the_val': [1, 2, 3],
    '--the-val': [1, 2, 3],
}
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_arg import *  # NOQA
>>> import utool as ut
>>> import sys
>>> argv = ['--slice1', '::', '--slice2=4:', '--slice3=::4', '--slice4', '[1,2,3,
↪4]', '--slice5=3']
>>> # specify a list of args and kwargs to get_argval
>>> argstr_kwargs_list = [
>>>     ('--slice1',            dict(type_='fuzzy_subset', default=None,␣
↪argv=argv)),
>>>     ('--slice2',            dict(type_='fuzzy_subset', default=None,␣
↪argv=argv)),
>>>     ('--slice3',            dict(type_='fuzzy_subset', default=None,␣
↪argv=argv)),
>>>     ('--slice4',            dict(type_='fuzzy_subset', default=None,␣
↪argv=argv)),
>>>     ('--slice5',            dict(type_='fuzzy_subset', default=None,␣
↪argv=argv)),
>>> ]
>>> # Execute the command with for each of the test cases
```

(continues on next page)

(continued from previous page)

```
>>> res_list = []
>>> argstr_list = ut.get_list_column(argstr_kwargs_list, 0)
>>> list1 = [1, 3, 5, 7, 9]
>>> import numpy as np
>>> list2 = np.array([[1, 2], [3, 4], [5, 6], [7, 8], [9, 1]])
>>> for argstr_, kwargs in argstr_kwargs_list:
>>>     res = get_argval(argstr_, **kwargs)
>>>     print('---')
>>>     print('res = %r' % (res,))
>>>     print('list1[%r=%r] = %r' % (argstr_, res, ut.take(list1, res),))
>>>     print('list2[%r=%r] = %r' % (argstr_, res, list2[res].tolist(),))
>>>     res_list.append(res)
>>> result = ut.repr4(ut.odict(zip(argstr_list, res_list)))
>>> result = result.replace('u\'', '\'')  # hack
>>> print(result)
```

utool.util_arg.**get_arg_dict**(*argv=None, prefix_list=['--']*, *type_hints={}*)

Yet another way for parsing args

CommandLine: python -m utool.util_arg --exec-get_arg_dict python -m utool.util_arg --test-get_arg_dict

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_arg import *  # NOQA
>>> import utool as ut
>>> import shlex
>>> argv = shlex.split('--test-show_name --name=IBEIS_PZ_0303 --db testdb3 --save
↪"~/latex/crall-candidacy-2015/figures/IBEIS_PZ_0303.jpg" --dpath figures --
↪caption="Shadowed"  --figsize=11,3 --no-figtitle -t foo bar baz biz --notitle')
>>> arg_dict = ut.get_arg_dict(argv, prefix_list=['--', '-'], type_hints={'t':␣
↪list})
>>> result = ut.repr2(arg_dict, nl=1)
>>> # verify results
>>> print(result)
{
    'caption': 'Shadowed',
    'db': 'testdb3',
    'dpath': 'figures',
    'figsize': '11,3',
    'name': 'IBEIS_PZ_0303',
    'no-figtitle': True,
    'notitle': True,
    'save': '~/latex/crall-candidacy-2015/figures/IBEIS_PZ_0303.jpg',
    't': ['foo', 'bar', 'baz', 'biz'],
    'test-show_name': True,
}
```

utool.util_arg.**get_argflag**(*argstr_*, *default=False*, *help_=''*, *return_specified=None*, *need_prefix=True*, *return_was_specified=False*, *argv=None*, *debug=None*, ***kwargs*)

Checks if the commandline has a flag or a corresponding noflag

Parameters

- **argstr** (*str, list, or tuple*) – the flag to look for

- **default** (*bool*) – dont use this (default = False)
- **help** (*str*) – a help string (default = '')
- **return_specified** (*bool*) – returns if flag was specified or not (default = False)

Returns (parsed_val, was_specified)

Return type tuple

Todo: depricate return_was_specified

CommandLine: python -m utool.util_arg –exec-get_argflag –noface –exec-mode python -m utool.util_arg –exec-get_argflag –foo –exec-mode python -m utool.util_arg –exec-get_argflag –no-foo –exec-mode python -m utool.util_arg –exec-get_argflag –foo=True –exec-mode python -m utool.util_arg –exec-get_argflag –foo=False –exec-mode

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_arg import *  # NOQA
>>> argstr_ = '--foo'
>>> default = False
>>> help_ = ''
>>> return_specified = True
>>> (parsed_val, was_specified) = get_argflag(argstr_, default, help_, return_
↪specified)
>>> result = ('(parsed_val, was_specified) = %s' % (str((parsed_val, was_
↪specified)),))
>>> print(result)
```

utool.util_arg.**get_argv_tail**(*scriptname*, *prefer_main=None*, *argv=None*)

gets the rest of the arguments after a script has been invoked hack. accounts for python -m scripts.

Parameters **scriptname** (*str*) –

CommandLine: python -m utool.util_arg –test-get_argv_tail

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_arg import *  # NOQA
>>> import utool as ut
>>> from os.path import relpath, dirname
>>> scriptname = 'utool.util_arg'
>>> prefer_main = False
>>> argv=['python', '-m', 'utool.util_arg', '--test-get_argv_tail']
>>> tail = get_argv_tail(scriptname, prefer_main, argv)
>>> # hack
>>> tail[0] = ut.ensure_unixslash(relpath(tail[0], dirname(dirname(ut.__file__))))
>>> result = ut.repr2(tail)
>>> print(result)
['utool/util_arg.py', '--test-get_argv_tail']
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_arg import *  # NOQA
>>> import utool as ut
>>> from os.path import relpath, dirname
>>> scriptname = 'utprof.py'
>>> prefer_main = True
>>> argv=['utprof.py', '-m', 'utool', '--tf', 'get_argv_tail']
>>> tail = get_argv_tail(scriptname, prefer_main, argv)
>>> # hack
>>> tail[0] = ut.ensure_unixslash(relpath(tail[0], dirname(dirname(ut.__file__))))
>>> result = ut.repr2(tail)
>>> print(result)
['utool/__main__.py', '--tf', 'get_argv_tail']
```

utool.util_arg.**get_argval**(*argstr_*, *type_=None*, *default=None*, *help_=None*, *smartcast=True*, *return_specified=None*, *argv=None*, *verbose=None*, *debug=None*, *return_was_specified=False*, *pos=None*)

Returns a value of an argument specified on the command line after some flag

Parameters

- **argstr** (*str or tuple*) – string or tuple of strings denoting the command line values to parse
- **type** (*None*) – type of the variable to parse (default = None)
- **default** (*None*) – (default = None)
- **help** (*None*) – help for this argument (not fully integrated) (default = None)
- **smartcast** (*bool*) – tries to be smart about casting the parsed strings (default = True)
- **return_specified** (*bool*) – (default = False)
- **argv** (*None*) – override sys.argv with custom command line vector (default = None)
- **pos** (*int*) – if specified the argument can also be found in position *pos* of the command line varargs

---

Todo: depricate return_was_specified

CommandLine: python -m utool.util_arg –test-get_argval python -m utool.util_arg –exec-get_argval:0 python -m utool.util_arg –exec-get_argval:1 python -c “import utool; print([(type(x), x) for x in [utool.get_argval(’–quest’)]])” –quest=”holy grail” python -c “import utool; print([(type(x), x) for x in [utool.get_argval(’–quest’)]])” –quest=”42” python -c “import utool; print([(type(x), x) for x in [utool.get_argval(’–quest’)]])” –quest=42 python -c “import utool; print([(type(x), x) for x in [utool.get_argval(’–quest’)]])” –quest 42 python -c “import utool; print([(type(x), x) for x in [utool.get_argval(’–quest’, float)]])” –quest 42 python -c “import utool; print([(type(x), x) for x in [utool.get_argval((’–nAssign’), int)]])” –nAssign 42 python -c “import utool; print([(type(x), x) for x in [utool.get_argval((’–test’), str)]])” –test python -c “import utool; print([(type(x), x) for x in [utool.get_argval((’–test’), str)]])” –test “foobar is good” –youbar ok

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_arg import *  # NOQA
>>> import utool as ut
>>> import sys
>>> argv = ['--spam', 'eggs', '--quest=holy grail', '--ans=42', '--the-val=1,2,3']
>>> # specify a list of args and kwargs to get_argval
>>> argstr_kwargs_list = [
>>>     ('--spam',                    dict(type_=str, default=None, argv=argv)),
>>>     ('--quest',                   dict(type_=str, default=None, argv=argv)),
>>>     (('--ans', '--foo'),          dict(type_=int, default=None, argv=argv)),
>>>     (('--not-there', '--absent'), dict(argv=argv)),
>>>     ('--the_val',                 dict(type_=list, argv=argv)),
>>>     ('--the-val',                 dict(type_=list, argv=argv)),
>>> ]
>>> # Execute the command with for each of the test cases
>>> res_list = []
>>> argstr_list = ut.get_list_column(argstr_kwargs_list, 0)
>>> for argstr_, kwargs in argstr_kwargs_list:
>>>     res = get_argval(argstr_, **kwargs)
>>>     res_list.append(res)
>>> result = ut.repr2(ut.odict(zip(argstr_list, res_list)), nl=1)
>>> result = result.replace('u\'', '\'')  # hack
>>> print(result)
{
    '--spam': 'eggs',
    '--quest': 'holy grail',
    ('--ans', '--foo'): 42,
    ('--not-there', '--absent'): None,
    '--the_val': [1, 2, 3],
    '--the-val': [1, 2, 3],
}
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_arg import *  # NOQA
>>> import utool as ut
>>> import sys
>>> argv = ['--slice1', '::', '--slice2=4:', '--slice3=::4', '--slice4', '[1,2,3,
↪4]', '--slice5=3']
>>> # specify a list of args and kwargs to get_argval
>>> argstr_kwargs_list = [
>>>     ('--slice1',            dict(type_='fuzzy_subset', default=None,␣
↪argv=argv)),
>>>     ('--slice2',            dict(type_='fuzzy_subset', default=None,␣
↪argv=argv)),
>>>     ('--slice3',            dict(type_='fuzzy_subset', default=None,␣
↪argv=argv)),
>>>     ('--slice4',            dict(type_='fuzzy_subset', default=None,␣
↪argv=argv)),
>>>     ('--slice5',            dict(type_='fuzzy_subset', default=None,␣
↪argv=argv)),
>>> ]
>>> # Execute the command with for each of the test cases
```

(continues on next page)

(continued from previous page)

```
>>> res_list = []
>>> argstr_list = ut.get_list_column(argstr_kwargs_list, 0)
>>> list1 = [1, 3, 5, 7, 9]
>>> import numpy as np
>>> list2 = np.array([[1, 2], [3, 4], [5, 6], [7, 8], [9, 1]])
>>> for argstr_, kwargs in argstr_kwargs_list:
>>>     res = get_argval(argstr_, **kwargs)
>>>     print('---')
>>>     print('res = %r' % (res,))
>>>     print('list1[%r=%r] = %r' % (argstr_, res, ut.take(list1, res),))
>>>     print('list2[%r=%r] = %r' % (argstr_, res, list2[res].tolist(),))
>>>     res_list.append(res)
>>> result = ut.repr4(ut.odict(zip(argstr_list, res_list)))
>>> result = result.replace('u\'', '\'')  # hack
>>> print(result)
```

utool.util_arg.**get_cmdline_varargs**(*argv=None*)

Returns positional args specified directly after the scriptname and before any args starting with '-' on the commandline.

utool.util_arg.**get_dict_vals_from_commandline**(*default_dict_*, *lbl=None*, *verbose=None*, *only_specified=False*, *force_keys={}*, *type_hint=None*, *alias_dict={}*)

Gets values for a dict based on the command line

Parameters

- **default_dict_** –
- **only_specified** (*bool*) – if True only returns keys that are specified on commandline. no defaults.

Returns a dictionary

Return type dict_

CommandLine: python -m utool.util_arg –test-argparse_dict python -m utool.util_arg –test-argparse_dict –foo=3 python -m utool.util_arg –test-argparse_dict –flag1 python -m utool.util_arg –test-argparse_dict –flag2 python -m utool.util_arg –test-argparse_dict –noflag2 python -m utool.util_arg –test-argparse_dict –thresh=43 python -m utool.util_arg –test-argparse_dict –bins=-10 python -m utool.util_arg –test-argparse_dict –bins=-10 –only-specified –helpx

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_arg import *  # NOQA
>>> import utool as ut
>>> # build test data
>>> default_dict_ = {
...     'bins': 8,
...     'foo': None,
...     'flag1': False,
...     'flag2': True,
...     'max': 0.2,
...     'neg': -5,
...     'thresh': -5.333,
```

(continues on next page)

(continued from previous page)

```
... }
>>> # execute function
>>> only_specified = ut.get_argflag('--only-specified')
>>> dict_ = argparse_dict(default_dict_, only_specified=only_specified)
>>> # verify results
>>> result = ut.repr4(dict_, sorted_=True)
>>> print(result)
```

utool.util_arg.**get_flag**(*argstr_*, *default=False*, *help_=''*, *return_specified=None*, *need_prefix=True*, *return_was_specified=False*, *argv=None*, *debug=None*, ***kwargs*)

Checks if the commandline has a flag or a corresponding noflag

Parameters

- **argstr** (*str, list, or tuple*) – the flag to look for
- **default** (*bool*) – dont use this (default = False)
- **help** (*str*) – a help string (default = '')
- **return_specified** (*bool*) – returns if flag was specified or not (default = False)

Returns (parsed_val, was_specified)

Return type tuple

Todo: depricate return_was_specified

CommandLine: python -m utool.util_arg –exec-get_argflag –noface –exec-mode python -m utool.util_arg –exec-get_argflag –foo –exec-mode python -m utool.util_arg –exec-get_argflag –no-foo –exec-mode python -m utool.util_arg –exec-get_argflag –foo=True –exec-mode python -m utool.util_arg –exec-get_argflag –foo=False –exec-mode

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_arg import *  # NOQA
>>> argstr_ = '--foo'
>>> default = False
>>> help_ = ''
>>> return_specified = True
>>> (parsed_val, was_specified) = get_argflag(argstr_, default, help_, return_
↪specified)
>>> result = ('(parsed_val, was_specified) = %s' % (str((parsed_val, was_
↪specified)),))
>>> print(result)
```

utool.util_arg.**get_fpath_args**(*arglist_=None*, *pat='*'*)

utool.util_arg.**get_module_verbosity_flags**(**labels*)

checks for standard flags for enableing module specific verbosity

utool.util_arg.**get_varargs**(*argv=None*)

Returns positional args specified directly after the scriptname and before any args starting with '-' on the commandline.

utool.util_arg.**get_verbflag**(**labels*)
checks for standard flags for enableing module specific verbosity

utool.util_arg.**make_argparse2**(*prog='Program'*, *description=''*, **args*, ***kwargs*)

utool.util_arg.**parse_arglist_hack**(*argx*, *argv=None*)

utool.util_arg.**parse_cfgstr_list**(*cfgstr_list*, *smartcast=True*, *oldmode=True*)
Parses a list of items in the format ['var1:val1', 'var2:val2', 'var3:val3'] the '=' character can be used instead of the ':' character if desired

TODO: see ut.parse_cfgstr3

Parameters **cfgstr_list** (*list*) –

Returns cfgdict

Return type dict

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_arg import *  # NOQA
>>> import utool as ut
>>> cfgstr_list = ['var1=val1', 'var2=1', 'var3=1.0', 'var4=None', 'var5=[1,2,3]',
↪ 'var6=(a,b,c)']
>>> smartcast = True
>>> cfgdict = parse_cfgstr_list(cfgstr_list, smartcast, oldmode=False)
>>> result = ut.repr2(cfgdict, sorted_=True, newlines=False)
>>> print(result)
{'var1': 'val1', 'var2': 1, 'var3': 1.0, 'var4': None, 'var5': [1, 2, 3], 'var6':␣
↪('a', 'b', 'c')}
```

{'var1': 'val1', 'var2': 1, 'var3': 1.0, 'var4': None}

{'var4': None, 'var1': 'val1', 'var3': 1.0, 'var2': 1}

utool.util_arg.**parse_dict_from_argv**(*default_dict_*, *lbl=None*, *verbose=None*, *only_specified=False*, *force_keys={}*, *type_hint=None*, *alias_dict={}*)
Gets values for a dict based on the command line

Parameters

- **default_dict_** –
- **only_specified** (*bool*) – if True only returns keys that are specified on commandline. no defaults.

Returns a dictionary

Return type dict_

CommandLine: python -m utool.util_arg –test-argparse_dict python -m utool.util_arg –test-argparse_dict –foo=3 python -m utool.util_arg –test-argparse_dict –flag1 python -m utool.util_arg –test-argparse_dict –flag2 python -m utool.util_arg –test-argparse_dict –noflag2 python -m utool.util_arg –test-argparse_dict –thresh=43 python -m utool.util_arg –test-argparse_dict –bins=-10 python -m utool.util_arg –test-argparse_dict –bins=-10 –only-specified –helpx

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_arg import *  # NOQA
>>> import utool as ut
>>> # build test data
>>> default_dict_ = {
...     'bins': 8,
...     'foo': None,
...     'flag1': False,
...     'flag2': True,
...     'max': 0.2,
...     'neg': -5,
...     'thresh': -5.333,
... }
>>> # execute function
>>> only_specified = ut.get_argflag('--only-specified')
>>> dict_ = argparse_dict(default_dict_, only_specified=only_specified)
>>> # verify results
>>> result = ut.repr4(dict_, sorted_=True)
>>> print(result)
```

utool.util_arg.**reset_argrecord**()
forgets about the args already parsed

utool.util_arg.**switch_sanataize**(*switch*)

1.12 utool.util_assert module

utool.util_assert.**assert_all_eq**(*item_list*, *eq_=<built-in function eq>*)

utool.util_assert.**assert_all_in**(*key_list*, *valid_list*, *msg=''*)

utool.util_assert.**assert_all_not_None**(*list_*, *list_name='some_list'*, *key_list=[]*, *verbose=True*, *veryverbose=False*)

utool.util_assert.**assert_almost_eq**(*arr_test*, *arr_target*, *thresh=1e-11*)

Parameters

- **arr_test** (*ndarray or list*) –
- **arr_target** (*ndarray or list*) –
- **thresh** (*scalar or ndarray or list*) –

utool.util_assert.**assert_eq**(*var1*, *var2*, *msg=''*, *var1_name=None*, *var2_name=None*, *verbose=None*)

utool.util_assert.**assert_eq_len**(*list1*, *list2*, *additional_msg=''*)

utool.util_assert.**assert_inbounds**(*num*, *low*, *high*, *msg=''*, *eq=False*, *verbose=True*)

Parameters

- **num** (*scalar*) –
- **low** (*scalar*) –
- **high** (*scalar*) –
- **msg** (*str*) –

utool.util_assert.**assert_lessthan**(*arr_test*, *arr_max*, *msg=''*)

Parameters

- **arr_test** (*ndarray or list*) –
- **arr_target** (*ndarray or list*) –
- **thresh** (*scalar or ndarray or list*) –

utool.util_assert.**assert_lists_eq**(*list1*, *list2*, *failmsg=''*, *verbose=False*)

utool.util_assert.**assert_raises**(*ex_type*, *func*, **args*, ***kwargs*)

Checks that a function raises an error when given specific arguments.

Parameters

- **ex_type** (*Exception*) – exception type
- **func** (*callable*) – live python function

CommandLine: python -m utool.util_assert assert_raises –show

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_assert import *  # NOQA
>>> import utool as ut
>>> ex_type = AssertionError
>>> func = len
>>> # Check that this raises an error when something else does not
>>> assert_raises(ex_type, assert_raises, ex_type, func, [])
>>> # Check this does not raise an error when something else does
>>> assert_raises(ValueError, [].index, 0)
```

utool.util_assert.**assert_same_len**(*list1*, *list2*, *additional_msg=''*)

utool.util_assert.**assert_scalar_list**(*list_*)

utool.util_assert.**assert_unflat_level**(*unflat_list*, *level=1*, *basetype=None*)

utool.util_assert.**assert_unique**(*item_list*, *ignore=[]*, *name='list'*, *verbose=None*)

utool.util_assert.**get_first_None_position**(*list_*)

utool.util_assert.**lists_eq**(*list1*, *list2*)

recursive

1.13 utool.util_autogen module

class utool.util_autogen.**PythonStatement**(*stmt*)

Bases: object

Thin wrapper around a string representing executable python code

utool.util_autogen.**auto_docstr**(*modname*, *funcname*, *verbose=True*, *moddir=None*, *modpath=None*, ***kwargs*)

called from vim. Uses strings of filename and modnames to build docstr

Parameters

- **modname** (*str*) – name of a python module
- **funcname** (*str*) – name of a function in the module

Returns docstr

Return type str

CommandLine: python -m utool.util_autogen auto_docstr python -m utool –tf auto_docstr

Example

```
>>> # DISABLE_DOCTEST
>>> import utool as ut
>>> from utool.util_autogen import *  # NOQA
>>> ut.util_autogen.rrr(verbose=False)
>>> #docstr = ut.auto_docstr('wbia.algo.hots.smk.smk_index', 'compute_negentropy_
↪names')
>>> modname = ut.get_argval('--modname', default='utool.util_autogen')
>>> funcname = ut.get_argval('--funcname', default='auto_docstr')
>>> moddir = ut.get_argval('--moddir', type_=str, default=None)
>>> docstr = ut.util_autogen.auto_docstr(modname, funcname)
>>> print(docstr)
```

utool.util_autogen.**autofix_codeblock**(*codeblock*, *max_line_len=80*, *aggressive=False*, *very_aggressive=False*, *experimental=False*)

Uses autopep8 to format a block of code

Example

```
>>> # DISABLE_DOCTEST
>>> import utool as ut
>>> codeblock = ut.codeblock(
>>> '''
>>> def func( with , some = 'Problems' ):
>>>
>>>
>>>  syntax ='Ok'
>>>  but = 'Its very messy'
>>>  if None:
>>>         # syntax might not be perfect due to being cut off
>>>         ommiting_this_line_still_works=   True
>>> ''')
>>> fixed_codeblock = ut.autofix_codeblock(codeblock)
>>> print(fixed_codeblock)
```

utool.util_autogen.**dump_autogen_code**(*fpath*, *autogen_text*, *codetype='python'*, *fullprint=None*, *show_diff=None*, *dowrite=None*)

Helper that write a file if -w is given on command line, otherwise it just prints it out. It has the opption of comparing a diff to the file.

utool.util_autogen.**find_modname_in_pythonpath**(*modname*)

utool.util_autogen.**is_modname_in_pythonpath**(*modname*)

utool.util_autogen.**load_func_from_module**(*modname*, *funcname*, *verbose=True*, *moddir=None*, *modpath=None*)

Parameters

- **modname** (*str*) – module name
- **funcname** (*str*) – function name
- **verbose** (*bool*) – verbosity flag(Defaults to True)
- **moddir** (*None*) – (Defaults to None)

CommandLine: python -m utool.util_autogen load_func_from_module

Example

```
>>> # DISABLE_DOCTEST
>>> # UNSTABLE_DOCTEST
>>> from utool.util_autogen import *  # NOQA
>>> import utool as ut
>>> #funcname = 'multi_plot'
>>> modname = 'utool.util_path'
>>> funcname = 'checkpath'
>>> verbose = True
>>> moddir = None
>>> func, module, error_str = load_func_from_module(modname, funcname, verbose,␣
↪moddir)
>>> source = ut.get_func_sourcecode(func, strip_docstr=True, strip_comments=True)
>>> keyname = ut.named_field('keyname', ut.REGEX_VARNAME)
>>> default = ut.named_field('default', '[\'\"A-Za-z_][A-Za-z0-9_\'\"]*')
>>> pattern = re.escape('kwargs.get(\'') + keyname + re.escape('\',')
>>> kwarg_keys = [match.groupdict()['keyname'] for match in re.finditer(pattern,␣
↪source)]
```

utool.util_autogen.**make_args_docstr**(*argname_list*, *argtype_list*, *argdesc_list*, *ismethod*, *va_name=None*, *kw_name=None*, *kw_keys=[]*)

Builds the argument docstring

Parameters

- **argname_list** (*list*) – names
- **argtype_list** (*list*) – types
- **argdesc_list** (*list*) – descriptions
- **ismethod** (*bool*) – if generating docs for a method
- **va_name** (*Optional[str]*) – varargs name
- **kw_name** (*Optional[str]*) – kwargs name
- **kw_keys** (*Optional[list]*) – accepted kwarg keys

Returns arg_docstr

Return type str

CommandLine: python -m utool.util_autogen make_args_docstr

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_autogen import *  # NOQA
>>> argname_list = ['argname_list', 'argtype_list', 'argdesc_list']
>>> argtype_list = ['list', 'list', 'list']
>>> argdesc_list = ['names', 'types', 'descriptions']
>>> va_name = 'args'
>>> kw_name = 'kwargs'
>>> kw_keys = ['']
>>> ismethod = False
>>> arg_docstr = make_args_docstr(argname_list, argtype_list,
>>>                               argdesc_list, ismethod, va_name,
>>>                               kw_name, kw_keys)
>>> result = str(arg_docstr)
>>> print(result)
argname_list (list): names
argtype_list (list): types
argdesc_list (list): descriptions
*args:
**kwargs:
```

utool.util_autogen.**make_cmdline_docstr**(*funcname*, *modname*)

utool.util_autogen.**make_default_docstr**(*func*, *with_args=True*, *with_ret=True*, *with_commandline=True*, *with_example=True*, *with_header=False*, *with_debug=False*)

Tries to make a sensible default docstr so the user can fill things in without typing too much

TODO: Interleave old documentation with new documentation

Parameters

- **func** (*function*) – live python function
- **with_args** (*bool*) –
- **with_ret** (*bool*) – (Defaults to True)
- **with_commandline** (*bool*) – (Defaults to True)
- **with_example** (*bool*) – (Defaults to True)
- **with_header** (*bool*) – (Defaults to False)
- **with_debug** (*bool*) – (Defaults to False)

Returns (argname, val)

Return type tuple

Ignore: pass

CommandLine: python -m utool.util_autogen –exec-make_default_docstr –show

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_autogen import *  # NOQA
>>> import utool as ut
```

(continues on next page)

(continued from previous page)

```
>>> func = ut.make_default_docstr
>>> #func = ut.make_args_docstr
>>> #func = PythonStatement
>>> func = auto_docstr
>>> default_docstr = make_default_docstr(func)
>>> result = str(default_docstr)
>>> print(result)
```

utool.util_autogen.**make_default_module_maintest**(*modname*, *modpath=None*, *test_code=None*)

DEPRICATE

TODO: use path relative to home dir if the file is a script

Parameters **modname** (*str*) – module name

Returns text source code

Return type str

CommandLine: python -m utool.util_autogen –test-make_default_module_maintest

References

http://legacy.python.org/dev/peps/pep-0338/

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_autogen import *  # NOQA
>>> modname = 'utool.util_autogen'
>>> text = make_default_module_maintest(modname)
>>> result = str(text)
>>> print(result)
```

utool.util_autogen.**make_docstr_block**(*header*, *block*)

utool.util_autogen.**make_example_docstr**(*funcname=None*, *modname=None*, *argname_list=None*, *defaults=None*, *return_type=None*, *return_name=None*, *ismethod=False*)

Creates skeleton code to build an example doctest

Parameters

- **funcname** (*str*) – function name
- **modname** (*str*) – module name
- **argname_list** (*str*) – list of argument names
- **defaults** (*None*) –
- **return_type** (*None*) –
- **return_name** (*str*) – return variable name
- **ismethod** (*bool*) –

Returns examplecode

Return type str

CommandLine: python -m utool.util_autogen –test-make_example_docstr

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_autogen import *  # NOQA
>>> funcname = 'make_example_docstr'
>>> modname = 'utool.util_autogen'
>>> argname_list = ['qaids', 'qreq_']
>>> defaults = None
>>> return_type = tuple
>>> return_name = 'foo'
>>> ismethod = False
>>> examplecode = make_example_docstr(funcname, modname, argname_list, defaults,␣
↪return_type, return_name, ismethod)
>>> result = str(examplecode)
>>> print(result)
# DISABLE_DOCTEST
from utool.util_autogen import *  # NOQA
import utool as ut
import wbia
ibs = wbia.opendb(defaultdb='testdb1')
species = wbia.const.TEST_SPECIES.ZEB_PLAIN
qaids = ibs.get_valid_aids(species=species)
qreq_ = wbia.testdata_qreq_()
foo = make_example_docstr(qaids, qreq_)
result = ('foo = %s' % (ut.repr2(foo),))
print(result)
```

utool.util_autogen.**make_returns_or_yeilds_docstr**(*return_type*, *return_name*, *return_desc*)

utool.util_autogen.**makeinit**(*mod_dpath*, *exclude_modnames=[]*, *use_star=False*)

Parameters

- **mod_dpath** (*str*) –
- **exclude_modnames** (*list*) – (Defaults to [])
- **use_star** (*bool*) – (Defaults to False)

Returns init_codeblock

Return type str

CommandLine: python -m utool.util_autogen makeinit –modname=wbia.algo

Example

```
>>> # DISABLE_DOCTEST
>>> # SCRIPT
>>> from utool.util_autogen import *  # NOQA
>>> import utool as ut
>>> modname = ut.get_argval('--modname', str, default=None)
```

(continues on next page)

(continued from previous page)

```
>>> mod_dpath = (os.getcwd() if modname is None else
>>>              ut.get_modpath(modname, prefer_pkg=True))
>>> mod_dpath = ut.unixpath(mod_dpath)
>>> mod_fpath = join(mod_dpath, '__init__.py')
>>> exclude_modnames = ut.get_argval(('--exclude', '-x'), list, default=[])
>>> use_star = ut.get_argflag('--star')
>>> init_codeblock = makeinit(mod_dpath, exclude_modnames, use_star)
>>> ut.dump_autogen_code(mod_fpath, init_codeblock)
```

utool.util_autogen.**print_auto_docstr**(*modname*, *funcname*)
python -c "import utool; utool.print_auto_docstr('wbia.algo.hots.smk.smk_index', 'compute_negentropy_names')" python -c "import utool; utool.print_auto_docstr('wbia.algo.hots.smk.smk_index', 'compute_negentropy_names')"

utool.util_autogen.**remove_codeblock_syntax_sentinals**(*code_text*)
Removes template comments and vim sentinals

Parameters **code_text** (*str*) –

Returns code_text_

Return type str

utool.util_autogen.**write_modscript_alias**(*fpath*, *modname*, *args=''*, *pyscript='python'*)
convinience function because $@ is annoying to paste into the terminal

1.14 utool.util_cache module

This module needs serious refactoring and testing

class utool.util_cache.**Cachable**
Bases: object

Abstract base class.

This class which enables easy caching of object dictionarys

must implement get_cfgstr()

delete(*cachedir=None*, *cfgstr=None*, *verbose=True*)
saves query result to directory

ext = '.cPkl'

fuzzyload(*cachedir=None*, *partial_cfgstr=''*, ***kwargs*)
Try and load from a partially specified configuration string

get_cachedir(*cachedir=None*)

get_cfgstr()

get_fname(*cfgstr=None*, *ext=None*)

get_fpath(*cachedir=None*, *cfgstr=None*, *ext=None*)

Ignore: fname = _fname cfgstr = _cfgstr

get_prefix()

glob_valid_targets(*cachedir=None*, *partial_cfgstr=''*)

load(*cachedir=None*, *cfgstr=None*, *fpath=None*, *verbose=None*, *quiet=False*, *ignore_keys=None*)
Loads the result from the given database

save(*cachedir=None*, *cfgstr=None*, *verbose=False*, *quiet=False*, *ignore_keys=None*)
saves query result to directory

exception utool.util_cache.**CacheMissException**
Bases: Exception

class utool.util_cache.**Cacher**(*fname*, *cfgstr=None*, *cache_dir='default'*, *appname='utool'*, *ext='.cPkl'*, *verbose=None*, *enabled=True*)
Bases: object

old non inhertable version of cachable

ensure(*func*, **args*, ***kwargs*)

existing_versions()
Returns data with different cfgstr values that were previously computed with this cacher.

exists(*cfgstr=None*)

get_fpath()

load(*cfgstr=None*)

save(*data*, *cfgstr=None*)

tryload(*cfgstr=None*)
Like load, but returns None if the load fails

class utool.util_cache.**GlobalShelfContext**(*appname*)
Bases: object

older class. might need update

class utool.util_cache.**KeyedDefaultDict**(*default_func*, **args*, ***kwargs*)
Bases: *utool.util_dict.DictLike*

getitem(*key*)

keys()

setitem(*key*, *value*)

values()

class utool.util_cache.**LRUDict**(*max_size*)
Bases: object

Pure python implementation for lru cache fallback

References

http://www.kunxi.org/blog/2014/05/lru-cache-in-python/

Parameters **max_size** (*int*) – (default = 5)

Returns cache_obj

Return type *LRUDict*

CommandLine: python -m utool.util_cache –test-LRUDict

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_cache import *  # NOQA
>>> max_size = 5
>>> self = LRUDict(max_size)
>>> for count in range(0, 5):
...     self[count] = count
>>> print(self)
>>> self[0]
>>> for count in range(5, 8):
...     self[count] = count
>>> print(self)
>>> del self[5]
>>> assert 4 in self
>>> result = ('self = %r' % (self,))
>>> print(result)
self = LRUDict({
    4: 4,
    0: 0,
    6: 6,
    7: 7,
})
```

clear()

has_key(*item*)

items()

iteritems()

iterkeys()

itervalues()

keys()

values()

class utool.util_cache.**LazyDict**(*other=None*, *is_eager=True*, *verbose=False*, *reprkw=None*, *mutable=False*, ***kwargs*)

Bases: object

Hacky dictionary where values that are functions are counted as lazy

CommandLine: python -m utool.util_cache –exec-LazyDict

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_cache import *  # NOQA
>>> import utool as ut
>>> self = ut.LazyDict()
>>> self['foo'] = lambda: 5
>>> self['bar'] = 4
>>> try:
>>>     self['foo'] = lambda: 9
>>>     assert False, 'should not be able to override computable functions'
>>> except ValueError:
```

(continues on next page)

(continued from previous page)

```
>>>     pass
>>> self['biz'] = lambda: 9
>>> d = {}
>>> d.update(**self)
>>> self['spam'] = lambda: 'eggs'
>>> self.printinfo()
>>> print(self.tostring(is_eager=False))
```

all_keys()

asdict(*is_eager=None*)

cached_keys()
only keys whose vals that have been explicitly set without a backup func

clear_evaluated()

clear_stored(*keys=None*)

eager_eval(*key*)

evaluated_keys()
only keys whose vals have been evaluated from a stored function

get(*key*, **d*)

getitem(*key*, *is_eager=None*)

items()

keys()

lazy_eval(*key*)

nocache_eval(*key*)
forces function evaluation

nonreconstructable_keys()
only keys whose vals that have been explicitly set without a backup func

printinfo()

reconstructable_keys()
only keys whose vals that have been set with a backup func

rrr(*verbose=True*, *reload_module=True*)
special class reloading function This function is often injected as rrr of classes

set_lazy_func(*key*, *func*)

setitem(*key*, *value*)

stored_keys()
keys whose vals that have been explicitly set or evaluated

tostring(*is_eager=None*, *keys=None*, ***kwargs*)

unevaluated_keys()
keys whose vals can be constructed but have not been

update(*dict_*, ***kwargs*)

values()

class utool.util_cache.**LazyList**(***kwargs*)
Bases: object

very hacky list implemented as a dictionary

append(*item*)

rrr(*verbose=True*, *reload_module=True*)
special class reloading function This function is often injected as rrr of classes

tolist()

class utool.util_cache.**ShelfCacher**(*fpath*, *enabled=True*)
Bases: object

yet another cacher

clear()

close()

keys()

load(*cachekey*)

rrr(*verbose=True*, *reload_module=True*)
special class reloading function This function is often injected as rrr of classes

save(*cachekey*, *data*)

utool.util_cache.**cached_func**(*fname=None*, *cache_dir='default'*, *appname='utool'*, *key_argx=None*, *key_kwds=None*, *use_cache=None*, *verbose=None*)
Wraps a function with a Cacher object

uses a hash of arguments as input

Parameters

- **fname** (*str*) – file name (defaults to function name)
- **cache_dir** (*unicode*) – (default = u'default')
- **appname** (*unicode*) – (default = u'utool')
- **key_argx** (*None*) – (default = None)
- **key_kwds** (*None*) – (default = None)
- **use_cache** (*bool*) – turns on disk based caching(default = None)

CommandLine: python -m utool.util_cache –exec-cached_func

Example

```
>>> # ENABLE_DOCTEST
>>> import utool as ut
>>> def costly_func(a, b, c='d', *args, **kwargs):
...     return ([a] * b, c, args, kwargs)
>>> ans0 = costly_func(41, 3)
>>> ans1 = costly_func(42, 3)
>>> closure_ = ut.cached_func('costly_func', appname='utool_test',
>>>                           key_argx=[0, 1])
>>> efficient_func = closure_(costly_func)
```

(continues on next page)

(continued from previous page)

```
>>> ans2 = efficient_func(42, 3)
>>> ans3 = efficient_func(42, 3)
>>> ans4 = efficient_func(41, 3)
>>> ans5 = efficient_func(41, 3)
>>> assert ans1 == ans2
>>> assert ans2 == ans3
>>> assert ans5 == ans4
>>> assert ans5 == ans0
>>> assert ans1 != ans0
```

utool.util_cache.**cachestr_repr**(*val*)
Representation of an object as a cache string.

utool.util_cache.**consensed_cfgstr**(*prefix*, *cfgstr*, *max_len=128*, *cfgstr_hashlen=16*)

utool.util_cache.**delete_global_cache**(*appname='default'*)
Reads cache files to a safe place in each operating system

utool.util_cache.**from_json**(*json_str*, *allow_pickle=False*)
Decodes a JSON object specified in the utool convention

Parameters

- **json_str** (*str*) –
- **allow_pickle** (*bool*) – (default = False)

Returns val

Return type object

CommandLine: python -m utool.util_cache from_json –show

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_cache import *  # NOQA
>>> import utool as ut
>>> json_str = 'just a normal string'
>>> json_str = '["just a normal string"]'
>>> allow_pickle = False
>>> val = from_json(json_str, allow_pickle)
>>> result = ('val = %s' % (ut.repr2(val),))
>>> print(result)
```

utool.util_cache.**get_cfgstr_from_args**(*func*, *args*, *kwargs*, *key_argx*, *key_kwds*, *kwdefaults*, *argnames*, *use_hash=None*)

Dev: argx = ['fdsf', '432443432432', 43423432, 'fdsfsd', 3.2, True] memlist = list(map(cachestr_repr, argx))

Ignore: argx = key_argx[0] argval = args[argx] val = argval %timeit repr(argval) %timeit to_json(argval) %timeit utool.hashstr(to_json(argval)) %timeit memoryview(argval)

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_cache import *  # NOQA
>>> import utool as ut
>>> use_hash = None
>>> func = consensed_cfgstr
>>> args = ('a', 'b', 'c', 'd')
>>> kwargs = {}
>>> key_argx = [0, 1, 2]
>>> key_kwds = []
>>> kwdefaults = ut.util_inspect.get_kwdefaults(func)
>>> argnames   = ut.util_inspect.get_argnames(func)
>>> get_cfgstr_from_args(func, args, kwargs, key_argx, key_kwds, kwdefaults,␣
↪argnames)
```

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_cache import *  # NOQA
>>> import utool as ut
>>> self = ut.LazyList
>>> use_hash = None
>>> func = self.append
>>> args = ('a', 'b')
>>> kwargs = {}
>>> key_argx = [1]
>>> key_kwds = []
>>> kwdefaults = ut.util_inspect.get_kwdefaults(func)
>>> argnames   = ut.util_inspect.get_argnames(func)
>>> get_cfgstr_from_args(func, args, kwargs, key_argx, key_kwds, kwdefaults,␣
↪argnames)
```

utool.util_cache.**get_default_appname**()

utool.util_cache.**get_func_result_cachekey**(*func_*, *args_=()*, *kwargs_={}*)
TODO: recursive partial definitions kwargs = {} args = ([],)

utool.util_cache.**get_global_cache_dir**(*appname='default'*, *ensure=False*)
Returns (usually) writable directory for an application cache

utool.util_cache.**get_global_shelf_fpath**(*appname='default'*, *ensure=False*)
Returns the filepath to the global shelf

utool.util_cache.**get_lru_cache**(*max_size=5*)

Parameters **max_size** (*int*) –

References

https://github.com/amitdev/lru-dict

CommandLine: python -m utool.util_cache –test-get_lru_cache

Example

```
>>> # DISABLE_DOCTEST
>>> # UNSTABLE_DOCTEST
>>> from utool.util_cache import *  # NOQA
>>> import utool as ut  # NOQA
>>> max_size = 5
>>> # execute function
>>> cache_obj = get_lru_cache(max_size)
>>> cache_obj[1] = 1
>>> cache_obj[2] = 2
>>> cache_obj[3] = 3
>>> cache_obj[4] = 4
>>> cache_obj[5] = 5
>>> cache_obj[6] = 6
>>> # verify results
>>> result = ut.repr2(dict(cache_obj), nl=False)
>>> print(result)
{2: 2, 3: 3, 4: 4, 5: 5, 6: 6}
```

utool.util_cache.**global_cache_dump**(*appname='default'*)

utool.util_cache.**global_cache_read**(*key*, *appname='default'*, ***kwargs*)

utool.util_cache.**global_cache_write**(*key*, *val*, *appname='default'*)
Writes cache files to a safe place in each operating system

utool.util_cache.**load_cache**(*dpath*, *fname*, *cfgstr*, *ext='.cPkl'*, *verbose=None*, *enabled=True*)
Loads data using util_io, but smartly constructs a filename

utool.util_cache.**make_utool_json_encoder**(*allow_pickle=False*)

References

http://stackoverflow.com/questions/8230315/python-sets-are http://stackoverflow.com/questions/11561932/why-does-json https://github.com/jsonpickle/jsonpickle http://stackoverflow.com/questions/24369666/typeerror-b1 http://stackoverflow.com/questions/30469575/how-to-pickle

utool.util_cache.**save_cache**(*dpath*, *fname*, *cfgstr*, *data*, *ext='.cPkl'*, *verbose=None*)
Saves data using util_io, but smartly constructs a filename

utool.util_cache.**shelf_open**(*fpath*)
allows for shelf to be used in with statements

References

http://stackoverflow.com/questions/7489732/easiest-way-to-add-a-function-to-existing-class

CommandLine: python -m utool.util_cache –test-shelf_open

Example

```
>>> # DISABLE_DOCTEST
>>> # UNSTABLE_DOCTEST
>>> import utool as ut
>>> fpath = ut.unixjoin(ut.ensure_app_resource_dir('utool'), 'testshelf.shelf')
```

(continues on next page)

(continued from previous page)

```
>>> with ut.shelf_open(fpath) as dict_:
...     print(ut.repr4(dict_))
```

utool.util_cache.**text_dict_read**(*fpath*)

utool.util_cache.**text_dict_write**(*fpath*, *dict_*)
Very naive, but readable way of storing a dictionary on disk FIXME: This broke on RoseMary's big dataset. Not sure why. It gave bad syntax. And the SyntaxError did not seem to be excepted.

utool.util_cache.**time_different_diskstores**()
%timeit shelf_write_test() # 15.1 ms per loop %timeit cPickle_write_test() # 1.26 ms per loop

%timeit shelf_read_test() # 8.77 ms per loop %timeit cPickle_read_test() # 2.4 ms per loop %timeit cPickle_read_test2() # 2.35 ms per loop

%timeit json_read_test() %timeit json_write_test()

utool.util_cache.**to_json**(*val*, *allow_pickle=False*, *pretty=False*)
Converts a python object to a JSON string using the utool convention

Parameters **val** (*object*) –

Returns json_str

Return type str

References

http://stackoverflow.com/questions/11561932/why-does-json-dumpslistnp

CommandLine: python -m utool.util_cache –test-to_json python3 -m utool.util_cache –test-to_json

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_cache import *  # NOQA
>>> import utool as ut
>>> import numpy as np
>>> import uuid
>>> val = [
>>>     '{"foo": "not a dict"}',
>>>     1.3,
>>>     [1],
>>>     # {1: 1, 2: 2, 3: 3}, cant use integer keys
>>>     {1, 2, 3},
>>>     slice(1, None, 1),
>>>     b'an ascii string',
>>>     np.array([1, 2, 3]),
>>>     ut.get_zero_uuid(),
>>>     ut.LazyDict(x='fo'),
>>>     ut.LazyDict,
>>>     {'x': {'a', 'b', 'cde'}, 'y': [1]}
>>> ]
>>> #val = ut.LazyDict(x='fo')
>>> allow_pickle = True
>>> if not allow_pickle:
>>>     val = val[:-2]
```

(continues on next page)

(continued from previous page)

```
>>> json_str = ut.to_json(val, allow_pickle=allow_pickle)
>>> result = ut.repr3(json_str)
>>> reload_val = ut.from_json(json_str, allow_pickle=allow_pickle)
>>> # Make sure pickle doesnt happen by default
>>> try:
>>>     json_str = ut.to_json(val)
>>>     assert False or not allow_pickle, 'expected a type error'
>>> except TypeError:
>>>     print('Correctly got type error')
>>> try:
>>>     json_str = ut.from_json(val)
>>>     assert False, 'expected a type error'
>>> except TypeError:
>>>     print('Correctly got type error')
>>> print(result)
>>> print('original = ' + ut.repr3(val, nl=1))
>>> print('reconstructed = ' + ut.repr3(reload_val, nl=1))
>>> assert reload_val[6] == val[6].tolist()
>>> assert reload_val[6] is not val[6]
```

Example

```
>>> # test 3.7 safe uuid
>>> import uuid
>>> import utool as ut
>>> ut.to_json([uuid.uuid4()])
```

utool.util_cache.**tryload_cache**(*dpath*, *fname*, *cfgstr*, *verbose=None*)
returns None if cache cannot be loaded

utool.util_cache.**tryload_cache_list**(*dpath*, *fname*, *cfgstr_list*, *verbose=False*)
loads a list of similar cached datas. Returns flags that needs to be computed

utool.util_cache.**tryload_cache_list_with_compute**(*use_cache*, *dpath*, *fname*, *cfgstr_list*, *compute_fn*, **args*)
tries to load data, but computes it if it can't give a compute function

utool.util_cache.**view_global_cache_dir**(*appname='default'*)

1.15 utool.util_class module

In this module:

- a metaclass allowing for reloading of single class instances
- functions to autoinject methods into a class upon instance creation.
- A wrapper class allowing an object's properties to be used as kwargs
- a metaclass to forward properties to another class

ReloadingMetaclass KwargsWrapper

class utool.util_class.**HashComparable**
Bases: object

class utool.util_class.**HashComparable2**
Bases: object

class utool.util_class.**HashComparableMetaclass**
Bases: type

Defines extra methods for Configs

FIXME: this breaks in python3 because anything that overwrites hash overwrites inherited __eq__

https://docs.python.org/3.6/reference/datamodel.html#object.__hash__

class utool.util_class.**KwargsWrapper**(*obj*)
Bases: collections.abc.Mapping

Allows an arbitrary object attributes to be passed as a **kwargs argument.

class utool.util_class.**ReloadingMetaclass**(*name*, *bases*, *dct*)
Bases: type

Classes with this metaclass will be able to reload themselves on a per-instance basis using the rrr function.

If the functions _on_reload and _initialize_self exist they will be called after and before reload respectively. Any inject_instance functions should be handled there.

SeeAlso: test_reloading_metaclass - shows a working example of this doctest

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_class import *  # NOQA
>>> import utool as ut
>>> @six.add_metaclass(ut.ReloadingMetaclass)
>>> class Foo(object):
...     def __init__(self):
...         pass
>>> # You can edit foo on disk and call rrr in ipython
>>> # if you add a new function to it
>>> foo = Foo()
>>> # This will not work as a doctests because
>>> # Foo's parent module will be __main__ but
>>> # there will be no easy way to write to it.
>>> # This does work when you run from ipython
>>> @six.add_metaclass(ut.ReloadingMetaclass)
>>> class Foo(object):
...     def __init__(self):
...         pass
...     def bar(self):
...         return 'spam'
>>> foo.rrr()
>>> result = foo.bar()
>>> print(result)
spam
```

utool.util_class.**autogen_explicit_injectable_metaclass**(*classname*, *regen_command=None*, *conditional_imports=None*)

Parameters **classname** –

Returns

Return type

?

CommandLine: python -m utool.util_class –exec-autogen_explicit_injectable_metaclass

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_class import *  # NOQA
>>> from utool.util_class import  __CLASSTYPE_ATTRIBUTES__  # NOQA
>>> import wbia
>>> import wbia.control.IBEISControl
>>> classname = wbia.control.controller_inject.CONTROLLER_CLASSNAME
>>> result = autogen_explicit_injectable_metaclass(classname)
>>> print(result)
```

utool.util_class.**autogen_import_list**(*classname*, *conditional_imports=None*)

utool.util_class.**compare_instance**(*op*, *self*, *other*)

utool.util_class.**decorate_class_method**(*func*, *classkey=None*, *skipmain=False*)
Will inject all decorated function as methods of classkey

classkey is some identifying string, tuple, or object

func can also be a tuple

utool.util_class.**decorate_postinject**(*func*, *classkey=None*, *skipmain=False*)
Will perform func with argument self after inject_instance is called on classkey

classkey is some identifying string, tuple, or object

utool.util_class.**get_classname**(*class_*, *local=False*)

Parameters

- **class** (*type*) –
- **local** (*bool*) – (default = False)

Returns classname

Return type str

CommandLine: python -m utool.util_class –exec-get_classname –show

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_class import *  # NOQA
>>> import utool as ut
>>> class_ = ReloadingMetaclass
>>> local = False
>>> assert get_classname(class_, local) == 'utool.util_class.ReloadingMetaclass'
>>> assert get_classname(class_, local=True) == 'ReloadingMetaclass'
```

utool.util_class.**get_comparison_methods**()
makes methods for >, <, =, etc...

utool.util_class.**get_comparison_operators**()

utool.util_class.**get_injected_modules**(*classname*)

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_class import __CLASSNAME_CLASSKEY_REGISTER__  # NOQA
```

utool.util_class.**get_method_func**(*method*)

utool.util_class.**inject_all_external_modules**(*self*, *classname=None*, *allow_override='override+warn'*, *strict=True*)

dynamically injects registered module methods into a class instance

FIXME: naming convention and use this in all places where this clas is used

utool.util_class.**inject_func_as_method**(*self*, *func*, *method_name=None*, *class_=None*, *allow_override=False*, *allow_main=False*, *verbose=True*, *override=None*, *force=False*)

Injects a function into an object as a method

Wraps func as a bound method of self. Then injects func into self It is preferable to use make_class_method_decorator and inject_instance

Parameters

- **self** (*object*) – class instance
- **func** – some function whos first arugment is a class instance
- **method_name** (*str*) – default=func.__name__, if specified renames the method
- **class** (*type*) – if func is an unbound method of this class

References

http://stackoverflow.com/questions/1015307/python-bind-an-unbound-method

utool.util_class.**inject_func_as_property**(*self*, *func*, *method_name=None*, *class_=None*)

Warning: properties are more safely injected using metaclasses

References

http://stackoverflow.com/questions/13850114/dynamically-adding-methods-with-or-without-metaclass-in-python

utool.util_class.**inject_func_as_unbound_method**(*class_*, *func*, *method_name=None*)

This is actually quite simple

utool.util_class.**inject_instance**(*self*, *classkey=None*, *allow_override=False*, *verbose=False*, *strict=True*)

Injects an instance (self) of type (classkey) with all functions registered to (classkey)

call this in the __init__ class function

Parameters

- **self** – the class instance
- **classkey** – key for a class, preferably the class type itself, but it doesnt have to be

SeeAlso: make_class_method_decorator

Example

```
>>> # DISABLE_DOCTEST
>>> # DOCTEST_DISABLE
>>> utool.make_class_method_decorator(InvertedIndex)(smk_debug.invindex_dbgstr)
>>> utool.inject_instance(invindex)
```

utool.util_class.**makeForwardingMetaclass**(*forwarding_dest_getter*, *whitelist*, *base_class=<class 'object'>*)

makes a metaclass that overrides __getattr__ and __setattr__ to forward some specific attribute references to a specified instance variable

utool.util_class.**make_class_method_decorator**(*classkey*, *modname=None*)

register a class to be injectable classkey is a key that identifies the injected class REMEMBER to call inject_instance in __init__

Parameters

- **classkey** – the class to be injected into
- **modname** – the global __name__ of the module youa re injecting from

Returns decorator for injectable methods

Return type closure_decorate_class_method (func)

Example

```
>>> # ENABLE_DOCTEST
>>> import utool as ut
>>> class CheeseShop(object):
...     def __init__(self):
...         import utool as ut
...         ut.inject_all_external_modules(self)
>>> cheeseshop_method = ut.make_class_method_decorator(CheeseShop)
>>> shop1 = CheeseShop()
>>> assert not hasattr(shop1, 'has_cheese'), 'have not injected yet'
>>> @cheeseshop_method
>>> def has_cheese(self):
>>>     return False
>>> shop2 = CheeseShop()
>>> assert shop2.has_cheese() is False, 'external method not injected'
>>> print('Cheese shop does not have cheese. All is well.')
```

utool.util_class.**make_class_postinject_decorator**(*classkey*, *modname=None*)

Parameters

- **classkey** – the class to be injected into
- **modname** – the global __name__ of the module youa re injecting from

Returns decorator for injectable methods

Return type closure_decorate_postinject (func)

SeeAlso: make_class_method_decorator

utool.util_class.**postinject_instance**(*self*, *classkey*, *verbose=False*)

utool.util_class.**reload_class**(*self*, *verbose=True*, *reload_module=True*)
special class reloading function This function is often injected as rrr of classes

utool.util_class.**reload_class_methods**(*self*, *class_*, *verbose=True*)
rebinds all class methods

Parameters

- **self** (*object*) – class instance to reload
- **class** (*type*) – type to reload as

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_class import *  # NOQA
>>> self = '?'
>>> class_ = '?'
>>> result = reload_class_methods(self, class_)
>>> print(result)
```

utool.util_class.**reload_injected_modules**(*classname*)

utool.util_class.**reloadable_class**(*cls*)
convinience decorator instead of @six.add_metaclass(ReloadingMetaclass)

utool.util_class.**reloading_meta_metaclass_factory**(*BASE_TYPE=<class 'type'>*)
hack for pyqt

utool.util_class.**remove_private_obfuscation**(*self*)
removes the python obfuscation of class privates so they can be executed as they appear in class source. Useful when playing with IPython.

utool.util_class.**test_reloading_metaclass**()

CommandLine: python -m utool.util_class –test-test_reloading_metaclass

References

http://stackoverflow.com/questions/8122734/pythons-imp-reload-function-is-not-working

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_class import *  # NOQA
>>> result = test_reloading_metaclass()
>>> print(result)
```

1.16 utool.util_config module

util_config

TODO: FINISH ME AND ALLOW FOR CUSTOM SETTINGS BASED OFF OF A USER PROFILE

utool.util_config.**get_default_global_config**()

utool.util_config.**get_default_repo_config**()
import utool

utool.util_config.**read_repo_config**()

utool.util_config.**write_default_repo_config**()

1.17 utool.util_const module

1.18 utool.util_cplat module

cross platform utilities

utool.util_cplat.**assert_installed_debian**(*pkgname*)

utool.util_cplat.**change_term_title**(*title*)
only works on unix systems only tested on Ubuntu GNOME changes text on terminal title for identifying debugging tasks.

The title will remain until python exists

Parameters **title** (*str*) –

References

http://stackoverflow.com/questions/5343265/setting-title-for-tabs-in-terminator-console-application-in-ubuntu/8850484#8850484

CommandLine: python -m utool change_term_title echo -en "033]0;newtitlea"

printf "e]2;newtitlea";

echo -en "033]0;DocTest /home/joncrall/code/ibeis/wbia.algo.graph.core.py –test-AnnotInference._make_state_deltaa"

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_cplat import *  # NOQA
>>> title = 'change title test'
>>> result = change_term_title(title)
>>> print(result)
```

utool.util_cplat.**check_installed_debian**(*pkgname*)

References

http://www.cyberciti.biz/faq/find-out-if-package-is-installed-in-linux/

utool.util_cplat.**chmod**(*fpath*, *option*)

utool.util_cplat.**chmod_add_executable**(*fpath*, *group=True*, *user=True*)

References

http://stackoverflow.com/questions/15607903/python-module-os-chmodfile-664-does-not-change-the-permission-to-rw-rw-r-bu
http://www.tutorialspoint.com/python/os_chmod.htm https://en.wikipedia.org/wiki/Chmod

utool.util_cplat.**cmd**(**args*, ***kwargs*)
A really roundabout way to issue a system call

FIXME: This function needs some work # It should work without a hitch on windows or unix. # It should be able to spit out stdout in realtime. # Should be able to configure detatchment, shell, and sudo.

FIXME: on a mac ut.cmd('/Users/joncrall/Library/Application Support/ibeis/tomcat/bin/shutdown.sh') will fail due to spaces

Kwargs: quiet (bool) : silence (bool) : verbose (bool) : detatch (bool) : shell (bool) : sudo (bool) : pad_stdout (bool) : dryrun (bool) :

Returns (None, None, None)

Return type tuple

CommandLine: python -m utool.util_cplat –test-cmd python -m utool.util_cplat –test-cmd:0 python -m utool.util_cplat –test-cmd:1 python -m utool.util_cplat –test-cmd:2 python -m utool.util_cplat –test-cmd:1 –test-sudo python -m utool.util_cplat –test-cmd:2 –test-sudo

Example

```
>>> # ENABLE_DOCTEST
>>> import utool as ut
>>> (out, err, ret) = ut.cmd('echo', 'hello world')
>>> result = ut.repr4(list(zip(('out', 'err', 'ret'), (out, err, ret))),␣
↪nobraces=True)
>>> print(result)
('out', 'hello world\n'),
('err', None),
('ret', 0),
```

Example

```
>>> # ENABLE_DOCTEST
>>> import utool as ut
>>> target = ut.codeblock(
...       r'''
          ('out', 'hello world\n'),
          ('err', None),
          ('ret', 0),
          ''')
```

(continues on next page)

(continued from previous page)

```
>>> varydict = {
...     'shell': [True, False],
...     'detatch': [False],
...     'sudo': [True, False] if ut.get_argflag('--test-sudo') else [False],
...     'args': ['echo hello world', ('echo', 'hello world')],
... }
>>> for count, kw in enumerate(ut.all_dict_combinations(varydict), start=1):
>>>     print('+ --- TEST CMD %d ---' % (count,))
>>>     print('testing cmd with params ' + ut.repr4(kw))
>>>     args = kw.pop('args')
>>>     restup = ut.cmd(args, pad_stdout=False, **kw)
>>>     tupfields = ('out', 'err', 'ret')
>>>     output = ut.repr4(list(zip(tupfields, restup)), nobraces=True)
>>>     ut.assert_eq(output, target)
>>>     print('L ___ TEST CMD %d ___\n' % (count,))
```

Example

```
>>> # DISABLE_DOCTEST
>>> # UNSTABLE_DOCTEST
>>> # ping is not as universal of a command as I thought
>>> from utool.util_cplat import *  # NOQA
>>> import utool as ut
>>> varydict = {
...     'shell': [True, False],
...     'detatch': [True],
...     'args': ['ping localhost', ('ping', 'localhost')],
... }
>>> proc_list = []
>>> for count, kw in enumerate(ut.all_dict_combinations(varydict), start=1):
>>>     print('+ --- TEST CMD %d ---' % (count,))
>>>     print('testing cmd with params ' + ut.repr4(kw))
>>>     args = kw.pop('args')
>>>     restup = ut.cmd(args, pad_stdout=False, **kw)
>>>     out, err, proc = restup
>>>     proc_list.append(proc)
>>>     print(proc)
>>>     print(proc)
>>>     print(proc.poll())
>>>     print('L ___ TEST CMD %d ___\n' % (count,))
```

utool.util_cplat.**cmd2**(*command*, *shell=False*, *detatch=False*, *verbose=False*, *verbout=None*)

Trying to clean up cmd

Parameters

- **command** (*str*) – string command
- **shell** (*bool*) – if True, process is run in shell
- **detatch** (*bool*) – if True, process is run in background
- **verbose** (*int*) – verbosity mode
- **verbout** (*bool*) – if True, *command* writes to stdout in realtime. defaults to True iff verbose > 0

Returns info - information about command status

Return type dict

utool.util_cplat.**editfile**(*fpath*)
Runs gvim. Can also accept a module / class / function

utool.util_cplat.**ensure_app_cache_dir**(*appname*, **args*)

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_cplat import *  # NOQA
>>> import utool as ut
>>> dpath = ut.ensure_app_cache_dir('utool')
>>> assert exists(dpath)
```

utool.util_cplat.**ensure_app_resource_dir**(**args*, ***kwargs*)

utool.util_cplat.**get_app_cache_dir**(*appname*, **args*)
Returns a writable directory for an application. This should be used for temporary deletable data.

Parameters

- **appname** (*str*) – the name of the application
- ***args** – any other subdirectories may be specified

Returns dpath: writable cache directory

Return type str

utool.util_cplat.**get_computer_name**()
Returns machine name

utool.util_cplat.**get_dir_diskspaces**(*dir_*)

utool.util_cplat.**get_disk_space**(*start_path='.'*)

References

http://stackoverflow.com/questions/1392413/calculating-a-directory-size-using-python

utool.util_cplat.**get_dynamic_lib_globstrs**()

utool.util_cplat.**get_dynlib_dependencies**(*lib_path*)
Executes tools for inspecting dynamic library dependencies depending on the current platform.

utool.util_cplat.**get_dynlib_exports**(*lib_path*)
Executes tools for inspecting dynamic library dependencies depending on the current platform. Returns the names of callable functions.

Parameters **lib_path** (*str*) –

Returns depend_out

Return type str

CommandLine: python -m utool.util_cplat –test-get_dynlib_exports

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_cplat import *  # NOQA
>>> lib_path = '/home/joncrall/venv/local/lib/python2.7/site-packages/pyflann/lib/
↪libflann.so'
>>> depend_out = get_dynlib_exports(lib_path)
>>> result = ('depend_out = %s' % (str(depend_out),))
>>> print(result)
```

utool.util_cplat.**get_file_info**(*fpath*, *with_fpath=False*, *nice=True*)

utool.util_cplat.**get_file_nBytes**(*fpath*)

utool.util_cplat.**get_file_nBytes_str**(*fpath*)

utool.util_cplat.**get_flops**()
DOESNT WORK

utool.util_cplat.**get_free_diskbytes**(*dir_*)

Parameters **dir** (*str*) –

Returns bytes_ folder/drive free space (in bytes)

Return type int

References

http://stackoverflow.com/questions/51658/cross-platform-space-remaining-on-volume-using-python
http://linux.die.net/man/2/statvfs

CommandLine: python -m utool.util_cplat –exec-get_free_diskbytes python -m utool.util_cplat –exec-get_free_diskbytes –dir /media/raid python -m utool.util_cplat –exec-get_free_diskbytes –dir E:

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_cplat import *  # NOQA
>>> import utool as ut
>>> dir_ = ut.get_argval('--dir', type_=str, default=ut.truepath('~'))
>>> bytes_ = get_free_diskbytes(dir_)
>>> result = ('bytes_ = %s' % (str(bytes_),))
>>> print(result)
>>> print('Unused space in %r = %r' % (dir_, ut.byte_str2(bytes_)))
>>> print('Total space in %r = %r' % (dir_, ut.byte_str2(get_total_diskbytes(dir_
↪))))
```

utool.util_cplat.**get_install_dirs**()

utool.util_cplat.**get_lib_ext**()

utool.util_cplat.**get_path_dirs**()
returns a list of directories in the PATH system variable

Returns pathdirs

Return type list

CommandLine: python -m utool.util_cplat –exec-get_path_dirs

Example

```
>>> # DISABLE_DOCTEST
>>> # SCRIPT
>>> from utool.util_cplat import *  # NOQA
>>> import utool as ut
>>> pathdirs = get_path_dirs()
>>> result = ('pathdirs = %s' % (ut.repr4(pathdirs),))
>>> print(result)
```

utool.util_cplat.**get_plat_specifier**()
Standard platform specifier used by distutils

utool.util_cplat.**get_pylib_ext**()

utool.util_cplat.**get_python_dynlib**()
Ge Python's dynamic library

python -c "import utool; print(utool.get_python_dynlib())"

get_python_dynlib

Returns dynlib

Return type

?

Example

```
>>> # DISABLE_DOCTEST
>>> # DOCTEST_DISABLE
>>> from utool.util_cplat import *  # NOQA
>>> dynlib = get_python_dynlib()
>>> print(dynlib)
/usr/lib/x86_64-linux-gnu/libpython2.7.so
```

utool.util_cplat.**get_system_python_library**()
FIXME; hacky way of finding python library. Not cross platform yet.

utool.util_cplat.**get_total_diskbytes**(*dir_*)

utool.util_cplat.**get_user_name**()
Returns user homefolder name

utool.util_cplat.**geteditor**()

utool.util_cplat.**getroot**()

utool.util_cplat.**in_pyinstaller_package**()

References

http://stackoverflow.com/questions/22472124/what-is-sys-meipass-in-python http://stackoverflow.com/questions/7674790/bundling-data-files-with-pyinstaller-onefile

utool.util_cplat.**ipython_paste**(**args*, ***kwargs*)
pastes for me FIXME: make something like this work on unix and windows

utool.util_cplat.**is64bit_python**()

Returns True if running 64 bit python and False if running on 32 bit

utool.util_cplat.**is_file_executable**(*fpath*)

utool.util_cplat.**is_file_writable**(*fpath*)

utool.util_cplat.**ls_libs**(*dpath*)

utool.util_cplat.**pip_install**(*package*)

References

http://stackoverflow.com/questions/15974100/ipython-install-new-modules

utool.util_cplat.**platform_cache_dir**()
Returns a directory which should be writable for any application This should be used for temporary deletable data.

utool.util_cplat.**print_dir_diskspace**(*dir_*)

utool.util_cplat.**print_path**(*sort=True*)

utool.util_cplat.**print_system_users**()
prints users on the system

On unix looks for /bin/bash users in /etc/passwd

CommandLine: python -m utool.util_cplat –test-print_system_users

Example

```
>>> # DISABLE_DOCTEST
>>> # SCRIPT
>>> from utool.util_cplat import *  # NOQA
>>> result = print_system_users()
>>> print(result)
```

utool.util_cplat.**python_executable**(*check=True*, *short=False*)

Parameters **short** (*bool*) – (default = False)

Returns

Return type str

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_cplat import *  # NOQA
>>> short = False
>>> result = python_executable(short)
>>> print(result)
```

utool.util_cplat.**quote_single_command**(*cmdstr*)

utool.util_cplat.**run_realtime_process**(*exe*, *shell=False*)

utool.util_cplat.**search_env_paths**(*fname*, *key_list=None*, *verbose=None*)
Searches your PATH to see if fname exists

Parameters **fname** (*str*) – file name to search for (can be glob pattern)

CommandLine: python -m utool search_env_paths –fname msvcr*.dll python -m utool search_env_paths –fname '*flann*'

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_cplat import *  # NOQA
>>> import utool as ut
>>> fname = 'opencv2/highgui/libopencv_highgui.so'
>>> fname = ut.get_argval('--fname', default='*')
>>> print('fname = %r' % (fname,))
>>> key_list = None # ['PATH']
>>> found = search_env_paths(fname, key_list)
>>> print(ut.repr4(found, nl=True, strvals=True))
```

Ignore: OpenCV_DIR:PATH={share_opencv} OpenCV_CONFIG_PATH:FILEPATH={share_opencv}

utool.util_cplat.**send_keyboard_input**(*text=None*, *key_list=None*)

Parameters

- **text** (*None*) –
- **key_list** (*list*) –

References

http://stackoverflow.com/questions/14788036/python-win32api-sendmesage http://www.pinvoke.net/default.aspx/user32.sendinput

CommandLine: python -m utool.util_cplat –test-send_keyboard_input

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_cplat import *  # NOQA
>>> text = '%paste'
>>> result = send_keyboard_input('%paste')
>>> print(result)
```

utool.util_cplat.**set_process_title**(*title*)

utool.util_cplat.**shell**(**args*, ***kwargs*)
Dangerous. Take out of production code

utool.util_cplat.**spawn_delayed_ipython_paste**()

utool.util_cplat.**startfile**(*fpath*, *detatch=True*, *quote=False*, *verbose=False*, *quiet=True*)
Uses default program defined by the system to open a file.

References

http://stackoverflow.com/questions/2692873/quote-posix-shell-special-characters-in-python-output

utool.util_cplat.**unload_module**(*modname*)
WARNING POTENTIALLY DANGEROUS AND MAY NOT WORK

References

http://stackoverflow.com/questions/437589/how-do-i-unload-reload-a-python-module

CommandLine: python -m utool.util_cplat –test-unload_module

Example

```
>>> # DISABLE_DOCTEST
>>> import sys, gc  # NOQA
>>> import pyhesaff
>>> import utool as ut
>>> modname = 'pyhesaff'
>>> print('%s refcount=%r' % (modname, sys.getrefcount(pyhesaff),))
>>> #referrer_list = gc.get_referrers(sys.modules[modname])
>>> #print('referrer_list = %s' % (ut.repr4(referrer_list),))
>>> ut.unload_module(modname)
>>> assert pyhesaff is None
```

utool.util_cplat.**vd**(*dname=None*, *fname=None*, *verbose=True*)
View a directory in the operating system file browser. Currently supports windows explorer, mac open, and linux nautlius.

Parameters

- **dname** (*str*) – directory name
- **fname** (*str*) – a filename to select in the directory (nautlius only)
- **verbose** (*bool*) –

CommandLine: python -m utool.util_cplat –test-view_directory

Example

```
>>> # DISABLE_DOCTEST
>>> # DOCTEST_DISABLE
>>> from utool.util_cplat import *  # NOQA
>>> import utool as ut
>>> dname = ut.truepath('~')
>>> verbose = True
>>> view_directory(dname, verbose)
```

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_cplat import *  # NOQA
>>> import utool as ut
>>> base = ut.ensure_app_cache_dir('utool', 'test_vd')
>>> dirs = [
>>>     '',
>>>     'dir1',
>>>     'has space',
>>>     'space at end ',
>>>     ' space at start ',
>>>     '"quotes and spaces"',
>>>     "'single quotes and spaces'",
>>>     'Frogram Piles (y2K)',
>>> ]
>>> dirs_ = [ut.ensuredir(join(base, d)) for d in dirs]
>>> for dname in dirs_:
>>>     ut.view_directory(dname, verbose=False)
>>> fpath = join(base, 'afile.txt')
>>> ut.touch(fpath)
>>> ut.view_directory(base, fpath, verbose=False)
```

utool.util_cplat.**view_directory**(*dname=None*, *fname=None*, *verbose=True*)

View a directory in the operating system file browser. Currently supports windows explorer, mac open, and linux nautlius.

Parameters

- **dname** (*str*) – directory name
- **fname** (*str*) – a filename to select in the directory (nautlius only)
- **verbose** (*bool*) –

CommandLine: python -m utool.util_cplat –test-view_directory

Example

```
>>> # DISABLE_DOCTEST
>>> # DOCTEST_DISABLE
>>> from utool.util_cplat import *  # NOQA
>>> import utool as ut
>>> dname = ut.truepath('~')
>>> verbose = True
>>> view_directory(dname, verbose)
```

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_cplat import *  # NOQA
>>> import utool as ut
>>> base = ut.ensure_app_cache_dir('utool', 'test_vd')
>>> dirs = [
>>>     '',
>>>     'dir1',
>>>     'has space',
```

(continues on next page)

(continued from previous page)

```
>>>     'space at end ',
>>>     ' space at start ',
>>>     '"quotes and spaces"',
>>>     "'single quotes and spaces'",
>>>     'Frogram Piles (y2K)',
>>> ]
>>> dirs_ = [ut.ensuredir(join(base, d)) for d in dirs]
>>> for dname in dirs_:
>>>     ut.view_directory(dname, verbose=False)
>>> fpath = join(base, 'afile.txt')
>>> ut.touch(fpath)
>>> ut.view_directory(base, fpath, verbose=False)
```

utool.util_cplat.**view_file_in_directory**(*fpaths*)

1.19 utool.util_csv module

class utool.util_csv.**CSV**(*row_data*, *row_headers=None*, *col_headers=None*)

Bases: *utool.util_dev.NiceRepr*

compress_cols(*flags*)

compress_rows(*flags*, *with_header=True*, *inplace=True*)

classmethod from_fpath(*fpath*, ***kwargs*)

fuzzy_filter_columns(*fuzzy_headers*)

fuzzy_find_colx(*pat*)

fuzzy_find_colxs(*pat*)

fuzzy_reorder_columns(*fuzzy_headers*, *inplace=True*)

nice_table()

nice_table2(***kwargs*)

permute_columns(*new_order*, *inplace=True*)

raw_table()

shape

tabulate()

take_column(*colx*, *with_header=True*)

take_fuzzy_column(*pat*)

transpose()

utool.util_csv.**make_csv_table**(*column_list=[]*, *column_lbls=None*, *header=''*, *column_type=None*, *row_lbls=None*, *transpose=False*, *precision=2*, *use_lbl_width=True*, *comma_repl='<com>'*, *raw=False*, *new=False*, *standardize=False*)

Creates a csv table with aligned columns

make_csv_table

Parameters

- **column_list** (*list*) –
- **column_lbls** (*None*) –
- **header** (*str*) –
- **column_type** (*None*) –
- **row_lbls** (*None*) –
- **transpose** (*bool*) –

Returns csv_text

Return type str

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_csv import *  # NOQA
>>> column_list = [[1, 2, 3], ['A', 'B', 'C']]
>>> column_lbls = ['num', 'alpha']
>>> header = '# Test CSV'
>>> column_type = (int, str)
>>> row_lbls = None
>>> transpose = False
>>> csv_text = make_csv_table(column_list, column_lbls, header, column_type, row_
↪lbls, transpose)
>>> result = csv_text
>>> print(result)
# Test CSV
# num_rows=3
#   num,  alpha
      1,      A
      2,      B
      3,      C
```

utool.util_csv.**make_standard_csv**(*column_list*, *column_lbls=None*)

utool.util_csv.**numpy_to_csv**(*arr*, *col_lbls=None*, *header=''*, *col_type=None*)

utool.util_csv.**read_csv**(*fpath*, *binary=True*)
reads csv in unicode

1.20 utool.util_dbg module

TODO: rectify name difference between parent and caller

class utool.util_dbg.**EmbedOnException**
Bases: object

Context manager which embeds in ipython if an exception is thrown

utool.util_dbg.**breakpoint**(**tags*)

utool.util_dbg.**debug_exception**(*func*)

utool.util_dbg.**debug_hstack**(*stacktup*)

utool.util_dbg.**debug_list**(*list_*)

utool.util_dbg.**debug_npstack**(*stacktup*)

utool.util_dbg.**debug_vstack**(*stacktup*)

utool.util_dbg.**embed**(*parent_locals=None*, *parent_globals=None*, *exec_lines=None*, *remove_pyqt_hook=True*, *N=0*)
Starts interactive session. Similar to keyboard command in matlab. Wrapper around IPython.embed

utool.util_dbg.**embed2**(***kwargs*)
Modified from IPython.terminal.embed.embed so I can mess with stack_depth

utool.util_dbg.**execstr_attr_list**(*obj_name*, *attr_list=None*)

utool.util_dbg.**execstr_dict**(*dict_*, *local_name=None*, *exclude_list=None*, *explicit=False*)
returns execable python code that declares variables using keys and values

execstr_dict

Parameters

- **dict** (*dict*) –
- **local_name** (*str*) – optional: local name of dictionary. Specifying this is much safer
- **exclude_list** (*list*) –

Returns

execstr — the executable string that will put keys from dict into local vars

Return type str

CommandLine: python -m utool.util_dbg –test-execstr_dict

Example

```
>>> # DISABLE_DOCTEST
>>> # UNSTABLE_DOCTEST
>>> from utool.util_dbg import *  # NOQA
>>> my_dictionary = {'a': True, 'b': False}
>>> execstr = execstr_dict(my_dictionary)
>>> exec(execstr)
>>> assert 'a' in vars() and 'b' in vars(), 'execstr failed'
>>> assert b is False and a is True, 'execstr failed'
>>> result = execstr
>>> print(result)
a = my_dictionary['a']
b = my_dictionary['b']
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dbg import *  # NOQA
>>> import utool as ut
>>> my_dictionary = {'a': True, 'b': False}
>>> execstr = execstr_dict(my_dictionary)
>>> locals_ = locals()
>>> exec(execstr, locals_)
>>> a, b = ut.dict_take(locals_, ['a', 'b'])
```

(continues on next page)

(continued from previous page)

```
>>> assert 'a' in locals_ and 'b' in locals_, 'execstr failed'
>>> assert b is False and a is True, 'execstr failed'
>>> result = execstr
>>> print(result)
a = my_dictionary['a']
b = my_dictionary['b']
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dbg import *  # NOQA
>>> import utool as ut
>>> my_dictionary = {'a': True, 'b': False}
>>> execstr = execstr_dict(my_dictionary, explicit=True)
>>> result = execstr
>>> print(result)
a = True
b = False
```

utool.util_dbg.**execstr_func**(*func*)

utool.util_dbg.**execstr_parent_locals**()

utool.util_dbg.**explore_module**(*module_*, *seen=None*, *maxdepth=2*, *nonmodules=False*)

utool.util_dbg.**explore_stack**()

utool.util_dbg.**fix_embed_globals**()
HACK adds current locals() to globals(). Can be dangerous.

utool.util_dbg.**fmtlocals**(*string*)
mimics lisp quasi quote functionality

utool.util_dbg.**formatex**(*ex*, *msg='[!?] Caught exception'*, *prefix=None*, *key_list=[]*, *locals_=None*, *iswarning=False*, *tb=False*, *N=0*, *keys=None*, *colored=None*)
Formats an exception with relevant info

Parameters

- **ex** (*Exception*) – exception to print
- **msg** (*unicode*) – a message to display to the user (default = u'[!?] Caught exception')
- **keys** (*None*) – a list of strings denoting variables or expressions of interest (default = [])
- **iswarning** (*bool*) – prints as a warning rather than an error if True (default = False)
- **tb** (*bool*) – if True prints the traceback in the error message (default = False)
- **prefix** (*None*) – (default = None)
- **locals** (*None*) – (default = None)
- **N** (*int*) – (default = 0)
- **colored** (*None*) – (default = None)
- **key_list** (*list*) – DEPRICATED use keys

Returns formated exception

Return type str

CommandLine: python -m utool.util_dbg –exec-formatex

Example

```
>>> # DISABLE_DOCTET
>>> from utool.util_dbg import *  # NOQA
>>> import utool as ut
>>> msg = 'Testing Exception'
>>> prefix = '[test]'
>>> key_list = ['N', 'foo', 'tb']
>>> locals_ = None
>>> iswarning = True
>>> keys = None
>>> colored = None
>>> def failfunc():
>>>     tb = True
>>>     N = 0
>>>     try:
>>>         raise Exception('test exception. This is not an error')
>>>     except Exception as ex:
>>>         result = formatex(ex, msg, prefix, key_list, locals_,
>>>                           iswarning, tb, N, keys, colored)
>>>         return result
>>> result = failfunc().replace('\n\n', '')
>>> print(result)
<!!! WARNING !!!>
Traceback (most recent call last):
  File "<string>", line 15, in failfunc
Exception: test exception. This is not an error[test] Testing Exception
<class 'Exception'>: test exception. This is not an error
[test] N = 0
[test] foo = NameError (this likely due to a misformatted printex and is not␣
↪related to the exception)
[test] tb = True
</!!! WARNING !!!>
```

utool.util_dbg.**get_caller_lineno**(*N=0*, *strict=True*)

utool.util_dbg.**get_caller_modname**(*N=0*, *allowmain=True*)

utool.util_dbg.**get_caller_name**(*N=0*, *allow_genexpr=True*)

get the name of the function that called you

Parameters

- **N** (*int*) – (defaults to 0) number of levels up in the stack
- **allow_genexpr** (*bool*) – (default = True)

Returns a function name

Return type str

CommandLine: python -m utool.util_dbg get_caller_name python -m utool get_caller_name python ~/code/utool/utool/__main__.py get_caller_name python ~/code/utool/utool/__init__.py get_caller_name python ~/code/utool/utool/util_dbg.py get_caller_name

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dbg import *  # NOQA
>>> import utool as ut
>>> N = list(range(0, 13))
>>> allow_genexpr = True
>>> caller_name = get_caller_name(N, allow_genexpr)
>>> print(caller_name)
```

utool.util_dbg.**get_caller_prefix**(*N=0*, *aserror=False*)

utool.util_dbg.**get_current_stack_depth**()

utool.util_dbg.**get_localvar_from_stack**(*varname*)
Finds a local varable somewhere in the stack and returns the value

Parameters **varname** (*str*) – variable name

Returns None if varname is not found else its value

utool.util_dbg.**get_parent_frame**(*N=0*)

utool.util_dbg.**get_reprs**(**args*, ***kwargs*)

utool.util_dbg.**get_stack_frame**(*N=0*, *strict=True*)

Parameters

- **N** (*int*) – N=0 means the frame you called this function in. N=1 is the parent frame.
- **strict** (*bool*) – (default = True)

utool.util_dbg.**get_var_from_stack**(*varname*, *verbose=True*)
Finds a varable (local or global) somewhere in the stack and returns the value

Parameters **varname** (*str*) – variable name

Returns None if varname is not found else its value

utool.util_dbg.**get_varname_from_locals**(*val*, *locals_*, *default='varname-not-found'*, *strict=False*, *cmpfunc_=<built-in function is_>*)
Finds the string name which has where locals_[name] is val

Check the varname is in the parent namespace This will only work with objects not primatives

Parameters

- **()** (*val*) – some value
- **locals** (*dict*) – local dictionary to search
- **default** (*str*) –
- **strict** (*bool*) –

Returns the varname which is Val (if it exists)

Return type str

utool.util_dbg.**get_varname_from_stack**(*var*, *N=0*, ***kwargs*)

utool.util_dbg.**get_varstr**(*val*, *pad_stdout=True*, *locals_=None*)

utool.util_dbg.**get_varval_from_locals**(*key*, *locals_*, *strict=False*)
Returns a variable value from locals. Different from locals()['varname'] because get_varval_from_locals('varname.attribute', locals()) is allowed

utool.util_dbg.**haveIPython**()
Tests if IPython is available

utool.util_dbg.**import_testdata**()

utool.util_dbg.**inIPython**()
Tests if running in IPython.

Reference: https://stackoverflow.com/a/39662359

utool.util_dbg.**in_jupyter_notebook**()
http://stackoverflow.com/questions/15411967/how-can-i-check-if-code-is-executed-in-the-ipython-notebook

utool.util_dbg.**ipython_execstr**()

utool.util_dbg.**is_valid_varname**(*varname*)
Checks syntax and validity of a variable name

utool.util_dbg.**load_testdata**(**args*, ***kwargs*)
tries to load previously cached testdata

utool.util_dbg.**module_functions**(*module*)

utool.util_dbg.**parse_locals_keylist**(*locals_*, *key_list*, *strlist_=None*, *prefix=''*)
For each key in keylist, puts its value in locals into a stringlist

Parameters

- **locals** –
- **key_list** (*list*) –
- **strlist_** (*list*) – (default = None)
- **prefix** (*unicode*) – (default = u'')

Returns list

CommandLine: python -m utool.util_dbg –exec-parse_locals_keylist

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dbg import *  # NOQA
>>> import utool as ut
>>> locals_ = {'foo': [1, 2, 3], 'bar': 'spam', 'eggs': 4, 'num': 5}
>>> key_list = [(len, 'foo'), 'bar.lower.__name__', 'eggs', 'num', 'other']
>>> strlist_ = None
>>> prefix = u''
>>> strlist_ = parse_locals_keylist(locals_, key_list, strlist_, prefix)
>>> result = ('strlist_ = %s' % (ut.repr2(strlist_, nl=True),))
>>> print(result)
strlist_ = [
    ' len(foo) = 3',
    " bar.lower.__name__ = 'lower'",
    ' eggs = 4',
    ' num = 5',
    ' other = NameError (this likely due to a misformatted printex and is not␣
↪related to the exception)',
]
```

utool.util_dbg.**print_frame**(*frame*)

utool.util_dbg.**print_keys**(*key_list*, *locals_=None*)

utool.util_dbg.**print_traceback**(*with_colors=True*)
prints current stack

utool.util_dbg.**printex**(*ex*, *msg='[!?] Caught exception'*, *prefix=None*, *key_list=[]*, *locals_=None*, *iswarning=False*, *tb=False*, *pad_stdout=True*, *N=0*, *use_stdout=False*, *reraise=False*, *msg_=None*, *keys=None*, *colored=None*)
Prints (and/or logs) an exception with relevant info

Parameters

- **ex** (*Exception*) – exception to print
- **msg** (*None*) – a message to display to the user
- **keys** (*None*) – a list of strings denoting variables or expressions of interest
- **iswarning** (*bool*) – prints as a warning rather than an error if True (defaults to False)
- **tb** (*bool*) – if True prints the traceback in the error message
- **pad_stdout** (*bool*) – separate the error message from the rest of stdout with newlines
- **prefix** (*None*) –
- **locals** (*None*) –
- **N** (*int*) –
- **use_stdout** (*bool*) –
- **reraise** (*bool*) –
- **msg** –
- **key_list** (*list*) – DEPRICATED use keys

Returns None

utool.util_dbg.**printvar**(*locals_*, *varname*, *attr='.shape'*, *typepad=0*)

utool.util_dbg.**printvar2**(*varstr*, *attr=''*, *typepad=0*)

utool.util_dbg.**public_attributes**(*input*)

utool.util_dbg.**qflag**(*num=None*, *embed_=True*)

utool.util_dbg.**quasiquote**(*string*)
mimics lisp quasi quote functionality

utool.util_dbg.**quit**(*num=None*, *embed_=False*)

utool.util_dbg.**quitflag**(*num=None*, *embed_=False*, *parent_locals=None*, *parent_globals=None*)

utool.util_dbg.**save_testdata**(**args*, ***kwargs*)
caches testdata

utool.util_dbg.**search_stack_for_localvar**(*varname*)
Finds a local varable somewhere in the stack and returns the value

Parameters **varname** (*str*) – variable name

Returns None if varname is not found else its value

utool.util_dbg.**search_stack_for_var**(*varname*, *verbose=True*)
Finds a varable (local or global) somewhere in the stack and returns the value

Parameters **varname** (*str*) – variable name

Returns None if varname is not found else its value

utool.util_dbg.**super_print**(*val*, *locals_=None*)

1.21 utool.util_decor module

utool.util_decor.**accepts_numpy**(*func*)
Allows the first input to be a numpy array and get result in numpy form

utool.util_decor.**accepts_scalar_input**(*func*)
DEPRICATE in favor of accepts_scalar_input2 only accepts one input as vector

accepts_scalar_input is a decorator which expects to be used on class methods. It lets the user pass either a vector or a scalar to a function, as long as the function treats everything like a vector. Input and output is sanitized to the user expected format on return.

Parameters **func** (*func*) –

Returns wrp_asi

Return type func

CommandLine: python -m utool.util_decor –test-accepts_scalar_input

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_decor import *  # NOQA
>>> @accepts_scalar_input
... def foobar(self, list_):
...     return [x + 1 for x in list_]
>>> self = None  # dummy self because this decorator is for classes
>>> assert 2 == foobar(self, 1)
>>> assert [2, 3] == foobar(self, [1, 2])
```

utool.util_decor.**accepts_scalar_input2**(*argx_list=[0]*, *outer_wrapper=True*)
FIXME: change to better name. Complete implementation.

used in IBEIS setters

accepts_scalar_input2 is a decorator which expects to be used on class methods. It lets the user pass either a vector or a scalar to a function, as long as the function treats everything like a vector. Input and output is sanitized to the user expected format on return.

Parameters **argx_list** (*list*) – indexes of args that could be passed in as scalars to code that operates on lists. Ensures that decorated function gets the argument as an iterable.

utool.util_decor.**accepts_scalar_input_vector_output**(*func*)
DEPRICATE IN FAVOR OF accepts_scalar_input2

accepts_scalar_input_vector_output

utool.util_decor.**apply_docstr**(*docstr_func*)
Changes docstr of one functio to that of another

class utool.util_decor.**classproperty**
Bases: property

Decorates a method turning it into a classattribute

References

https://stackoverflow.com/questions/1697501/python-staticmethod-with-property

utool.util_decor.**debug_function_exceptions**(*func*)

utool.util_decor.**dummy_args_decor**(**args*, ***kwargs*)

utool.util_decor.**getter_1to1**(*func*)
DEPRICATE in favor of accepts_scalar_input2 only accepts one input as vector

accepts_scalar_input is a decorator which expects to be used on class methods. It lets the user pass either a vector or a scalar to a function, as long as the function treats everything like a vector. Input and output is sanitized to the user expected format on return.

Parameters **func** (*func*) –

Returns wrp_asi

Return type func

CommandLine: python -m utool.util_decor –test-accepts_scalar_input

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_decor import *  # NOQA
>>> @accepts_scalar_input
... def foobar(self, list_):
...     return [x + 1 for x in list_]
>>> self = None  # dummy self because this decorator is for classes
>>> assert 2 == foobar(self, 1)
>>> assert [2, 3] == foobar(self, [1, 2])
```

utool.util_decor.**getter_1toM**(*func*)
DEPRICATE IN FAVOR OF accepts_scalar_input2

accepts_scalar_input_vector_output

utool.util_decor.**ignores_exc_tb**(**args*, ***kwargs*)
PYTHON 3 VERSION

ignore_exc_tb decorates a function and remove both itself and the function from any exception traceback that occurs.

This is useful to decorate other trivial decorators which are polluting your stacktrace.

utool.util_decor.**indent_func**(*input_*)
Takes either no arguments or an alias label

utool.util_decor.**interested**(*func*)

utool.util_decor.**lazyfunc**(*func*)
Returns a memcached version of a function

utool.util_decor.**memoize**(*func*)
simple memoization decorator

References

https://wiki.python.org/moin/PythonDecoratorLibrary#Memoize

Parameters **func** (*function*) – live python function

Returns

Return type func

CommandLine: python -m utool.util_decor memoize

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_decor import *  # NOQA
>>> import utool as ut
>>> closure = {'a': 'b', 'c': 'd'}
>>> incr = [0]
>>> def foo(key):
>>>     value = closure[key]
>>>     incr[0] += 1
>>>     return value
>>> foo_memo = memoize(foo)
>>> assert foo('a') == 'b' and foo('c') == 'd'
>>> assert incr[0] == 2
>>> print('Call memoized version')
>>> assert foo_memo('a') == 'b' and foo_memo('c') == 'd'
>>> assert incr[0] == 4
>>> assert foo_memo('a') == 'b' and foo_memo('c') == 'd'
>>> print('Counter should no longer increase')
>>> assert incr[0] == 4
>>> print('Closure changes result without memoization')
>>> closure = {'a': 0, 'c': 1}
>>> assert foo('a') == 0 and foo('c') == 1
>>> assert incr[0] == 6
>>> assert foo_memo('a') == 'b' and foo_memo('c') == 'd'
```

utool.util_decor.**memoize_nonzero**(*func*)
Memoization decorator for functions taking a nonzero number of arguments.

References

http://code.activestate.com/recipes/578231-fastest-memoization-decorator

utool.util_decor.**memoize_single**(*func*)
Memoization decorator for a function taking a single argument

References

http://code.activestate.com/recipes/578231-fastest-memoization-decorator

utool.util_decor.**memoize_zero**(*func*)
Memoization decorator for a function taking no arguments

utool.util_decor.**on_exception_report_input**(*func_=None*, *force=False*, *keys=None*)
If an error is thrown in the scope of this function's stack frame then the decorated function name and the arguments passed to it will be printed to the utool print function.

utool.util_decor.**preserve_sig**(*wrapper*, *orig_func*, *force=False*)
Decorates a wrapper function.

It seems impossible to presever signatures in python 2 without eval (Maybe another option is to write to a temporary module?)

Parameters

- **wrapper** – the function wrapping orig_func to change the signature of
- **orig_func** – the original function to take the signature from

References

http://emptysqua.re/blog/copying-a-python-functions-signature/ https://code.google.com/p/micheles/source/browse/decorator/src/decorator.py

---

Todo: checkout funcsigs https://funcsigs.readthedocs.org/en/latest/

CommandLine: python -m utool.util_decor –test-preserve_sig

Example

```
>>> # ENABLE_DOCTEST
>>> import utool as ut
>>> #ut.rrrr(False)
>>> def myfunction(self, listinput_, arg1, *args, **kwargs):
>>>     " just a test function "
>>>     return [x + 1 for x in listinput_]
>>> #orig_func = ut.take
>>> orig_func = myfunction
>>> wrapper = ut.accepts_scalar_input2([0])(orig_func)
>>> _wrp_preserve1 = ut.preserve_sig(wrapper, orig_func, True)
>>> _wrp_preserve2 = ut.preserve_sig(wrapper, orig_func, False)
>>> print('_wrp_preserve2 = %r' % (_wrp_preserve1,))
>>> print('_wrp_preserve2 = %r' % (_wrp_preserve2,))
>>> #print('source _wrp_preserve1 = %s' % (ut.get_func_sourcecode(_wrp_preserve1),
↪))
>>> #print('source _wrp_preserve2 = %s' % (ut.get_func_sourcecode(_wrp_
↪preserve2)),)
>>> result = str(_wrp_preserve1)
>>> print(result)
```

utool.util_decor.**show_return_value**(*func*)

utool.util_decor.**test_ignore_exec_traceback**()

CommandLine: python -m utool.util_decor –test-test_ignore_exec_traceback

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_decor import *  # NOQA
>>> result = test_ignore_exec_traceback()
>>> print(result)
```

utool.util_decor.**time_func**(*func*)

utool.util_decor.**tracefunc**(*func*)

utool.util_decor.**tracefunc_xml**(*func*)
Causes output of function to be printed in an XML style block

1.22 utool.util_depricated module

utool.util_depricated.**cartesian**(*arrays*, *out=None*)
Generate a cartesian product of input arrays.

Parameters

- **arrays** (*list of array-like*) – 1-D arrays to form the cartesian product of
- **out** (*ndarray*) – Outvar which is modified in place if specified

Returns

cartesian products formed of input arrays 2-D array of shape (M, len(arrays))

Return type out (ndarray)

References

gist.github.com/hernamesbarbara/68d073f551565de02ac5

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_depricated import *  # NOQA
>>> arrays = ([1, 2, 3], [4, 5], [6, 7])
>>> out = cartesian(arrays)
>>> result = repr(out.T)
array([[1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3],
       [4, 4, 5, 5, 4, 4, 5, 5, 4, 4, 5, 5],
       [6, 7, 6, 7, 6, 7, 6, 7, 6, 7, 6, 7]])
```

Timeit:

```
>>> # DISABLE_DOCTEST
>>> # Use itertools product instead
>>> setup = 'import utool as ut\n' + ut.get_doctest_examples(ut.
↪cartesian)[0][0]
>>> statements = [
>>>     'cartesian(arrays)',
>>>     'np.array(list(ut.iprod(*arrays)))',
```

(continues on next page)

(continued from previous page)

```
>>> ]
>>> ut.timeit_compare(statements, setup=setup)
```

utool.util_depricated.**find_std_inliers**(*data*, *m=2*)

1.23 utool.util_dev module

class utool.util_dev.**ClassAttrDictProxy**(*obj*, *keys*, *attrs=None*)

Bases: *utool.util_dict.DictLike*

Allows class attributes to be specified using dictionary syntax

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dev import *  # NOQA
>>> import utool as ut
>>> class Foo(object):
>>>     def __init__(self):
>>>         self.attr1 = 'foo'
>>>         self.attr2 = 'bar'
>>>         self.proxy = ut.ClassAttrDictProxy(self, ['attr1', 'attr2'])
>>> self = Foo()
>>> self.proxy['attr2'] = 'baz'
>>> assert self.attr2 == 'baz', 'should have changed in object'
```

getitem(*key*)

keys()

setitem(*key*, *val*)

class utool.util_dev.**ColumnLists**(*key_to_list={}*, *_meta=None*)

Bases: *utool.util_dev.NiceRepr*

Way to work with column data

Parameters **key_to_list** (*dict*) – (default = {})

CommandLine: python -m utool.util_dev ColumnLists –show

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_dev import *  # NOQA
>>> import utool as ut
>>> key_to_list = {'a': [1, 2, 3], 'b': [4, 5, 6]}
>>> self = ColumnLists(key_to_list)
>>> newself = self.take([1, 2])
>>> print(self)
>>> print(newself)
```

asdataframe()

asdict()

aspandas()

cast_column(*keys*, *func*)
like map column but applies values inplace

chunks(*chunksize*)

compress(*flags*)

copy()

classmethod flatten(*list_*)

get_multis(*key*)

get_singles(*key*)

group(*labels*)
group as list

group_indicies(*labels*)

group_items(*labels*)
group as dict

keys()

loc_by_key(*key*, *vals*)

map_column(*keys*, *func*)

Parameters

- **keys** (*list or str*) – the column name(s) to apply the *func* to
- **func** (*callable*) – applied to each element in the specified columns

merge_rows(*key*, *merge_scalars=True*)
Uses key as a unique index an merges all duplicates rows. Use cast_column to modify types of columns before merging to affect behavior of duplicate rectification.

Parameters

- **key** – row to merge on
- **merge_scalars** – if True, scalar values become lists

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_dev import *  # NOQA
>>> import utool as ut
>>> key_to_list = {
>>>     'uuid': [1, 1, 2, 3, 4, 2, 1],
>>>     'a':    [1, 2, 3, 4, 5, 6, 7],
>>>     'b':    [[1], [2], [3], [4], [5], [6], [7]],
>>>     'c':    [[1], [1], [2], [3], [4], [2], [1]],
>>> }
>>> self = ColumnLists(key_to_list)
>>> key = 'uuid'
>>> newself = self.merge_rows('uuid')
>>> print(newself.to_csv())
```

(continues on next page)

(continued from previous page)

```
#             a,            c,            b,         uuid
              4,          [3],          [4],            3
              5,          [4],          [5],            4
    "[1, 2, 7]",  "[1, 1, 1]",  "[1, 2, 7]",  "[1, 1, 1]"
       "[3, 6]",     "[2, 2]",     "[3, 6]",     "[2, 2]"
```

print(*ignore=None*, *keys=None*)

remove(*idxs*)
Returns a copy with idxs removed

reorder_columns(*keys*)

rrr()

shape

take(*idxs*)
Takes a subset of rows

take_column(*keys*, **extra_keys*)
Takes a subset of columns

to_csv(***kwargs*)

values()

class utool.util_dev.**DictLike_old**
Bases: object

DEPRICATE TODO: use util_dict.DictLike instead

copy()

get(*key*, *default=None*)

items()

iteritems()

itervalues()

keys()

values()

class utool.util_dev.**InteractiveIter**(*iterable=None*, *enabled=True*, *startx=0*, *default_action='next'*, *custom_actions=[]*, *wraparound=False*, *display_item=False*, *verbose=True*)
Bases: object

Choose next value interactively

iterable should be a list, not a generator. sorry

classmethod eventloop(*custom_actions=[]*)
For use outside of iteration wrapping. Makes an interactive event loop custom_actions should be specified in format [dispname, keys, desc, func]

handle_ans(*ans_*)
preforms an actionm based on a user answer

prompt()

wait_for_input()

class utool.util_dev.**InteractivePrompt**(*actions={}*)
Bases: object

handle_ans(*ans_*)
preforms an actionm based on a user answer

loop()

prompt()

register(*tup*)

class utool.util_dev.**MemoryTracker**(*lbl='Memtrack Init'*, *disable=None*, *avail=True*, *used=True*, *total_diff=True*, *abs_mag=True*, *peak=True*)
Bases: object

A class for tracking memory usage. On initialization it logs the current available (free) memory. Calling the report method logs the current available memory as well as memory usage difference w.r.t the last report.

Example

```
>>> # DISABLE_DOCTEST
>>> import utool
>>> import numpy as np
>>> memtrack = utool.MemoryTracker('[ENTRY]')
>>> memtrack.report('[BEFORE_CREATE]')
>>> arr = np.ones(128 * (2 ** 20), dtype=np.uint8)
>>> memtrack.report('[AFTER_CREATE]')
>>> memtrack.track_obj(arr, 'arr')
>>> memtrack.report_objs()
>>> memtrack.report_largest()
>>> del arr
>>> memtrack.report('[DELETE]')
#>>> memtrack.report_largest()
```

collect(**args*, ***kwargs*)

get_available_memory(**args*, ***kwargs*)

get_peak_memory(**args*, ***kwargs*)

get_used_memory(**args*, ***kwargs*)

measure_memory()

record()

report(**args*, ***kwargs*)

report_largest(**args*, ***kwargs*)

report_obj(**args*, ***kwargs*)

report_objs(**args*, ***kwargs*)

report_type(*class_*, *more=False*)

track_obj(**args*, ***kwargs*)

class utool.util_dev.**NamedPartial**(*func*, **args*, ***kwargs*)
Bases: functools.partial, *utool.util_dev.NiceRepr*

class utool.util_dev.**NiceRepr**
Bases: object

base class that defines a nice representation and string func for a class given that the user implements __nice__

Rename to NiceObject?

class utool.util_dev.**PriorityQueue**(*items=None*, *ascending=True*)
Bases: *utool.util_dev.NiceRepr*

abstracted priority queue for our needs

Combines properties of dicts and heaps Uses a heap for fast minimum/maximum value search Uses a dict for fast read only operations

CommandLine: python -m utool.util_dev PriorityQueue

References

http://code.activestate.com/recipes/522995-priority-dict-a-priority-queue-with-updatable-prio/ https://stackoverflow.com/questions/33024215/built-in-max-heap-api-in-python

Example

```
>>> # DISABLE_DOCTEST
>>> import utool as ut
>>> items = dict(a=42, b=29, c=40, d=95, e=10)
>>> self = ut.PriorityQueue(items)
>>> print(self)
>>> assert len(self) == 5
>>> print(self.pop())
>>> assert len(self) == 4
>>> print(self.pop())
>>> assert len(self) == 3
>>> print(self.pop())
>>> print(self.pop())
>>> print(self.pop())
>>> assert len(self) == 0
```

Example

```
>>> # DISABLE_DOCTEST
>>> import utool as ut
>>> items = dict(a=(1.0, (2, 3)), b=(1.0, (1, 2)), c=(.9, (3, 2)))
>>> self = ut.PriorityQueue(items)
```

Ignore: # TODO: can also use sortedcontainers to maintain priority queue import sortedcontainers queue = sortedcontainers.SortedListWithKey(items, key=lambda x: x[1]) queue.add((‘a’, 1))

clear()

delete_items(*key_list*)

get(*key*, *default=None*)

peek()
Peek at the next item in the queue

peek_many(*n*)
Actually this can be quite inefficient

Example

```
>>> # DISABLE_DOCTEST
>>> import utool as ut
>>> items = list(zip(range(256), range(256)))
>>> n = 32
>>> ut.shuffle(items)
>>> self = ut.PriorityQueue(items, ascending=False)
>>> self.peek_many(56)
```

pop(*key=NoParam*, *default=NoParam*)
Pop the next item off the queue

pop_many(*n*)

update(*items*)

class utool.util_dev.**Shortlist**(*maxsize=None*)
Bases: *utool.util_dev.NiceRepr*

Keeps an ordered collection of items. Removes smallest items if size grows to large.

Example

```
>>> # DISABLE_DOCTEST
>>> shortsize = 3
>>> shortlist = Shortlist(shortsize)
>>> print('shortlist = %r' % (shortlist,))
>>> item = (10, 1)
>>> shortlist.insert(item)
>>> print('shortlist = %r' % (shortlist,))
>>> item = (9, 1)
>>> shortlist.insert(item)
>>> print('shortlist = %r' % (shortlist,))
>>> item = (4, 1)
>>> shortlist.insert(item)
>>> print('shortlist = %r' % (shortlist,))
>>> item = (14, 1)
>>> shortlist.insert(item)
>>> print('shortlist = %r' % (shortlist,))
>>> item = (1, 1)
>>> shortlist.insert(item)
>>> print('shortlist = %r' % (shortlist,))
```

insert(*item*)

utool.util_dev.**are_you_sure**(*msg=''*)
Prompts user to accept or checks command line for -y

Parameters **msg** (*str*) –

Returns accept or not

Return type bool

utool.util_dev.**autopep8_diff**(*fpath*)

Parameters **fpath** (*str*) – file path string

CommandLine: python -m utool.util_dev –test-autopep8_diff –fpath ingest_data.py

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_dev import *  # NOQA
>>> fpath = ut.get_argval('--fpath', type_=str, default='ingest_data.py')
>>> result = autopep8_diff(fpath)
>>> print(result)
```

utool.util_dev.**compile_cython**(*fpath*, *clean=True*)

Compiles a cython pyx into a shared library

This seems broken compiles pyx -> pyd/dylib/so

Example

REAL SETUP.PY OUTPUT cythoning vtool/linalg_cython.pyx to vtoollinalg_cython.c C:MinGWbingcc.exe -mdll -O -Wall ^ -IC:Python27Libsite-packagesnumpycoreinclude ^ -IC:Python27include -IC:Python27PC ^ -c vtoollinalg_cython.c ^ -o buildtemp.win32-2.7Releasevtoollinalg_cython.o

writing buildtemp.win32-2.7Releasevtoollinalg_cython.def

C:MinGWbingcc.exe -shared -s buildtemp.win32-2.7Releasevtoollinalg_cython.o buildtemp.win32-2.7Releasevtoollinalg_cython.def -LC:Python27libs -LC:Python27PCbuild -lpython27 -lmsvcr90 -o buildlib.win32-2.7vtoollinalg_cython.pyd

utool.util_dev.**copy_text_to_clipboard**(*text*)

Copies text to the clipboard

CommandLine: pip install pyperclip sudo apt-get install xclip sudo apt-get install xsel

References

http://stackoverflow.com/questions/11063458/python-script-to-copy-text-to-clipboard http://stackoverflow.com/questions/579687/how-do-i-copy-a-string-to-the-clipboard-on-windows-using-python

Ignore:

```
>>> import pyperclip
>>> # Qt is by far the fastest, followed by xsel, and then xclip
>>> #
>>> backend_order = ['xclip', 'xsel', 'qt', 'gtk']
>>> backend_order = ['qt', 'xsel', 'xclip', 'gtk']
>>> for be in backend_order:
>>>     print('be = %r' % (be,))
>>>     pyperclip.set_clipboard(be)
>>>     %timeit pyperclip.copy('a line of reasonable length text')
>>>     %timeit pyperclip.paste()
```

utool.util_dev.**delayed_retry_gen**(*delay_schedule=[0.1, 1, 10], msg=None, timeout=None, raise_=True*)
template code for a infinte retry loop

utool.util_dev.**dev_ipython_copypaster**(*func*)

utool.util_dev.**disable_garbage_collection**()

utool.util_dev.**enable_garbage_collection**()

utool.util_dev.**ensure_str_list**(*input_*)

utool.util_dev.**ensureqt**()

utool.util_dev.**exec_argparse_funckw**(*func*, *globals_=None*, *defaults={}*, ***kwargs*)
for doctests kwargs

SeeAlso: ut.exec_func_src ut.argparse_funckw

utool.util_dev.**exec_funckw**(*func*, *globals_*)

SeeAlso: ut.argparse_funckw

utool.util_dev.**execstr_funckw**(*func*)
for doctests kwargs

SeeAlso: ut.exec_func_src ut.argparse_funckw

utool.util_dev.**extract_timeit_setup**(*func*, *doctest_number*, *sentinal*)

utool.util_dev.**find_exe**(*name*, *path_hints=[]*, *required=True*)

utool.util_dev.**fix_super_reload**(*this_class*, *self*)
Fixes an error where reload causes super(X, self) to raise an exception

The problem is that reloading causes X to point to the wrong version of the class. This function fixes the problem by searching and returning the correct version of the class. See example for proper usage.

USE *ut.super2* INSTEAD

Parameters

- **this_class** (*class*) – class passed into super
- **self** (*instance*) – instance passed into super

DisableExample:

```
>>> # DISABLE_DOCTEST
>>> import utool as ut
>>> class Parent(object):
>>>     def __init__(self):
>>>         self.parent_attr = 'bar'
>>> #
>>> class Foo(Parent):
>>>     def __init__(self):
>>>         # Dont do this, it will error if you reload
>>>         # super(Foo, self).__init__()
>>>         # Do this instead
>>>         _Foo = ut.super2(Foo, self)
>>>         super(_Foo, self).__init__()
>>> self = Foo()
>>> assert self.parent_attr == 'bar'
```

utool.util_dev.**focusvim**()

utool.util_dev.**garbage_collect**()

utool.util_dev.**get_clipboard**()

References

http://stackoverflow.com/questions/11063458/python-script-to-copy-text-to-clipboard

utool.util_dev.**get_cython_exe**()

utool.util_dev.**get_dev_paste_code**(*func*)

utool.util_dev.**get_jagged_stats**(*arr_list*, ***kwargs*)

Parameters **arr_list** (*list*) –

Returns stats_dict

Return type dict

CommandLine: python -m utool.util_dev –test-get_jagged_stats

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_dev import *  # NOQA
>>> import utool as ut
>>> kwargs = dict(use_nan=True)
>>> arr_list = [[1, 2, 3, 4], [3, 10], [np.nan, 3, 3, 3]]
>>> stats_dict = get_jagged_stats(arr_list, **kwargs)
>>> result = ut.align(str(ut.repr4(stats_dict)), ':')
>>> print(result)
{
    'mean'    : [2.5, 6.5, 3.0],
    'std'     : [1.118034, 3.5, 0.0],
    'max'     : [4.0, 10.0, 3.0],
    'min'     : [1.0, 3.0, 3.0],
    'nMin'    : [1, 1, 3],
    'nMax'    : [1, 1, 3],
    'shape'   : ['(4,)', '(2,)', '(4,)'],
    'num_nan': [0, 0, 1],
}
```

utool.util_dev.**get_nonconflicting_path_old**(*base_fmtstr*, *dpath*, *offset=0*)
base_fmtstr must have a %d in it

utool.util_dev.**get_nonconflicting_string**(*base_fmtstr*, *conflict_set*, *offset=0*)
gets a new string that wont conflict with something that already exists

Parameters

- **base_fmtstr** (*str*) –
- **conflict_set** (*set*) –

CommandLine: python -m utool.util_dev –test-get_nonconflicting_string

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dev import *  # NOQA
>>> # build test data
>>> base_fmtstr = 'somestring%d'
>>> conflict_set = ['somestring0']
>>> # execute function
>>> result = get_nonconflicting_string(base_fmtstr, conflict_set)
>>> # verify results
>>> print(result)
somestring1
```

utool.util_dev.**get_object_base**()

utool.util_dev.**get_object_nbytes**(*obj*, *fallback_type=None*, *follow_pointers=False*, *exclude_modules=True*, *listoverhead=False*)

CommandLine: python -m utool.util_dev –test-get_object_nbytes python -m utool.util_dev –test-get_object_nbytes:1

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dev import *  # NOQA
>>> import numpy as np
>>> import utool as ut
>>> obj = [np.empty(1, dtype=np.uint8) for _ in range(8)]
>>> nBytes = ut.get_object_nbytes(obj)
>>> result = ('nBytes = %s' % (nBytes,))
>>> print(result)
nBytes = 8
```

Example

```
>>> # DISABLE_DOCTEST
>>> # UNSTABLE_DOCTEST
>>> from utool.util_dev import *  # NOQA
>>> import wbia
>>> import utool as ut
>>> species = wbia.const.TEST_SPECIES.ZEB_PLAIN
>>> ibs = wbia.opendb(defaultdb='testdb1')
>>> qaids = ibs.get_valid_aids(species=species)
>>> daids = ibs.get_valid_aids(species=species)
>>> qreq_ = ibs.new_query_request(qaids, daids, verbose=True)
>>> nBytes = ut.get_object_nbytes(qreq_)
>>> result = (ut.byte_str2(nBytes))
>>> print('result = %r' % (result,))
```

Ignore:

```
>>> import sys
>>> sizedict = {key: sys.getsizeof(key()) for key in [dict, list, set, tuple,␣
↪int, float]}
```

(continues on next page)

(continued from previous page)

```
>>> ut.print_dict(sizedict)
>>> sizedict = {
>>>     <type 'tuple'>: 56,
>>>     <type 'set'>: 232,
>>>     <type 'list'>: 72,
>>>     <type 'float'>: 24,
>>>     <type 'int'>: 24,
>>>     <type 'dict'>: 280,
>>> }
```

utool.util_dev.**get_object_size_str**(*obj*, *lbl=''*, *unit=None*)

CommandLine: python -m utool.util_dev –test-get_object_size_str –show

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dev import *  # NOQA
>>> import numpy as np
>>> import utool as ut
>>> obj = [np.empty((512), dtype=np.uint8) for _ in range(10)]
>>> nBytes = ut.get_object_size_str(obj)
>>> result = ('result = %s' % (ut.repr2(nBytes),))
>>> print(result)
result = '5.00 KB'
```

utool.util_dev.**get_overlaps**(*set1*, *set2*, *s1='s1'*, *s2='s2'*)

utool.util_dev.**get_partial_func_name**(*func*, *precision=3*)

utool.util_dev.**get_setdiff_info**(*set1*, *set2*, *s1='s1'*, *s2='s2'*)

utool.util_dev.**get_setdiff_items**(*set1*, *set2*, *s1='s1'*, *s2='s2'*)

utool.util_dev.**get_statdict**(*list_*, *axis=None*, *use_nan=False*, *use_sum=False*, *use_median=False*, *size=False*)

Parameters

- **list** (*listlike*) – values to get statistics of
- **axis** (*int*) – if *list_* is ndarray then this specifies the axis

Returns

stats: dictionary of common numpy statistics (min, max, mean, std, nMin, nMax, shape)

Return type OrderedDict

SeeAlso: get_stats_str

CommandLine: python -m utool.util_dev –test-get_stats python -m utool.util_dev –test-get_stats:1

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dev import *  # NOQA
>>> import numpy as np
>>> import utool
>>> axis = 0
>>> np.random.seed(0)
>>> list_ = np.random.rand(10, 2).astype(np.float32)
>>> stats = get_stats(list_, axis, use_nan=False)
>>> result = str(utool.repr4(stats, nl=1, precision=4, with_dtype=True))
>>> print(result)
{
    'mean': np.array([0.5206, 0.6425], dtype=np.float32),
    'std': np.array([0.2854, 0.2517], dtype=np.float32),
    'max': np.array([0.9637, 0.9256], dtype=np.float32),
    'min': np.array([0.0202, 0.0871], dtype=np.float32),
    'nMin': np.array([1, 1], dtype=np.int32),
    'nMax': np.array([1, 1], dtype=np.int32),
    'shape': (10, 2),
}
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dev import *  # NOQA
>>> import numpy as np
>>> import utool
>>> axis = 0
>>> rng = np.random.RandomState(0)
>>> list_ = rng.randint(0, 42, size=100).astype(np.float32)
>>> list_[4] = np.nan
>>> stats = get_stats(list_, axis, use_nan=True)
>>> result = str(utool.repr2(stats, precision=1, strkeys=True))
>>> print(result)
{mean: 20.0, std: 13.2, max: 41.0, min: 0.0, nMin: 7, nMax: 3, shape: (100,), num_
↪nan: 1}
```

utool.util_dev.**get_stats**(*list_*, *axis=None*, *use_nan=False*, *use_sum=False*, *use_median=False*, *size=False*)

Parameters

- **list** (*listlike*) – values to get statistics of
- **axis** (*int*) – if *list_* is ndarray then this specifies the axis

Returns

stats: dictionary of common numpy statistics (min, max, mean, std, nMin, nMax, shape)

Return type OrderedDict

SeeAlso: get_stats_str

CommandLine: python -m utool.util_dev –test-get_stats python -m utool.util_dev –test-get_stats:1

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dev import *  # NOQA
>>> import numpy as np
>>> import utool
>>> axis = 0
>>> np.random.seed(0)
>>> list_ = np.random.rand(10, 2).astype(np.float32)
>>> stats = get_stats(list_, axis, use_nan=False)
>>> result = str(utool.repr4(stats, nl=1, precision=4, with_dtype=True))
>>> print(result)
{
    'mean': np.array([0.5206, 0.6425], dtype=np.float32),
    'std': np.array([0.2854, 0.2517], dtype=np.float32),
    'max': np.array([0.9637, 0.9256], dtype=np.float32),
    'min': np.array([0.0202, 0.0871], dtype=np.float32),
    'nMin': np.array([1, 1], dtype=np.int32),
    'nMax': np.array([1, 1], dtype=np.int32),
    'shape': (10, 2),
}
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dev import *  # NOQA
>>> import numpy as np
>>> import utool
>>> axis = 0
>>> rng = np.random.RandomState(0)
>>> list_ = rng.randint(0, 42, size=100).astype(np.float32)
>>> list_[4] = np.nan
>>> stats = get_stats(list_, axis, use_nan=True)
>>> result = str(utool.repr2(stats, precision=1, strkeys=True))
>>> print(result)
{mean: 20.0, std: 13.2, max: 41.0, min: 0.0, nMin: 7, nMax: 3, shape: (100,), num_
↪nan: 1}
```

utool.util_dev.**get_stats_str**(*list_=None*, *newlines=False*, *keys=None*, *exclude_keys=[]*, *lbl=None*, *precision=None*, *axis=0*, *stat_dict=None*, *use_nan=False*, *align=False*, *use_median=False*, ***kwargs*)

Returns the string version of get_stats

DEPRICATE in favor of ut.repr3(ut.get_stats(...))

if keys is not None then it only displays chosen keys excluded keys are always removed

CommandLine: python -m utool.util_dev –test-get_stats_str

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dev import *  # NOQA
>>> list_ = [1, 2, 3, 4, 5]
>>> newlines = False
```

(continues on next page)

(continued from previous page)

```
>>> keys = None
>>> exclude_keys = []
>>> lbl = None
>>> precision = 2
>>> stat_str = get_stats_str(list_, newlines, keys, exclude_keys, lbl, precision)
>>> result = str(stat_str)
>>> print(result)
{'mean': 3, 'std': 1.41, 'max': 5, 'min': 1, 'nMin': 1, 'nMax': 1, 'shape': (5,)}
```

SeeAlso: repr2 get_stats

utool.util_dev.**get_submodules_from_dpath**(*dpath*, *only_packages=False*, *recursive=True*)

Parameters

- **dpath** (*str*) – directory path
- **only_packages** (*bool*) – if True returns only package directories, otherwise returns module files. (default = False)

Returns submod_fpaths

Return type list

CommandLine: python -m utool.util_dev –exec-get_submodules_from_dpath –only_packages

Example

```
>>> # DISABLE_DOCTEST
>>> # SCRIPT
>>> from utool.util_dev import *  # NOQA
>>> import utool as ut
>>> dpath = ut.truepath_relative(ut.get_argval('--dpath', default='.'))
>>> print(dpath)
>>> only_packages = ut.get_argflag('--only_packages')
>>> submod_fpaths = get_submodules_from_dpath(dpath, only_packages)
>>> submod_fpaths = ut.lmap(ut.truepath_relative, submod_fpaths)
>>> result = ('submod_fpaths = %s' % (ut.repr3(submod_fpaths),))
>>> print(result)
```

utool.util_dev.**grace_period**(*msg=''*, *seconds=10*)
Gives user a window to stop a process before it happens

utool.util_dev.**init_catch_ctrl_c**()

utool.util_dev.**input_timeout**(*msg='Waiting for input...'*, *timeout=30*)
FIXME: Function does not work quite right yet.

Parameters

- **msg** (*str*) –
- **timeout** (*int*) –

Returns ans

Return type

?

References

http://stackoverflow.com/questions/1335507/keyboard-input-with-timeout-in-python http://home.wlu.edu/~levys/software/kbhit.py http://stackoverflow.com/questions/3471461/raw-input-and-timeout/3911560#3911560

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_dev import *  # NOQA
>>> msg = 'Waiting for input...'
>>> timeout = 30
>>> ans = input_timeout(msg, timeout)
>>> print(ans)
```

utool.util_dev.**instancelist**(*obj_list*, *check=False*, *shared_attrs=None*)
Executes methods and attribute calls on a list of objects of the same type

Bundles a list of object of the same type into a single object. The new object contains the same functions as each original object but applies them to each element of the list independantly when called.

CommandLine: python -m utool.util_dev instancelist

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dev import *  # NOQA
>>> import utool as ut
>>> obj_list = ['hi', 'bye', 'foo']
>>> self = ut.instancelist(obj_list, check=False)
>>> print(self)
>>> print(self.upper())
>>> print(self.isalpha())
```

utool.util_dev.**inverable_group_multi_list**(*item_lists*)
aid_list1 = np.array([1, 1, 2, 2, 3, 3]) aid2_list = np.array([4, 2, 1, 9, 8, 7]) item_lists = (np.array(aid1_list), np.array(aid2_list))

utool.util_dev.**inverable_unique_two_lists**(*item1_list*, *item2_list*)
item1_list = aid1_list item2_list = aid2_list

utool.util_dev.**ipcopydev**()

utool.util_dev.**is_developer**(*mycomputers=None*)

utool.util_dev.**iup**()
shortcut when pt is not imported

utool.util_dev.**make_at_least_n_items_valid**(*flag_list*, *n*)
tries to make at least min(len(flag_list, n) items True in flag_list

Parameters

- **flag_list** (*list*) – list of booleans
- **n** (*int*) – number of items to ensure are True

CommandLine: python -m utool.util_dev –test-make_at_least_n_items_valid

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dev import *  # NOQA
>>> # build test data
>>> flag_list = [False, True, False, False, False, False, False, True]
>>> n = 5
>>> # execute function
>>> flag_list = make_at_least_n_items_valid(flag_list, n)
>>> # verify results
>>> result = str(flag_list)
>>> print(result)
[ True  True  True  True False False False  True]
```

utool.util_dev.**make_call_graph**(*func*, **args*, ***kwargs*)
profile with pycallgraph

Example

pycallgraph graphviz – ./mypythonscript.py

References

http://pycallgraph.slowchop.com/en/master/

utool.util_dev.**make_object_graph**(*obj*, *fpath='sample_graph.png'*)
memoryprofile with objgraph

Example

#import objgraph #objgraph.show_most_common_types() #objgraph.show_growth() #memtrack.report() #memtrack.report() #objgraph.show_growth() #import gc #gc.collect() #memtrack.report() #y = 0 #objgraph.show_growth() #memtrack.report() #utool.embed()

References

http://mg.pov.lt/objgraph/

utool.util_dev.**memory_dump**()

References

http://stackoverflow.com/questions/141351/how-do-i-find-what-is-using-memory-in-a-python-process-in-a-production-system

utool.util_dev.**numpy_list_num_bits**(*nparr_list*, *expected_type*, *expected_dims*)

utool.util_dev.**overrideable_partial**(*func*, **args*, ***default_kwargs*)
like partial, but given kwargs can be overrideden at calltime

utool.util_dev.**pandas_reorder**(*df*, *order*)

utool.util_dev.**print_object_size**(*obj*, *lbl=''*)

utool.util_dev.**print_object_size_tree**(*obj*, *lbl='obj'*, *maxdepth=None*)
Needs work

utool.util_dev.**pylab_qt4**()

utool.util_dev.**report_memsize**(*obj*, *name=None*, *verbose=True*)

utool.util_dev.**reset_catch_ctrl_c**()

utool.util_dev.**search_module**(*mod*, *pat*, *ignore_case=True*, *recursive=False*, *_seen=None*)
Searches module functions, classes, and constants for members matching a pattern.

Parameters

- **mod** (*module*) – live python module
- **pat** (*str*) – regular expression

Returns found_list

Return type list

CommandLine: python -m utool.util_dev –exec-search_module –mod=utool –pat=module python -m utool.util_dev –exec-search_module –mod=opengm –pat=cut python -m utool.util_dev –exec-search_module –mod=opengm –pat=multi python -m utool.util_dev –exec-search_module –mod=utool –pat=Levenshtein

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dev import *  # NOQA
>>> import utool as ut
>>> recursive = True
>>> ignore_case = True
>>> modname = ut.get_argval('--mod', type_=str, default='utool')
>>> pat = ut.get_argval('--pat', type_=str, default='search')
>>> mod = ut.import_modname(modname)
>>> print('pat = %r' % (pat,))
>>> print('mod = %r' % (mod,))
>>> found_list = search_module(mod, pat, recursive=recursive)
>>> result = ('found_list = %s' % (ut.repr2(found_list),))
>>> print(result)
```

Ignore: mod = cv2 pat = ‘freak’

utool.util_dev.**search_utool**(*pat*)

utool.util_dev.**set_clipboard**(*text*)
alias for copy_text_to_clipboard

utool.util_dev.**set_overlap_items**(*set1*, *set2*, *s1='s1'*, *s2='s2'*)

utool.util_dev.**set_overlaps**(*set1*, *set2*, *s1='s1'*, *s2='s2'*)

utool.util_dev.**stats_dict**(*list_*, *axis=None*, *use_nan=False*, *use_sum=False*, *use_median=False*, *size=False*)

Parameters

- **list** (*listlike*) – values to get statistics of
- **axis** (*int*) – if *list_* is ndarray then this specifies the axis

Returns

stats: dictionary of common numpy statistics (min, max, mean, std, nMin, nMax, shape)

Return type OrderedDict

SeeAlso: get_stats_str

CommandLine: python -m utool.util_dev –test-get_stats python -m utool.util_dev –test-get_stats:1

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dev import *  # NOQA
>>> import numpy as np
>>> import utool
>>> axis = 0
>>> np.random.seed(0)
>>> list_ = np.random.rand(10, 2).astype(np.float32)
>>> stats = get_stats(list_, axis, use_nan=False)
>>> result = str(utool.repr4(stats, nl=1, precision=4, with_dtype=True))
>>> print(result)
{
    'mean': np.array([0.5206, 0.6425], dtype=np.float32),
    'std': np.array([0.2854, 0.2517], dtype=np.float32),
    'max': np.array([0.9637, 0.9256], dtype=np.float32),
    'min': np.array([0.0202, 0.0871], dtype=np.float32),
    'nMin': np.array([1, 1], dtype=np.int32),
    'nMax': np.array([1, 1], dtype=np.int32),
    'shape': (10, 2),
}
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dev import *  # NOQA
>>> import numpy as np
>>> import utool
>>> axis = 0
>>> rng = np.random.RandomState(0)
>>> list_ = rng.randint(0, 42, size=100).astype(np.float32)
>>> list_[4] = np.nan
>>> stats = get_stats(list_, axis, use_nan=True)
>>> result = str(utool.repr2(stats, precision=1, strkeys=True))
>>> print(result)
{mean: 20.0, std: 13.2, max: 41.0, min: 0.0, nMin: 7, nMax: 3, shape: (100,), num_
↪nan: 1}
```

utool.util_dev.**strip_line_comments**(*code_text*, *comment_char='#'*)

utool.util_dev.**super2**(*this_class*, *self*)

Fixes an error where reload causes super(X, self) to raise an exception

The problem is that reloading causes X to point to the wrong version of the class. This function fixes the problem by searching and returning the correct version of the class. See example for proper usage.

Parameters

- **this_class** (*class*) – class passed into super

- **self** (*instance*) – instance passed into super

Ignore:

```
>>> # ENABLE_DOCTEST
>>> # If the parent module is reloaded, the super call may fail
>>> # super(Foo, self).__init__()
>>> # This will work around the problem most of the time
>>> # ut.super2(Foo, self).__init__()
>>> import utool as ut
>>> class Parent(object):
>>>     def __init__(self):
>>>         self.parent_attr = 'bar'
>>> class ChildSafe(Parent):
>>>     def __init__(self):
>>>         ut.super2(ChildSafe, self).__init__()
>>> class ChildDanger(Parent):
>>>     def __init__(self):
>>>         super(ChildDanger, self).__init__()
>>> # initial loading is fine
>>> safe1 = ChildSafe()
>>> danger1 = ChildDanger()
>>> assert safe1.parent_attr == 'bar'
>>> assert danger1.parent_attr == 'bar'
>>> # But if we reload (via simulation), then there will be issues
>>> Parent_orig = Parent
>>> ChildSafe_orig = ChildSafe
>>> ChildDanger_orig = ChildDanger
>>> # reloading the changes the outer classname
>>> # but the inner class is still bound via the closure
>>> # (we simulate this by using the old functions)
>>> # (note in reloaded code the names would not change)
>>> class Parent_new(object):
>>>     __init__ = Parent_orig.__init__
>>> Parent_new.__name__ = 'Parent'
>>> class ChildSafe_new(Parent_new):
>>>     __init__ = ChildSafe_orig.__init__
>>> ChildSafe_new.__name__ = 'ChildSafe'
>>> class ChildDanger_new(Parent_new):
>>>     __init__ = ChildDanger_orig.__init__
>>> ChildDanger_new.__name__ = 'ChildDanger'
>>> #
>>> safe2 = ChildSafe_new()
>>> assert safe2.parent_attr == 'bar'
>>> import pytest
>>> with pytest.raises(TypeError):
>>>     danger2 = ChildDanger_new()
```

utool.util_dev.**timeit_compare**(*stmt_list*, *setup=''*, *iterations=100000*, *verbose=True*, *strict=False*, *assertsame=True*)

Compares several statments by timing them and also checks that they have the same return value

Parameters

- **stmt_list** (*list*) – list of statments to compare
- **setup** (*str*) –
- **iterations** (*int*) –

- **verbose** (*bool*) –
- **strict** (*bool*) –

Returns

(passed, time_list, result_list) passed (bool): True if all results are the same time_list (list): list of times for each statment result_list (list): list of results values for each statment

Return type tuple (bool, list, list)

CommandLine: python -m utool.util_dev –exec-timeit_compare

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dev import *  # NOQA
>>> import utool as ut
>>> setup = ut.codeblock(
        '''
        import numpy as np
        rng = np.random.RandomState(0)
        invVR_mats = rng.rand(1000, 3, 3).astype(np.float64)
        ''')
>>> stmt1 = 'invVR_mats[:, 0:2, 2].T'
>>> stmt2 = 'invVR_mats.T[2, 0:2]'
>>> iterations = 1000
>>> verbose = True
>>> stmt_list = [stmt1, stmt2]
>>> ut.timeit_compare(stmt_list, setup=setup, iterations=iterations,␣
↪verbose=verbose)
```

utool.util_dev.**timeit_grid**(*stmt_list*, *setup=''*, *iterations=10000*, *input_sizes=None*, *verbose=True*, *show=False*)

Timeit:

```
>>> import utool as ut
>>> setup = ut.codeblock(
>>>     '''
>>>     import utool as ut
>>>     from six.moves import range, zip
>>>     import time
>>>     def time_append(size):
>>>         start_time    = time.time()
>>>         last_time     = start_time
>>>         list2 = []
>>>         for x in range(size):
>>>             now_time    = time.time()
>>>             between = now_time - last_time
>>>             last_time   = now_time
>>>             list2.append(between)
>>>
>>>     def time_assign(size):
>>>         start_time    = time.time()
>>>         last_time     = start_time
>>>         list1 = ut.alloc_nones(size)
>>>         for x in range(size):
```

(continues on next page)

(continued from previous page)

```
>>>             now_time    = time.time()
>>>             between = now_time - last_time
>>>             last_time   = now_time
>>>             list1[x] = between
>>>
>>>     def time_baseline(size):
>>>         start_time    = time.time()
>>>         last_time     = start_time
>>>         for x in range(size):
>>>             now_time    = time.time()
>>>             between = now_time - last_time
>>>             last_time   = now_time
>>>
>>>     def time_null(size):
>>>         for x in range(size):
>>>             pass
>>>     ''')
>>>
>>> input_sizes = [2 ** count for count in range(7, 12)]
>>> stmt_list = ['time_assign', 'time_append', 'time_baseline', 'time_null']
>>> input_sizes=[100, 1000, 10000]
>>> ut.timeit_grid(stmt_list, setup, input_sizes=input_sizes, show=True)
```

utool.util_dev.**tuples_to_unique_scalars**(*tup_list*)

utool.util_dev.**uninvert_unique_two_lists**(*flat_list*, *reconstruct_tup*)
flat_list = thumb_list

utool.util_dev.**user_cmdline_prompt**(*msg=''*)

1.24 utool.util_dict module

convinience functions for dictionaries

utool.util_dict.**AutoDict**
alias of *utool.util_dict.AutoVivification*

utool.util_dict.**AutoOrderedDict**
alias of *utool.util_dict.OrderedAutoVivification*

class utool.util_dict.**AutoVivification**
Bases: dict

Implementation of perl's autovivification feature.

An AutoVivification is an infinitely nested default dict of dicts.

References

http://stackoverflow.com/questions/651794/best-way-to-init-dict-of-dicts

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dict import *  # NOQA
>>> dict_ = AutoVivification()
>>> # Notice that there is no KeyError
>>> dict_[0][10][100] = None
>>> result = ('dict_ = %r' % (dict_,))
>>> print(result)
dict_ = {0: {10: {100: None}}}
```

class utool.util_dict.**DefaultValueDict**(*default*, *other=None*, ***kwargs*)

Bases: dict

picklable default dictionary that can store scalar values.

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_dict import *  # NOQA
>>> import utool as ut
>>> self = ut.DefaultValueDict(0)
>>> print(self[4])
>>> self[4] = 4
>>> print(self[4])
0
4
```

class utool.util_dict.**DictLike**

Bases: object

move to util_dict rectify with util_dev

An inherited class must specify the `getitem`, `setitem`, and `keys` methods.

asdict()

copy()

delitem(*key*)

get(*key*, *default=None*)

getitem(*key*)

items()

iteritems()

iterkeys()

itervalues()

keys()

setitem(*key*, *value*)

values()

class utool.util_dict.**OrderedAutoVivification**

Bases: collections.OrderedDict

Implementation of perl's autovivification feature.

An OrderedAutoVivification is an infinitely nested default dict of ordered dicts.

References

http://stackoverflow.com/questions/651794/best-way-to-init-dict-of-dicts

Example

```
>>> from utool.util_dict import *  # NOQA
>>> dict_ = AutoOrderedDict()
>>> # Notice that there is no KeyError
>>> dict_[0][10][100] = None
>>> dict_[0][10][1] = None
>>> result = ('dict_ = %r' % (dict_,))
>>> print(result)
dict_ = {0: {10: {100: None, 1: None}}}
```

utool.util_dict.**all_dict_combinations**(*varied_dict*)

Parameters **varied_dict** (*dict*) – a dict with lists of possible parameter settings

Returns dict_list a list of dicts correpsonding to all combinations of params settings

Return type list

CommandLine: python -m utool.util_dict –test-all_dict_combinations

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dict import *  # NOQA
>>> import utool as ut
>>> varied_dict = {'logdist_weight': [0.0, 1.0], 'pipeline_root': ['vsmany'], 'sv_
↪on': [True, False, None]}
>>> dict_list = all_dict_combinations(varied_dict)
>>> result = str(ut.repr4(dict_list))
>>> print(result)
[
    {'logdist_weight': 0.0, 'pipeline_root': 'vsmany', 'sv_on': True},
    {'logdist_weight': 0.0, 'pipeline_root': 'vsmany', 'sv_on': False},
    {'logdist_weight': 0.0, 'pipeline_root': 'vsmany', 'sv_on': None},
    {'logdist_weight': 1.0, 'pipeline_root': 'vsmany', 'sv_on': True},
    {'logdist_weight': 1.0, 'pipeline_root': 'vsmany', 'sv_on': False},
    {'logdist_weight': 1.0, 'pipeline_root': 'vsmany', 'sv_on': None},
]
```

utool.util_dict.**all_dict_combinations_lbls**(*varied_dict*, *remove_singles=True*, *allow_lone_singles=False*)

returns a label for each variation in a varydict.

It tries to not be oververbose and returns only what parameters are varied in each label.

CommandLine: python -m utool.util_dict –test-all_dict_combinations_lbls python -m utool.util_dict –exec-all_dict_combinations_lbls:1

Example

```
>>> # ENABLE_DOCTEST
>>> import utool
>>> from utool.util_dict import *  # NOQA
>>> varied_dict = {'logdist_weight': [0.0, 1.0], 'pipeline_root': ['vsmany'], 'sv_
↪on': [True, False, None]}
>>> comb_lbls = utool.all_dict_combinations_lbls(varied_dict)
>>> result = (utool.repr4(comb_lbls))
>>> print(result)
[
    'logdist_weight=0.0,sv_on=True',
    'logdist_weight=0.0,sv_on=False',
    'logdist_weight=0.0,sv_on=None',
    'logdist_weight=1.0,sv_on=True',
    'logdist_weight=1.0,sv_on=False',
    'logdist_weight=1.0,sv_on=None',
]
```

Example

```
>>> # ENABLE_DOCTEST
>>> import utool as ut
>>> from utool.util_dict import *  # NOQA
>>> varied_dict = {'logdist_weight': [0.0], 'pipeline_root': ['vsmany'], 'sv_on':␣
↪[True]}
>>> allow_lone_singles = True
>>> comb_lbls = ut.all_dict_combinations_lbls(varied_dict, allow_lone_
↪singles=allow_lone_singles)
>>> result = (ut.repr4(comb_lbls))
>>> print(result)
[
    'logdist_weight=0.0,pipeline_root=vsmany,sv_on=True',
]
```

utool.util_dict.**all_dict_combinations_ordered**(*varied_dict*)
Same as all_dict_combinations but preserves order

utool.util_dict.**assert_keys_are_subset**(*dict1*, *dict2*)

Example

```
>>> # DISABLE_DOCTEST
>>> dict1 = {1:1, 2:2, 3:3}
>>> dict2 = {2:3, 3:3}
>>> assert_keys_are_subset(dict1, dict2)
>>> #dict2 = {4:3, 3:3}
```

utool.util_dict.**augdict**(*dict1*, *dict2=None*, ***kwargs*)

utool.util_dict.**build_conflict_dict**(*key_list*, *val_list*)
Builds dict where a list of values is associated with more than one key

Parameters

- **key_list** (*list*) –

- **val_list** (*list*) –

Returns key_to_vals

Return type dict

CommandLine: python -m utool.util_dict –test-build_conflict_dict

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dict import *  # NOQA
>>> import utool as ut
>>> key_list = [  1,   2,   2,   3,   1]
>>> val_list = ['a', 'b', 'c', 'd', 'e']
>>> key_to_vals = build_conflict_dict(key_list, val_list)
>>> result = ut.repr4(key_to_vals)
>>> print(result)
{
    1: ['a', 'e'],
    2: ['b', 'c'],
    3: ['d'],
}
```

utool.util_dict.**count_dict_vals**(*dict_of_lists*)

utool.util_dict.**delete_dict_keys**(*dict_*, *key_list*)

Removes items from a dictionary inplace. Keys that do not exist are ignored.

Parameters

- **dict** (*dict*) – dict like object with a __del__ attribute
- **key_list** (*list*) – list of keys that specify the items to remove

CommandLine: python -m utool.util_dict –test-delete_dict_keys

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dict import *  # NOQA
>>> import utool as ut
>>> dict_ = {'bread': 1, 'churches': 1, 'cider': 2, 'very small rocks': 2}
>>> key_list = ['duck', 'bread', 'cider']
>>> delete_dict_keys(dict_, key_list)
>>> result = ut.repr4(dict_, nl=False)
>>> print(result)
{'churches': 1, 'very small rocks': 2}
```

utool.util_dict.**delete_keys**(*dict_*, *key_list*)

Removes items from a dictionary inplace. Keys that do not exist are ignored.

Parameters

- **dict** (*dict*) – dict like object with a __del__ attribute
- **key_list** (*list*) – list of keys that specify the items to remove

CommandLine: python -m utool.util_dict –test-delete_dict_keys

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dict import *  # NOQA
>>> import utool as ut
>>> dict_ = {'bread': 1, 'churches': 1, 'cider': 2, 'very small rocks': 2}
>>> key_list = ['duck', 'bread', 'cider']
>>> delete_dict_keys(dict_, key_list)
>>> result = ut.repr4(dict_, nl=False)
>>> print(result)
{'churches': 1, 'very small rocks': 2}
```

utool.util_dict.**depth_atleast**(*list_*, *depth*)
Returns if depth of list is at least `depth`

Parameters

- **list** (*list*) –
- **depth** (*int*) –

Returns True

Return type bool

CommandLine: python -m utool.util_dict –exec-depth_atleast –show

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_dict import *  # NOQA
>>> import utool as ut
>>> list_ = [[[[0]]], [[0]]]
>>> depth = 0
>>> result = [depth_atleast(list_, depth) for depth in range(0, 7)]
>>> print(result)
```

utool.util_dict.**dict_accum**(**dict_list*)

utool.util_dict.**dict_assign**(*dict_*, *keys*, *vals*)
simple method for assigning or setting values with a similar interface to dict_take

utool.util_dict.**dict_filter_nones**(*dict_*)
Removes None values

Parameters **dict_** (*dict*) – a dictionary

Returns

Return type dict

CommandLine: python -m utool.util_dict –exec-dict_filter_nones

Example

```
>>> # DISABLE_DOCTEST
>>> # UNSTABLE_DOCTEST
>>> # fails on python 3 because of dict None order
>>> from utool.util_dict import *  # NOQA
>>> import utool as ut
>>> dict_ = {1: None, 2: 'blue', 3: 'four', None: 'fun'}
>>> dict2_ = dict_filter_nones(dict_)
>>> result = ut.repr4(dict2_, nl=False)
>>> print(result)
{None: 'fun', 2: 'blue', 3: 'four'}
```

utool.util_dict.**dict_find_keys**(*dict_*, *val_list*)

Parameters

- **dict** (*dict*) –
- **val_list** (*list*) –

Returns found_dict

Return type dict

CommandLine: python -m utool.util_dict –test-dict_find_keys

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dict import *  # NOQA
>>> import utool as ut
>>> dict_ = {'default': 1, 'hierarchical': 5, 'linear': 0, 'kdtree': 1,
...          'composite': 3, 'autotuned': 255, 'saved': 254, 'kmeans': 2,
...          'lsh': 6, 'kdtree_single': 4}
>>> val_list = [1]
>>> found_dict = dict_find_keys(dict_, val_list)
>>> result = ut.repr2(ut.map_vals(sorted, found_dict))
>>> print(result)
{1: ['default', 'kdtree']}
```

utool.util_dict.**dict_find_other_sameval_keys**(*dict_*, *key*)

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_dict import *  # NOQA
>>> dict_ = {'default': 1, 'hierarchical': 5, 'linear': 0, 'kdtree': 1,
...          'composite': 3, 'autotuned': 255, 'saved': 254, 'kmeans': 2,
...          'lsh': 6, 'kdtree_single': 4}
>>> key = 'default'
>>> found_dict = dict_find_keys(dict_, val_list)
```

utool.util_dict.**dict_hist**(*item_list*, *weight_list=None*, *ordered=False*, *labels=None*)

Builds a histogram of items in item_list

Parameters **item_list** (*list*) – list with hashable items (usually containing duplicates)

Returns

dictionary where the keys are items in item_list, and the values are the number of times the item appears in item_list.

Return type dict

CommandLine: python -m utool.util_dict –test-dict_hist

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dict import *  # NOQA
>>> import utool as ut
>>> item_list = [1, 2, 39, 900, 1232, 900, 1232, 2, 2, 2, 900]
>>> hist_ = dict_hist(item_list)
>>> result = ut.repr2(hist_)
>>> print(result)
{1: 1, 2: 4, 39: 1, 900: 3, 1232: 2}
```

utool.util_dict.**dict_hist_cumsum**(*hist_*, *reverse=True*)

VERY HACKY

utool.util_dict.**dict_intersection**(*dict1*, *dict2*, *combine=False*, *combine_op=<built-in function add>*)

Parameters

- **dict1** (*dict*) –
- **dict2** (*dict*) –
- **combine** (*bool*) – Combines keys only if the values are equal if False else values are combined using combine_op (default = False)
- **combine_op** (*func*) – (default = op.add)

Returns dict

CommandLine: python -m utool.util_dict –exec-dict_intersection

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dict import *  # NOQA
>>> import utool as ut
>>> dict1 = {'a': 1, 'b': 2, 'c': 3, 'd': 4}
>>> dict2 = {'b': 2, 'c': 3, 'd': 5, 'e': 21, 'f': 42}
>>> combine = False
>>> mergedict_ = dict_intersection(dict1, dict2, combine)
>>> result = ('mergedict_ = %s' % (ut.repr4(mergedict_, nl=False),))
>>> print(result)
mergedict_ = {'b': 2, 'c': 3}
```

utool.util_dict.**dict_isect**(*dict1*, *dict2*, *combine=False*, *combine_op=<built-in function add>*)

Parameters

- **dict1** (*dict*) –

- **dict2** (*dict*) –
- **combine** (*bool*) – Combines keys only if the values are equal if False else values are combined using combine_op (default = False)
- **combine_op** (*func*) – (default = op.add)

Returns dict

CommandLine: python -m utool.util_dict –exec-dict_intersection

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dict import *  # NOQA
>>> import utool as ut
>>> dict1 = {'a': 1, 'b': 2, 'c': 3, 'd': 4}
>>> dict2 = {'b': 2, 'c': 3, 'd': 5, 'e': 21, 'f': 42}
>>> combine = False
>>> mergedict_ = dict_intersection(dict1, dict2, combine)
>>> result = ('mergedict_ = %s' % (ut.repr4(mergedict_, nl=False),))
>>> print(result)
mergedict_ = {'b': 2, 'c': 3}
```

utool.util_dict.**dict_isect_combine**(*dict1*, *dict2*, *combine_op=<built-in function add>*)
Intersection of dict keys and combination of dict values

utool.util_dict.**dict_keysubset**(*dict_*, *keys*)

utool.util_dict.**dict_set_column**(*list_of_dicts_*, *colkey*, *value_list*)

utool.util_dict.**dict_setdiff**(*dict_*, *negative_keys*)
returns a copy of dict_ without keys in the negative_keys list

Parameters

- **dict_** (*dict*) –
- **negative_keys** (*list*) –

utool.util_dict.**dict_stack**(*dict_list*, *key_prefix=''*)
stacks values from two dicts into a new dict where the values are list of the input values. the keys are the same.

DEPRICATE in favor of dict_stack2

Parameters **dict_list** (*list*) – list of dicts with similar keys

Returns dict dict_stacked

CommandLine: python -m utool.util_dict –test-dict_stack python -m utool.util_dict –test-dict_stack:1

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dict import *  # NOQA
>>> import utool as ut
>>> dict1_ = {'a': 1, 'b': 2}
>>> dict2_ = {'a': 2, 'b': 3, 'c': 4}
>>> dict_stacked = dict_stack([dict1_, dict2_])
```

(continues on next page)

(continued from previous page)

```
>>> result = ut.repr2(dict_stacked, sorted_=True)
>>> print(result)
{'a': [1, 2], 'b': [2, 3], 'c': [4]}
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dict import *  # NOQA
>>> import utool as ut
>>> # Get equivalent behavior with dict_stack2?
>>> # Almost, as long as None is not part of the list
>>> dict1_ = {'a': 1, 'b': 2}
>>> dict2_ = {'a': 2, 'b': 3, 'c': 4}
>>> dict_stacked_ = dict_stack2([dict1_, dict2_])
>>> dict_stacked = {key: ut.filter_Nones(val) for key, val in dict_stacked_.
↪items()}
>>> result = ut.repr2(dict_stacked, sorted_=True)
>>> print(result)
{'a': [1, 2], 'b': [2, 3], 'c': [4]}
```

utool.util_dict.**dict_stack2**(*dict_list*, *key_suffix=None*, *default=None*)

Stacks vals from a list of dicts into a dict of lists. Inserts Nones in place of empty items to preserve order.

Parameters

- **dict_list** (*list*) – list of dicts
- **key_suffix** (*str*) – (default = None)

Returns stacked_dict

Return type dict

Example

```
>>> # ENABLE_DOCTEST
>>> # Usual case: multiple dicts as input
>>> from utool.util_dict import *  # NOQA
>>> import utool as ut
>>> dict1_ = {'a': 1, 'b': 2}
>>> dict2_ = {'a': 2, 'b': 3, 'c': 4}
>>> dict_list = [dict1_, dict2_]
>>> dict_stacked = dict_stack2(dict_list)
>>> result = ut.repr2(dict_stacked)
>>> print(result)
{'a': [1, 2], 'b': [2, 3], 'c': [None, 4]}
```

Example

```
>>> # ENABLE_DOCTEST
>>> # Corner case: one dict as input
>>> from utool.util_dict import *  # NOQA
>>> import utool as ut
```

(continues on next page)

(continued from previous page)

```
>>> dict1_ = {'a': 1, 'b': 2}
>>> dict_list = [dict1_]
>>> dict_stacked = dict_stack2(dict_list)
>>> result = ut.repr2(dict_stacked)
>>> print(result)
{'a': [1], 'b': [2]}
```

Example

```
>>> # ENABLE_DOCTEST
>>> # Corner case: zero dicts as input
>>> from utool.util_dict import *  # NOQA
>>> import utool as ut
>>> dict_list = []
>>> dict_stacked = dict_stack2(dict_list)
>>> result = ut.repr2(dict_stacked)
>>> print(result)
{}
```

Example

```
>>> # ENABLE_DOCTEST
>>> # Corner case: empty dicts as input
>>> from utool.util_dict import *  # NOQA
>>> import utool as ut
>>> dict_list = [{}]
>>> dict_stacked = dict_stack2(dict_list)
>>> result = ut.repr2(dict_stacked)
>>> print(result)
{}
```

Example

```
>>> # ENABLE_DOCTEST
>>> # Corner case: one dict is empty
>>> from utool.util_dict import *  # NOQA
>>> import utool as ut
>>> dict1_ = {'a': [1, 2], 'b': [2, 3]}
>>> dict2_ = {}
>>> dict_list = [dict1_, dict2_]
>>> dict_stacked = dict_stack2(dict_list)
>>> result = ut.repr2(dict_stacked)
>>> print(result)
{'a': [[1, 2], None], 'b': [[2, 3], None]}
```

Example

```
>>> # ENABLE_DOCTEST
>>> # Corner case: disjoint dicts
```

(continues on next page)

(continued from previous page)

```
>>> from utool.util_dict import *  # NOQA
>>> import utool as ut
>>> dict1_ = {'a': [1, 2], 'b': [2, 3]}
>>> dict2_ = {'c': 4}
>>> dict_list = [dict1_, dict2_]
>>> dict_stacked = dict_stack2(dict_list)
>>> result = ut.repr2(dict_stacked)
>>> print(result)
{'a': [[1, 2], None], 'b': [[2, 3], None], 'c': [None, 4]}
```

Example

```
>>> # ENABLE_DOCTEST
>>> # Corner case: 3 dicts
>>> from utool.util_dict import *  # NOQA
>>> import utool as ut
>>> dict_list = [{'a': 1}, {'b': 1}, {'c': 1}, {'b': 2}]
>>> default = None
>>> dict_stacked = dict_stack2(dict_list, default=default)
>>> result = ut.repr2(dict_stacked)
>>> print(result)
{'a': [1, None, None, None], 'b': [None, 1, None, 2], 'c': [None, None, 1, None]}
```

utool.util_dict.**dict_subset**(*dict_*, *keys*, *default=NoParam*)

Parameters

- **dict** (*dict*) –
- **keys** (*list*) –

Returns subset dictionary

Return type dict

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dict import *  # NOQA
>>> import utool as ut
>>> dict_ = {'K': 3, 'dcvs_clip_max': 0.2, 'p': 0.1}
>>> keys = ['K', 'dcvs_clip_max']
>>> d = tuple([])
>>> subdict_ = dict_subset(dict_, keys)
>>> result = ut.repr4(subdict_, sorted_=True, newlines=False)
>>> print(result)
{'K': 3, 'dcvs_clip_max': 0.2}
```

utool.util_dict.**dict_take**(*dict_*, *keys*, **d*)
get multiple values from a dictionary

utool.util_dict.**dict_take_asnametup**(*dict_*, *keys*, *name='_NamedTup'*)

utool.util_dict.**dict_take_column**(*list_of_dicts_*, *colkey*, *default=None*)

utool.util_dict.**dict_take_gen**(*dict_*, *keys*, **d*)
generate multiple values from a dictionary

Parameters

- **dict** (*dict*) –
- **keys** (*list*) –

Varargs: d: if specified is default for key errors

CommandLine: python -m utool.util_dict –test-dict_take_gen

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dict import *  # NOQA
>>> import utool as ut
>>> dict_ = {1: 'a', 2: 'b', 3: 'c'}
>>> keys = [1, 2, 3, 4, 5]
>>> result = list(dict_take_gen(dict_, keys, None))
>>> result = ut.repr4(result, nl=False)
>>> print(result)
['a', 'b', 'c', None, None]
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dict import *  # NOQA
>>> dict_ = {1: 'a', 2: 'b', 3: 'c'}
>>> keys = [1, 2, 3, 4, 5]
>>> try:
>>>     print(list(dict_take_gen(dict_, keys)))
>>>     result = 'did not get key error'
>>> except KeyError:
>>>     result = 'correctly got key error'
>>> print(result)
correctly got key error
```

utool.util_dict.**dict_take_list**(*dict_*, *keys*, **d*)
get multiple values from a dictionary

utool.util_dict.**dict_take_pop**(*dict_*, *keys*, **d*)
like dict_take but pops values off

CommandLine: python -m utool.util_dict –test-dict_take_pop

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dict import *  # NOQA
>>> import utool as ut
>>> dict_ = {1: 'a', 'other': None, 'another': 'foo', 2: 'b', 3: 'c'}
>>> keys = [1, 2, 3, 4, 5]
>>> print('before: ' + ut.repr4(dict_))
>>> result = list(dict_take_pop(dict_, keys, None))
>>> result = ut.repr4(result, nl=False)
```

(continues on next page)

(continued from previous page)

```
>>> print('after: ' + ut.repr4(dict_))
>>> assert len(dict_) == 2
>>> print(result)
['a', 'b', 'c', None, None]
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dict import *  # NOQA
>>> import utool as ut
>>> dict_ = {1: 'a', 2: 'b', 3: 'c'}
>>> keys = [1, 2, 3, 4, 5]
>>> print('before: ' + ut.repr4(dict_))
>>> try:
>>>     print(list(dict_take_pop(dict_, keys)))
>>>     result = 'did not get key error'
>>> except KeyError:
>>>     result = 'correctly got key error'
>>> assert len(dict_) == 0
>>> print('after: ' + ut.repr4(dict_))
>>> print(result)
correctly got key error
```

utool.util_dict.**dict_to_keyvals**(*dict_*)

utool.util_dict.**dict_union**(**args*)

utool.util_dict.**dict_union2**(*dict1*, *dict2*)

utool.util_dict.**dict_union3**(*dict1*, *dict2*, *combine_op=<built-in function add>*)

Parameters

- **dict1** (*dict*) –
- **dict2** (*dict*) –
- **combine_op** (*func*) – (default=op.add)

Returns dict

CommandLine: python -m utool.util_dict –exec-dict_union3

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dict import *  # NOQA
>>> import utool as ut
>>> dict1 = {'a': 1, 'b': 2, 'c': 3, 'd': 4}
>>> dict2 = {'b': 2, 'c': 3, 'd': 5, 'e': 21, 'f': 42}
>>> combine_op = op.add
>>> mergedict_ = dict_union3(dict1, dict2, combine_op)
>>> result = ('mergedict_ = %s' % (ut.repr4(mergedict_, nl=False),))
>>> print(result)
mergedict_ = {'a': 1, 'b': 4, 'c': 6, 'd': 9, 'e': 21, 'f': 42}
```

utool.util_dict.**dict_union_combine**(*dict1*, *dict2*, *combine_op=<built-in function add>*, *default=NoParam*, *default2=NoParam*)
Combine of dict keys and uses dfault value when key does not exist

CAREFUL WHEN USING THIS WITH REDUCE. Use dict_stack2 instead

utool.util_dict.**dict_update_newkeys**(*dict_*, *dict2*)
Like dict.update, but does not overwrite items

utool.util_dict.**dict_where_len0**(*dict_*)
Accepts a dict of lists. Returns keys that have vals with no length

utool.util_dict.**dictinfo**(*dict_*)
In depth debugging info

Parameters **dict** (*dict*) –

Returns str

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_dict import *  # NOQA
>>> dict_ = {}
>>> result = dictinfo(dict_)
>>> print(result)
```

utool.util_dict.**dzip**(*list1*, *list2*)
Zips elementwise pairs between list1 and list2 into a dictionary. Values from list2 can be broadcast onto list1.

Parameters

- **list1** (*sequence*) – full sequence
- **list2** (*sequence*) – can either be a sequence of one item or a sequence of equal length to *list1*

SeeAlso: util_list.broadcast_zip

Returns similar to dict(zip(list1, list2))

Return type dict

CommandLine: python -m utool.util_dict dzip

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_dict import *  # NOQA
>>> import utool as ut
>>> assert dzip([1, 2, 3], [4]) == {1: 4, 2: 4, 3: 4}
>>> assert dzip([1, 2, 3], [4, 4, 4]) == {1: 4, 2: 4, 3: 4}
>>> ut.assert_raises(ValueError, dzip, [1, 2, 3], [])
>>> ut.assert_raises(ValueError, dzip, [], [4, 5, 6])
>>> ut.assert_raises(ValueError, dzip, [], [4])
>>> ut.assert_raises(ValueError, dzip, [1, 2], [4, 5, 6])
>>> ut.assert_raises(ValueError, dzip, [1, 2, 3], [4, 5])
```

utool.util_dict.**flatten_dict_items**(*dict_*)

Flattens keys / values in a heirarchical dictionary

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dict import *  # NOQA
>>> import utool as ut
>>> item_list     = [1, 2, 3, 4]
>>> groupids_list = [[1, 1, 1, 2], [1, 2, 2, 2], [1, 3, 1, 1]]
>>> dict_ = hierarchical_group_items(item_list, groupids_list)
>>> flatter_dict = flatten_dict_items(dict_)
>>> result = ('flatter_dict = ' + ut.repr4(flatter_dict, nl=1))
>>> print(result)
flatter_dict = {
    (1, 1, 1): [1],
    (1, 2, 1): [3],
    (1, 2, 3): [2],
    (2, 2, 1): [4],
}
```

utool.util_dict.**flatten_dict_vals**(*dict_*)

Flattens only values in a heirarchical dictionary, keys are nested.

utool.util_dict.**get_dict_column**(*dict_*, *colx*)

Parameters

- **dict_** (*dict*) – a dictionary of lists
- **colx** (*int*) –

CommandLine: python -m utool.util_dict –test-get_dict_column

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dict import *  # NOQA
>>> import utool as ut
>>> dict_ = {'a': [0, 1, 2], 'b': [3, 4, 5], 'c': [6, 7, 8]}
>>> colx = [2, 0]
>>> retdict_ = get_dict_column(dict_, colx)
>>> result = ut.repr2(retdict_)
>>> print(result)
{'a': [2, 0], 'b': [5, 3], 'c': [8, 6]}
```

utool.util_dict.**get_dict_hashid**(*dict_*)

Parameters **dict_** (*dict*) –

Returns id hash

Return type int

References

http://stackoverflow.com/questions/5884066/hashing-a-python-dictionary

CommandLine: python -m utool.util_dict –test-get_dict_hashid python3 -m utool.util_dict –test-get_dict_hashid

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dict import *  # NOQA
>>> dict_ = {}
>>> dict_ = {'a': 'b'}
>>> dict_ = {'a': {'c': 'd'}}
>>> #dict_ = {'a': {'c': 'd'}, 1: 143, dict: set}
>>> #dict_ = {'a': {'c': 'd'}, 1: 143 } non-determenism
>>> hashid = get_dict_hashid(dict_)
>>> result = str(hashid)
>>> print(result)
mxgkepoboqjerkhb
```

oegknoalkrkojumi

utool.util_dict.**group_items**(*items*, *by=None*, *sorted_=True*)

Groups a list of items by group id.

Parameters

- **items** (*list*) – a list of the values to be grouped. if *by* is None, then each item is assumed to be a (groupid, value) pair.
- **by** (*list*) – a corresponding list to group items by. if specified, these are used as the keys to group values in *items*
- **sorted** (*bool*) – if True preserves the ordering of items within groups (default = True) FIXME. the opposite is true

Returns groupid_to_items: maps a groupid to a list of items

Return type dict

SeeAlso: group_indices - first part of a a more fine grained grouping algorithm apply_gropuing - second part of a more fine grained grouping algorithm

CommandLine: python -m utool.util_dict –test-group_items

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dict import *  # NOQA
>>> import utool as ut
>>> items = ['ham',     'jam',   'spam',     'eggs',    'cheese', 'bannana']
>>> by    = ['protein', 'fruit', 'protein',  'protein', 'dairy',  'fruit']
>>> groupid_to_items = ut.group_items(items, iter(by))
>>> result = ut.repr2(groupid_to_items)
>>> print(result)
{'dairy': ['cheese'], 'fruit': ['jam', 'bannana'], 'protein': ['ham', 'spam',
↪'eggs']}
```

utool.util_dict.**group_pairs**(*pair_list*)

Groups a list of items using the first element in each pair as the item and the second element as the groupid.

Parameters **pair_list** (*list*) – list of 2-tuples (item, groupid)

Returns groupid_to_items: maps a groupid to a list of items

Return type dict

SeeAlso: group_items

utool.util_dict.**groupby_attr**(*item_list*, *attrname*)

utool.util_dict.**groupby_tags**(*item_list*, *tags_list*)
case where an item can belong to multiple groups

Parameters

- **item_list** (*list*) –
- **tags_list** (*list*) –

Returns groupid_to_items

Return type dict

CommandLine: python -m utool.util_dict –test-groupby_tags

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dict import *  # NOQA
>>> import utool as ut
>>> tagged_item_list = {
>>>     'spam': ['meat', 'protein', 'food'],
>>>     'eggs': ['protein', 'food'],
>>>     'cheese': ['dairy', 'protein', 'food'],
>>>     'jam':  ['fruit', 'food'],
>>>     'banana': ['weapon', 'fruit', 'food'],
>>> }
>>> item_list = list(tagged_item_list.keys())
>>> tags_list = list(tagged_item_list.values())
>>> groupid_to_items = groupby_tags(item_list, tags_list)
>>> groupid_to_items = ut.map_vals(sorted, groupid_to_items)
>>> result = ('groupid_to_items = %s' % (ut.repr4(groupid_to_items),))
>>> print(result)
groupid_to_items = {
    'dairy': ['cheese'],
    'food': ['banana', 'cheese', 'eggs', 'jam', 'spam'],
    'fruit': ['banana', 'jam'],
    'meat': ['spam'],
    'protein': ['cheese', 'eggs', 'spam'],
    'weapon': ['banana'],
}
```

class utool.util_dict.**hashdict**
Bases: dict

hashable dict implementation, suitable for use as a key into other dicts.

Example

```
>>> # DISABLE_DOCTEST
>>> h1 = hashdict({"apples": 1, "bananas":2})
>>> h2 = hashdict({"bananas": 3, "mangoes": 5})
>>> h1+h2
hashdict(apples=1, bananas=3, mangoes=5)
>>> d1 = {}
>>> d1[h1] = "salad"
>>> d1[h1]
'salad'
>>> d1[h2]
Traceback (most recent call last):
...
KeyError: hashdict(bananas=3, mangoes=5)
```

References

http://stackoverflow.com/questions/1151658/python-hashable-dicts http://stackoverflow.com/questions/1151658/python-hashable-dicts

clear() → None. Remove all items from D.

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

popitem() → (k, v), remove and return some (key, value) pair as a
2-tuple; but raise KeyError if D is empty.

setdefault(**args*, ***kwargs*)
Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

update([*E*], ***F*) → None. Update D from dict/iterable E and F.
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

utool.util_dict.**hierarchical_group_items**(*item_list*, *groupids_list*)
Generalization of group_item. Convert a flast list of ids into a heirarchical dictionary.

TODO: move to util_dict

Reference: http://stackoverflow.com/questions/10193235/python-translate-a-table-to-a-hierarchical-dictionary

Parameters

- **item_list** (*list*) –
- **groupids_list** (*list*) –

CommandLine: python -m utool.util_dict –exec-hierarchical_group_items

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dict import *  # NOQA
>>> import utool as ut
>>> item_list     = [1, 2, 3, 4]
>>> groupids_list = [[1, 1, 2, 2]]
>>> tree = hierarchical_group_items(item_list, groupids_list)
>>> result = ('tree = ' + ut.repr4(tree, nl=len(groupids_list) - 1))
>>> print(result)
tree = {1: [1, 2], 2: [3, 4]}
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dict import *  # NOQA
>>> import utool as ut
>>> item_list     = [1, 2, 3, 4, 5, 6, 7, 8]
>>> groupids_list = [[1, 2, 1, 2, 1, 2, 1, 2], [3, 2, 2, 2, 3, 1, 1, 1]]
>>> tree = hierarchical_group_items(item_list, groupids_list)
>>> result = ('tree = ' + ut.repr4(tree, nl=len(groupids_list) - 1))
>>> print(result)
tree = {
    1: {1: [7], 2: [3], 3: [1, 5]},
    2: {1: [6, 8], 2: [2, 4]},
}
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dict import *  # NOQA
>>> import utool as ut
>>> item_list     = [1, 2, 3, 4]
>>> groupids_list = [[1, 1, 1, 2], [1, 2, 2, 2], [1, 3, 1, 1]]
>>> tree = hierarchical_group_items(item_list, groupids_list)
>>> result = ('tree = ' + ut.repr4(tree, nl=len(groupids_list) - 1))
>>> print(result)
tree = {
    1: {
        1: {1: [1]},
        2: {1: [3], 3: [2]},
    },
    2: {
        2: {1: [4]},
    },
}
```

utool.util_dict.**hierarchical_map_vals**(*func*, *node*, *max_depth=None*, *depth=0*)

node is a dict tree like structure with leaves of type list

TODO: move to util_dict

CommandLine: python -m utool.util_dict –exec-hierarchical_map_vals

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dict import *  # NOQA
>>> import utool as ut
>>> item_list     = [1, 2, 3, 4, 5, 6, 7, 8]
>>> groupids_list = [[1, 2, 1, 2, 1, 2, 1, 2], [3, 2, 2, 2, 3, 1, 1, 1]]
>>> tree = ut.hierarchical_group_items(item_list, groupids_list)
>>> len_tree = ut.hierarchical_map_vals(len, tree)
>>> result = ('len_tree = ' + ut.repr4(len_tree, nl=1))
>>> print(result)
len_tree = {
    1: {1: 1, 2: 1, 3: 2},
    2: {1: 2, 2: 2},
}
```

Example

```
>>> # DISABLE_DOCTEST
>>> # UNSTABLE_DOCTEST
>>> from utool.util_dict import *  # NOQA
>>> import utool as ut
>>> depth = 4
>>> item_list = list(range(2 ** (depth + 1)))
>>> num = len(item_list) // 2
>>> groupids_list = []
>>> total = 0
>>> for level in range(depth):
...     num2 = len(item_list) // int((num * 2))
...     #nonflat_levelids = [([total + 2 * x + 1] * num + [total + 2 * x + 2] *␣
↪num) for x in range(num2)]
...     nonflat_levelids = [([1] * num + [2] * num) for x in range(num2)]
...     levelids = ut.flatten(nonflat_levelids)
...     groupids_list.append(levelids)
...     total += num2 * 2
...     num //= 2
>>> print('groupids_list = %s' % (ut.repr4(groupids_list, nl=1),))
>>> print('depth = %r' % (len(groupids_list),))
>>> tree = ut.hierarchical_group_items(item_list, groupids_list)
>>> print('tree = ' + ut.repr4(tree, nl=None))
>>> flat_tree_values = list(ut.iflatten_dict_values(tree))
>>> assert sorted(flat_tree_values) == sorted(item_list)
>>> print('flat_tree_values = ' + str(flat_tree_values))
>>> #print('flat_tree_keys = ' + str(list(ut.iflatten_dict_keys(tree))))
>>> #print('iflatten_dict_items = ' + str(list(ut.iflatten_dict_items(tree))))
>>> len_tree = ut.hierarchical_map_vals(len, tree, max_depth=4)
>>> result = ('len_tree = ' + ut.repr4(len_tree, nl=None))
>>> print(result)
```

utool.util_dict.**hmap_vals**(*func*, *node*, *max_depth=None*, *depth=0*)

node is a dict tree like structure with leaves of type list

TODO: move to util_dict

CommandLine: python -m utool.util_dict –exec-hierarchical_map_vals

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dict import *  # NOQA
>>> import utool as ut
>>> item_list     = [1, 2, 3, 4, 5, 6, 7, 8]
>>> groupids_list = [[1, 2, 1, 2, 1, 2, 1, 2], [3, 2, 2, 2, 3, 1, 1, 1]]
>>> tree = ut.hierarchical_group_items(item_list, groupids_list)
>>> len_tree = ut.hierarchical_map_vals(len, tree)
>>> result = ('len_tree = ' + ut.repr4(len_tree, nl=1))
>>> print(result)
len_tree = {
    1: {1: 1, 2: 1, 3: 2},
    2: {1: 2, 2: 2},
}
```

Example

```
>>> # DISABLE_DOCTEST
>>> # UNSTABLE_DOCTEST
>>> from utool.util_dict import *  # NOQA
>>> import utool as ut
>>> depth = 4
>>> item_list = list(range(2 ** (depth + 1)))
>>> num = len(item_list) // 2
>>> groupids_list = []
>>> total = 0
>>> for level in range(depth):
...     num2 = len(item_list) // int((num * 2))
...     #nonflat_levelids = [([total + 2 * x + 1] * num + [total + 2 * x + 2] *␣
↪num) for x in range(num2)]
...     nonflat_levelids = [([1] * num + [2] * num) for x in range(num2)]
...     levelids = ut.flatten(nonflat_levelids)
...     groupids_list.append(levelids)
...     total += num2 * 2
...     num //= 2
>>> print('groupids_list = %s' % (ut.repr4(groupids_list, nl=1),))
>>> print('depth = %r' % (len(groupids_list),))
>>> tree = ut.hierarchical_group_items(item_list, groupids_list)
>>> print('tree = ' + ut.repr4(tree, nl=None))
>>> flat_tree_values = list(ut.iflatten_dict_values(tree))
>>> assert sorted(flat_tree_values) == sorted(item_list)
>>> print('flat_tree_values = ' + str(flat_tree_values))
>>> #print('flat_tree_keys = ' + str(list(ut.iflatten_dict_keys(tree))))
>>> #print('iflatten_dict_items = ' + str(list(ut.iflatten_dict_items(tree))))
>>> len_tree = ut.hierarchical_map_vals(len, tree, max_depth=4)
>>> result = ('len_tree = ' + ut.repr4(len_tree, nl=None))
>>> print(result)
```

utool.util_dict.**iflatten_dict_values**(*node*, *depth=0*)

```
>>> from utool.util_dict import *  # NOQA
```

utool.util_dict.**invert_dict**(*dict_*, *unique_vals=True*)

Reverses the keys and values in a dictionary. Set unique_vals to False if the values in the dict are not unique.

Parameters

- **dict_** (*dict*) – dictionary
- **unique_vals** (*bool*) – if False, inverted keys are returned in a list.

Returns inverted_dict

Return type dict

CommandLine: python -m utool.util_dict –test-invert_dict

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dict import *  # NOQA
>>> import utool as ut
>>> dict_ = {'a': 1, 'b': 2}
>>> inverted_dict = invert_dict(dict_)
>>> result = ut.repr4(inverted_dict, nl=False)
>>> print(result)
{1: 'a', 2: 'b'}
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dict import *  # NOQA
>>> import utool as ut
>>> dict_ = OrderedDict([(2, 'good',), (1, 'ok',), (0, 'junk',), (None, 'UNKNOWN',
↪)])
>>> inverted_dict = invert_dict(dict_)
>>> result = ut.repr4(inverted_dict, nl=False)
>>> print(result)
{'good': 2, 'ok': 1, 'junk': 0, 'UNKNOWN': None}
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dict import *  # NOQA
>>> import utool as ut
>>> dict_ = {'a': 1, 'b': 0, 'c': 0, 'd': 0, 'e': 0, 'f': 2}
>>> inverted_dict = invert_dict(dict_, unique_vals=False)
>>> inverted_dict = ut.map_dict_vals(sorted, inverted_dict)
>>> result = ut.repr4(inverted_dict, nl=False)
>>> print(result)
{0: ['b', 'c', 'd', 'e'], 1: ['a'], 2: ['f']}
```

utool.util_dict.**is_dicteq**(*dict1_*, *dict2_*, *almosteq_ok=True*, *verbose_err=True*)
Checks to see if dicts are the same. Performs recursion. Handles numpy

utool.util_dict.**iter_all_dict_combinations_ordered**(*varied_dict*)
Same as all_dict_combinations but preserves order

utool.util_dict.**iteritems_sorted**(*dict_*)
change to iteritems ordered

utool.util_dict.**keys_sorted_by_value**(*dict_*)

utool.util_dict.**map_dict_keys**(*func*, *dict_*)
applies a function to each of the keys in a dictionary

Parameters

- **func** (*callable*) – a function
- **dict_** (*dict*) – a dictionary

Returns transformed dictionary

Return type newdict

CommandLine: python -m utool.util_dict –test-map_dict_keys

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dict import *  # NOQA
>>> import utool as ut
>>> dict_ = {'a': [1, 2, 3], 'b': []}
>>> func = ord
>>> newdict = map_dict_keys(func, dict_)
>>> result = ut.repr2(newdict)
>>> ut.assert_raises(AssertionError, map_dict_keys, len, dict_)
>>> print(result)
{97: [1, 2, 3], 98: []}
```

utool.util_dict.**map_dict_vals**(*func*, *dict_*)
applies a function to each of the keys in a dictionary

Parameters

- **func** (*callable*) – a function
- **dict_** (*dict*) – a dictionary

Returns transformed dictionary

Return type newdict

CommandLine: python -m utool.util_dict –test-map_dict_vals

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dict import *  # NOQA
>>> import utool as ut
>>> dict_ = {'a': [1, 2, 3], 'b': []}
>>> func = len
>>> newdict = map_dict_vals(func, dict_)
>>> result = ut.repr2(newdict)
>>> print(result)
{'a': 3, 'b': 0}
```

utool.util_dict.**map_keys**(*func*, *dict_*)
applies a function to each of the keys in a dictionary

Parameters

- **func** (*callable*) – a function
- **dict_** (*dict*) – a dictionary

Returns transformed dictionary

Return type newdict

CommandLine: python -m utool.util_dict –test-map_dict_keys

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dict import *  # NOQA
>>> import utool as ut
>>> dict_ = {'a': [1, 2, 3], 'b': []}
>>> func = ord
>>> newdict = map_dict_keys(func, dict_)
>>> result = ut.repr2(newdict)
>>> ut.assert_raises(AssertionError, map_dict_keys, len, dict_)
>>> print(result)
{97: [1, 2, 3], 98: []}
```

utool.util_dict.**map_vals**(*func*, *dict_*)

applies a function to each of the keys in a dictionary

Parameters

- **func** (*callable*) – a function
- **dict_** (*dict*) – a dictionary

Returns transformed dictionary

Return type newdict

CommandLine: python -m utool.util_dict –test-map_dict_vals

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dict import *  # NOQA
>>> import utool as ut
>>> dict_ = {'a': [1, 2, 3], 'b': []}
>>> func = len
>>> newdict = map_dict_vals(func, dict_)
>>> result = ut.repr2(newdict)
>>> print(result)
{'a': 3, 'b': 0}
```

utool.util_dict.**merge_dicts**(**args*)

add / concatenate / union / join / merge / combine dictionaries

Copies the first dictionary given and then repeatedly calls update using the rest of the dicts given in args. Duplicate keys will receive the last value specified the list of dictionaries.

Returns dict

CommandLine: python -m utool.util_dict –test-merge_dicts

References

http://stackoverflow.com/questions/38987/how-can-i-merge-two-python-dictionaries-in-a-single-expression

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dict import *  # NOQA
>>> import utool as ut
>>> x = {'a': 1, 'b': 2}
>>> y = {'b': 3, 'c': 4}
>>> mergedict_ = merge_dicts(x, y)
>>> result = ut.repr4(mergedict_, sorted_=True, newlines=False)
>>> print(result)
{'a': 1, 'b': 3, 'c': 4}
```

utool.util_dict.**move_odict_item**(*odict*, *key*, *newpos*)

References

http://stackoverflow.com/questions/22663966/changing-order-of-ordered-dictionary-in-python

CommandLine: python -m utool.util_dict –exec-move_odict_item

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dict import *  # NOQA
>>> import utool as ut
>>> odict = OrderedDict()
>>> odict['a'] = 1
>>> odict['b'] = 2
>>> odict['c'] = 3
>>> odict['e'] = 5
>>> print(ut.repr4(odict, nl=False))
>>> move_odict_item(odict, 'c', 1)
>>> print(ut.repr4(odict, nl=False))
>>> move_odict_item(odict, 'a', 3)
>>> print(ut.repr4(odict, nl=False))
>>> move_odict_item(odict, 'a', 0)
>>> print(ut.repr4(odict, nl=False))
>>> move_odict_item(odict, 'b', 2)
>>> result = ut.repr4(odict, nl=False)
>>> print(result)
{'a': 1, 'c': 3, 'b': 2, 'e': 5}
```

utool.util_dict.**order_dict_by**(*dict_*, *key_order*)

Reorders items in a dictionary according to a custom key order

Parameters

- **dict_** (*dict*) – a dictionary

- **key_order** (*list*) – custom key order

Returns sorted_dict

Return type OrderedDict

CommandLine: python -m utool.util_dict –exec-order_dict_by

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dict import *  # NOQA
>>> import utool as ut
>>> dict_ = {1: 1, 2: 2, 3: 3, 4: 4}
>>> key_order = [4, 2, 3, 1]
>>> sorted_dict = order_dict_by(dict_, key_order)
>>> result = ('sorted_dict = %s' % (ut.repr4(sorted_dict, nl=False),))
>>> print(result)
>>> assert result == 'sorted_dict = {4: 4, 2: 2, 3: 3, 1: 1}'
```

utool.util_dict.**range_hist**(*items*, *bins*)

Bins items into a discrete histogram by values and/or ranges.

items = [1, 2, 3, 4, 5, 6, 7] bins = [0, 1, 2, (3, float('inf'))] ut.range_hist(items, bins)

utool.util_dict.**sort_dict**(*dict_*, *part='keys'*, *key=None*, *reverse=False*)

sorts a dictionary by its values or its keys

Parameters

- **dict_** (*dict*) – a dictionary
- **part** (*str*) – specifies to sort by keys or values
- **key** (*Optional[func]*) – a function that takes specified part and returns a sortable value
- **reverse** (*bool*) – (Defaults to False) - True for descinding order. False for ascending order.

Returns sorted dictionary

Return type OrderedDict

CommandLine: python -m utool.util_dict sort_dict

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dict import *  # NOQA
>>> import utool as ut
>>> dict_ = {'a': 3, 'c': 2, 'b': 1}
>>> results = []
>>> results.append(sort_dict(dict_, 'keys'))
>>> results.append(sort_dict(dict_, 'vals'))
>>> results.append(sort_dict(dict_, 'vals', lambda x: -x))
>>> result = ut.repr4(results)
>>> print(result)
[
```

(continues on next page)

(continued from previous page)

```
    {'a': 3, 'b': 1, 'c': 2},
    {'b': 1, 'c': 2, 'a': 3},
    {'a': 3, 'c': 2, 'b': 1},
]
```

utool.util_dict.**update_dict**(*dict1*, *dict2*, *copy=False*, *alias_dict=None*)

utool.util_dict.**update_existing**(*dict1*, *dict2*, *copy=False*, *assert_exists=False*, *iswarning=False*, *alias_dict=None*)

updates vals in dict1 using vals from dict2 only if the key is already in dict1.

Parameters

- **dict1** (*dict*) –
- **dict2** (*dict*) –
- **copy** (*bool*) – if true modifies dictionary in place (default = False)
- **assert_exists** (*bool*) – if True throws error if new key specified (default = False)
- **alias_dict** (*dict*) – dictionary of alias keys for dict2 (default = None)

Returns dict - updated dictionary

CommandLine: python -m utool.util_dict –test-update_existing

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_dict import *  # NOQA
>>> dict1 = {'a': 1, 'b': 2, 'c': 3}
>>> dict2 = {'a': 2, 'd': 3}
>>> dict1_ = update_existing(dict1, dict2)
>>> assert 'd' not in dict1
>>> assert dict1['a'] == 2
>>> assert dict1_ is dict1
```

1.25 utool.util_func module

utool.util_func.**compose_functions**(**func_list*)

Referenes: https://mathieularose.com/function-composition-in-python/

utool.util_func.**identity**(*input_*)

identity function

1.26 utool.util_git module

TODO: export from utool

python -m utool.util_inspect check_module_usage –pat=”util_git.py”

class utool.util_git.**Repo**(*url=None*, *code_dir=None*, *dpath=None*, *modname=None*, *python-cmd=None*)

Bases: *utool.util_dev.NiceRepr*

Handles a Python module repository

active_branch

active_remote

active_tracking_branch_name

active_tracking_remote_head

add_script(*key*, *script*)

aliases

as_gitpython()
pip install gitpython

branches

change_url_format(*out_type='ssh'*)
Changes the url format for committing

chdir_context(*verbose=False*)

check_cpp_build()

check_importable()

check_installed()

checkout2(*branch*, *overwrite=True*)
Checkout *branch* and automatically overwrites conflict files.

clone(*recursive=False*)

custom_build()

custom_install()

get_branch_remote(*branch*)

get_script(*type_*)

has_script(*type_*)

infer_info()

is_cloned()

is_gitrepo()

is_owner(*userid*)

issue(*command*, *sudo=False*, *dry=False*, *error='raise'*, *return_out=False*)
issues a command on a repo

CommandLine: python -m utool.util_git –exec-repocmd

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_git import *  # NOQA
>>> import utool as ut
>>> repo = dirname(ut.get_modpath(ut, prefer_pkg=True))
>>> command = 'git status'
>>> sudo = False
>>> result = repocmd(repo, command, sudo)
>>> print(result)
```

modname

owner()

pull(*has_submods=False*)

pull2(*overwrite=True*)
Pulls and automatically overwrites conflict files.

python_develop()

remotes

rename_branch(*old_branch_name*, *new_branch_name*, *remote='origin'*)

References

http://stackoverflow.com/questions/1526794/rename?answertab=votes#tab-top http://stackoverflow.com/questions/9524933/renaming-a-branch-in-github

CommandLine: python -m utool.util_git –test-rename_branch –old=mymaster –new=wbia_master

Example

```
>>> # DISABLE_DOCTEST
>>> # SCRIPT
>>> from utool.util_git import *  # NOQA
>>> repo = ut.get_argval('--repo', str, '.')
>>> remote = ut.get_argval('--remote', str, 'origin')
>>> old_branch_name = ut.get_argval('--old', str, None)
>>> new_branch_name = ut.get_argval('--new', str, None)
>>> rename_branch(old_branch_name, new_branch_name, repo, remote)
```

reponame

reset_branch_to_remote(*branch*, *hard=True*)
does a git reset –hard to whatever remote the branch is assigned to

static resolve_conflicts(*fpath*, *strat*, *force=False*, *verbose=True*)
Parses merge conflits and takes either version

rrr(*verbose=True*, *reload_module=True*)
special class reloading function This function is often injected as rrr of classes

set_branch_remote(*branch*, *remote*, *remote_branch=None*)

short_status()

CommandLine: python -m utool.util_git short_status

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_git import *  # NOQA
>>> import utool as ut
>>> repo = Repo(dpath=ut.truepath('.'))
>>> result = repo.short_status()
>>> print(result)
```

class utool.util_git.**RepoManager**(*repo_urls=None*, *code_dir=None*, *userid=None*, *permitted_repos=None*, *label=''*, *pythoncmd=None*)

Bases: *utool.util_dev.NiceRepr*

Batch git operations on multiple repos

add_repo(*repo*)

add_repos(*repo_urls=None*, *code_dir=None*)

check_cpp_build()

check_importable()

check_installed()

custom_build()

custom_install()

ensure()

issue(*command*, *sudo=False*)

Runs a command on all of managed repos

only_with_pysetup()

repo_dirs

repo_urls

rrr(*verbose=True*, *reload_module=True*)

special class reloading function This function is often injected as rrr of classes

union(*other*)

utool.util_git.**git_sequence_editor_squash**(*fpath*)

squashes wip messages

CommandLine: python -m utool.util_git –exec-git_sequence_editor_squash

Example

```
>>> # DISABLE_DOCTEST
>>> # SCRIPT
>>> import utool as ut
>>> from utool.util_git import *  # NOQA
>>> fpath = ut.get_argval('--fpath', str, default=None)
>>> git_sequence_editor_squash(fpath)
```

Ignore:

```
>>> text = ut.codeblock('''
>>>     pick 852aa05 better doctest for tips
>>>     pick 3c779b8 wip
>>>     pick 02bc21d wip
>>>     pick 1853828 Fixed root tablename
>>>     pick 9d50233 doctest updates
>>>     pick 66230a5 wip
>>>     pick c612e98 wip
>>>     pick b298598 Fixed tablename error
>>>     pick 1120a87 wip
>>>     pick f6c4838 wip
>>>     pick 7f92575 wip
>>> ''')
```

Notes

http://stackoverflow.com/questions/8226278/git-alias-to-squash-all-commits-with-a-particular-commit-message # Can do interactively with this. Can it be done automatically and pay attention to # Timestamps etc? git rebase –interactive HEAD~40 –autosquash git rebase –interactive $(git merge-base HEAD master) –autosquash

Lookbehind correct version %s/([a-z]* [a-z0-9]* wipn)@<=pick ([a-z0-9]*) wip/squash 2 wip/gc

THE FULL NON-INTERACTIVE AUTOSQUASH SCRIPT # TODO: Dont squash if there is a one hour timedelta between commits

GIT_EDITOR=”cat $1” GIT_SEQUENCE_EDITOR=”python -m utool.util_git –exec-git_sequence_editor_squash –fpath $1” git rebase -i $(git rev-list HEAD | tail -n 1) –autosquash –no-verify

GIT_EDITOR=”cat $1” GIT_SEQUENCE_EDITOR=”python -m utool.util_git –exec-git_sequence_editor_squash –fpath $1” git rebase -i HEAD~10 –autosquash –no-verify

GIT_EDITOR=”cat $1” GIT_SEQUENCE_EDITOR=”python -m utool.util_git –exec-git_sequence_editor_squash –fpath $1” git rebase -i $(git merge-base HEAD master) –autosquash –no-verify

14d778fa30a93f85c61f34d09eddb6d2cafd11e2 # c509a95d4468ebb61097bd9f4d302367424772a3 # b0ffc26011e33378ee30730c5e0ef1994bfe1a90 # GIT_SEQUENCE_EDITOR=<script> git rebase -i <params> # GIT_SEQUENCE_EDITOR=”echo ‘FOOBAR $1’ ” git rebase -i HEAD~40 –autosquash # git checkout master # git branch -D tmp # git checkout -b tmp # option to get the tail commit $(git rev-list HEAD | tail -n 1) # GIT_SEQUENCE_EDITOR=”python -m utool.util_git –exec-git_sequence_editor_squash # –fpath $1” git rebase -i HEAD~40 –autosquash # GIT_SEQUENCE_EDITOR=”python -m utool.util_git –exec-git_sequence_editor_squash # –fpath $1” git rebase -i HEAD~40 –autosquash –no-verify <params>

utool.util_git.**std_build_command**(*repo='.'*)

DEPRICATE My standard build script names.

Calls mingw_build.bat on windows and unix_build.sh on unix

1.27 utool.util_grabdata module

utool.util_grabdata.**archive_files**(*archive_fpath*, *fpath_list*, *small=True*, *allowZip64=False*, *overwrite=False*, *verbose=True*, *common_prefix=False*)

Adds the files in *fpath_list* to an zip/tar archive.

Parameters

- **archive_fpath** (*str*) – path to zipfile to create

- **fpath_list** (*list*) – path of files to add to the zipfile
- **small** (*bool*) – if True uses compression but the zipfile will take more time to write
- **allowZip64** (*bool*) – use if a file is over 2GB
- **overwrite** (*bool*) –
- **verbose** (*bool*) – verbosity flag(default = True)
- **common_prefix** (*bool*) – (default = False)

References

https://docs.python.org/2/library/zipfile.html

CommandLine: python -m utool.util_grabdata –test-archive_files

Ignore:

```
>>> # DISABLE_DOCTEST
>>> # SLOW_DOCTEST
>>> from utool.util_grabdata import *  # NOQA
>>> import utool as ut
>>> archive_fpath = ut.get_app_cache_dir('utool', 'testarchive.zip')
>>> # remove an existing test archive
>>> ut.delete(archive_fpath)
>>> assert not exists(archive_fpath), 'archive should not exist'
>>> fpath_list = [ut.grab_test_imgpath(key) for key in ut.TESTIMG_URL_DICT]
>>> small = True
>>> allowZip64 = False
>>> overwrite = True
>>> result = archive_files(archive_fpath, fpath_list, small, allowZip64,␣
↪overwrite)
>>> # verify results
>>> print(result)
>>> assert exists(archive_fpath), 'archive should exist'
```

Ignore: # http://superuser.com/questions/281573/best-options-compressing-files-7-zip # Create a small 7zip archive 7z a -t7z -m0=lzma -mx=9 -mfb=64 -md=32m -ms=on archive.7z dir1 7z a -t7z -m0=lzma -mx=9 -mfb=64 -md=32m -ms=on wbia-linux-binary.7z wbia

Create a small zip archive 7za a -mm=Deflate -mfb=258 -mpass=15 -r wbia-linux-binary.zip wbia

utool.util_grabdata.**clean_dropbox_link**(*dropbox_url*)

Dropbox links should be en-mass downloaed from dl.dropbox

DEPRICATE?

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_grabdata import *  # NOQA
>>> dropbox_url = 'www.dropbox.com/s/123456789abcdef/foobar.zip?dl=0'
>>> cleaned_url = clean_dropbox_link(dropbox_url)
>>> result = str(cleaned_url)
>>> print(result)
dl.dropbox.com/s/123456789abcdef/foobar.zip
```

utool.util_grabdata.**clear_test_img_cache**()

CommandLine: python -m utool.util_grabdata –test-clear_test_img_cache

Example

```
>>> # DISABLE_DOCTEST
>>> # UNSTABLE_DOCTEST
>>> from utool.util_grabdata import *  # NOQA
>>> testimg_fpath = clear_test_img_cache()
>>> result = str(testimg_fpath)
>>> print(result)
```

utool.util_grabdata.**download_url**(*url*, *filename=None*, *spoof=False*, *iri_fallback=True*, *verbose=True*, *new=True*, *chunk_size=None*)

downloads a url to a filename.

Parameters

- **url** (*str*) – url to download
- **filename** (*str*) – path to download to. Defaults to basename of url
- **spoof** (*bool*) – if True pretends to by Firefox
- **iri_fallback** – falls back to requests get call if there is a UnicodeError

References

http://blog.moleculea.com/2012/10/04/urlretrieve-progres-indicator/ http://stackoverflow.com/questions/15644964/python-progress-bar-and-downloads http://stackoverflow.com/questions/16694907/how-to-download-large-file-in-python-with-requests-py

Todo: Delete any partially downloaded files

Ignore:

```
>>> # DISABLE_DOCTEST
>>> from utool.util_grabdata import *  # NOQA
>>> url = 'http://www.jrsoftware.org/download.php/ispack.exe'
>>> fpath = download_url(url)
>>> print(fpath)
ispack.exe
```

utool.util_grabdata.**experiment_download_multiple_urls**(*url_list*)

References

http://stackoverflow.com/questions/1112343/capture-sigint-in-python http://stackoverflow.com/questions/16694907/download-large-file-requests GracefulInterruptHandler

Ignore:

```
>>> import signal
>>> import sys
>>> def signal_handler(signal, frame):
>>>     print('You pressed Ctrl+C!')
>>>     sys.exit(0)
>>> signal.signal(signal.SIGINT, signal_handler)
>>> print('Press Ctrl+C')
>>> signal.pause()
```

Example

```
>>> # DISABLE_DOCTEST
>>> # UNSTABLE_DOCTEST
>>> url_list = [
>>>     'https://wildbookiarepository.azureedge.net/random/ibeis-win32-setup-ymd_
↪hm-2015-08-01_16-28.exe',    # NOQA
>>>     'https://wildbookiarepository.azureedge.net/models/vgg.caffe.slice_0_30_
↪None.pickle',
>>>     'https://wildbookiarepository.azureedge.net/models/vgg.caffe.slice_0_30_
↪None.pickle',
>>>     'https://wildbookiarepository.azureedge.net/models/vgg.caffe.slice_0_30_
↪None.pickle',
>>>     'https://wildbookiarepository.azureedge.net/models/vgg.caffe.slice_0_30_
↪None.pickle',
>>>     'https://wildbookiarepository.azureedge.net/models/vgg.caffe.slice_0_30_
↪None.pickle',
>>>     'https://snapshotserengeti.s3.msi.umn.edu/S1/L10/L10_R1/S1_L10_R1_
↪PICT0070.JPG',
>>>     'https://snapshotserengeti.s3.msi.umn.edu/S1/B04/B04_R1/S1_B04_R1_
↪PICT0001.JPG',
>>>     'https://snapshotserengeti.s3.msi.umn.edu/S1/B04/B04_R1/S1_B04_R1_
↪PICT0002.JPG',
>>>     'https://snapshotserengeti.s3.msi.umn.edu/S1/B04/B04_R1/S1_B04_R1_
↪PICT0003.JPG',
>>>     'https://snapshotserengeti.s3.msi.umn.edu/S1/B04/B04_R1/S1_B04_R1_
↪PICT0004.JPG',
>>>     'https://snapshotserengeti.s3.msi.umn.edu/S1/B04/B04_R1/S1_B04_R1_
↪PICT0005.JPG',
>>>     'https://snapshotserengeti.s3.msi.umn.edu/S1/B04/B04_R1/S1_B04_R1_
↪PICT0006.JPG',
>>>     'https://snapshotserengeti.s3.msi.umn.edu/S1/B04/B04_R1/S1_B04_R1_
↪PICT0007.JPG',
>>>     'https://snapshotserengeti.s3.msi.umn.edu/S1/B04/B04_R1/S1_B04_R1_
↪PICT0008.JPG',
>>>     'https://snapshotserengeti.s3.msi.umn.edu/S1/B04/B04_R1/S1_B04_R1_
↪PICT0022.JPG',
>>>     'https://snapshotserengeti.s3.msi.umn.edu/S1/B04/B04_R1/S1_B04_R1_
↪PICT0023.JPG',
>>>     'https://snapshotserengeti.s3.msi.umn.edu/S1/B04/B04_R1/S1_B04_R1_
↪PICT0024.JPG',
>>>     'https://snapshotserengeti.s3.msi.umn.edu/S1/B04/B04_R1/S1_B04_R1_
↪PICT0025.JPG',
>>>     'https://snapshotserengeti.s3.msi.umn.edu/S1/B04/B04_R1/S1_B04_R1_
↪PICT0026.JPG',
>>>     'https://snapshotserengeti.s3.msi.umn.edu/S1/B04/B04_R1/S1_B04_R1_
↪PICT0027.JPG',
```

(continues on next page)

(continued from previous page)

```
>>>     'https://snapshotserengeti.s3.msi.umn.edu/S1/B04/B04_R1/S1_B04_R1_
↪PICT0028.JPG',
>>>     'https://snapshotserengeti.s3.msi.umn.edu/S1/B04/B04_R1/S1_B04_R1_
↪PICT0029.JPG'
>>> ]
```

utool.util_grabdata.**geo_locate**(*default='Unknown'*, *timeout=60.0*)

utool.util_grabdata.**get_file_local_hash**(*fpath*, *hash_list*, *verbose=False*)

utool.util_grabdata.**get_prefered_browser**(*pref_list=[]*, *fallback=True*)

Parameters

- **browser_preferences** (*list*) – (default = [])
- **fallback** (*bool*) – uses default if non of preferences work (default = True)

CommandLine: python -m utool.util_grabdata –test-get_prefered_browser

Ignore: import webbrowser webbrowser._tryorder pref_list = ['chrome', 'firefox', 'google-chrome'] pref_list = ['firefox', 'google-chrome']

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_grabdata import *  # NOQA
>>> browser_preferences = ['firefox', 'chrome', 'safari']
>>> fallback = True
>>> browser = get_prefered_browser(browser_preferences, fallback)
>>> result = ('browser = %s' % (str(browser),))
>>> print(result)
>>> ut.quit_if_noshow()
```

utool.util_grabdata.**get_valid_test_imgkeys**()
returns valid keys for grab_test_imgpath

utool.util_grabdata.**grab_file_remote_hash**(*file_url*, *hash_list*, *verbose=False*)

utool.util_grabdata.**grab_file_url**(*file_url*, *appname='utool'*, *download_dir=None*, *delay=None*, *spoof=False*, *fname=None*, *verbose=True*, *redownload=False*, *check_hash=False*, *attempts=3*)
Downloads a file and returns the local path of the file.

The resulting file is cached, so multiple calls to this function do not result in multiple dowloads.

Parameters

- **file_url** (*str*) – url to the file
- **appname** (*str*) – (default = 'utool')
- **custom directory** (*download_dir*) – (default = None)
- **delay** (*None*) – delay time before download (default = None)
- **spoof** (*bool*) – (default = False)
- **fname** (*str*) – custom file name (default = None)
- **verbose** (*bool*) – verbosity flag (default = True)

- **redownload** (*bool*) – if True forces redownload of the file (default = False)
- **check_hash** (*bool or iterable*) – if True, defaults to checking 4 hashes (in order): custom, md5, sha1, sha256. These hashes are checked for remote copies and, if found, will check the local file. You may also specify a list of hashes to check, for example ['md5', 'sha256'] in the specified order. The first verified hash to be found is used (default = False)

Returns fpath - file path string

Return type str

CommandLine: python -m utool.util_grabdata –test-grab_file_url:0 python -m utool.util_grabdata –test-grab_file_url:1

Ignore:

```
>>> # ENABLE_DOCTEST
>>> from utool.util_grabdata import *  # NOQA
>>> import utool as ut  # NOQA
>>> from os.path import basename
>>> ut.exec_funckw(ut.grab_file_url, locals())
>>> file_url = 'http://i.imgur.com/JGrqMnV.png'
>>> redownload = True
>>> fname = 'lena.png'
>>> lena_fpath = ut.grab_file_url(file_url, fname=fname,
>>>                               redownload=redownload)
>>> result = basename(lena_fpath)
>>> print(result)
lena.png
```

Ignore:

```
>>> # ENABLE_DOCTEST
>>> from utool.util_grabdata import *  # NOQA
>>> import utool as ut  # NOQA
>>> ut.exec_funckw(ut.grab_file_url, locals())
>>> file_url = 'https://wildbookiarepository.azureedge.net/models/detect.yolo.
↪12.classes'
>>> fname = 'detect.yolo.12.classes'
>>> check_hash = True
>>> fpath = ut.grab_file_url(file_url, fname=fname, check_hash=check_hash)
```

utool.util_grabdata.**grab_s3_contents**(*fpath*, *bucket*, *key*, *auth_access_id=None*, *auth_secret_key=None*, *auth_domain=None*)

utool.util_grabdata.**grab_selenium_chromedriver**(*redownload=False*)

Automatically download selenium chrome driver if needed

CommandLine: python -m utool.util_grabdata –test-grab_selenium_chromedriver:1

Example

```
>>> # DISABLE_DOCTEST
>>> ut.grab_selenium_chromedriver()
>>> import selenium.webdriver
>>> driver = selenium.webdriver.Chrome()
>>> driver.get('http://www.google.com')
```

(continues on next page)

(continued from previous page)

```
>>> search_field = driver.find_element_by_name('q')
>>> search_field.send_keys('puppies')
>>> search_field.send_keys(selenium.webdriver.common.keys.Keys.ENTER)
```

Example

```
>>> # DISABLE_DOCTEST
>>> import selenium.webdriver
>>> driver = selenium.webdriver.Firefox()
>>> driver.get('http://www.google.com')
>>> search_field = driver.find_element_by_name('q')
>>> search_field.send_keys('puppies')
>>> search_field.send_keys(selenium.webdriver.common.keys.Keys.ENTER)
```

utool.util_grabdata.**grab_selenium_driver**(*driver_name=None*)

pip install selenium -U

utool.util_grabdata.**grab_test_imgpath**(*key='lena.png'*, *allow_external=True*, *verbose=True*)

Gets paths to standard / fun test images. Downloads them if they dont exits

Parameters

- **key** (*str*) – one of the standard test images, e.g. lena.png, carl.jpg, ...
- **allow_external** (*bool*) – if True you can specify existing fpaths

Returns testimg_fpath - filepath to the downloaded or cached test image.

Return type str

SeeAlso: ut.get_valid_test_imgkeys

CommandLine: python -m utool.util_grabdata –test-grab_test_imgpath

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_grabdata import *  # NOQA
>>> import utool as ut
>>> # build test data
>>> key = 'carl.jpg'
>>> # execute function
>>> testimg_fpath = grab_test_imgpath(key)
>>> # verify results
>>> ut.assertpath(testimg_fpath)
```

utool.util_grabdata.**grab_zipped_url**(*zipped_url*, *ensure=True*, *appname='utool'*, *download_dir=None*, *force_commonprefix=True*, *cleanup=False*, *redownload=False*, *spoof=False*)

downloads and unzips the url

Parameters

- **zipped_url** (*str*) – url which must be either a .zip of a .tar.gz file
- **ensure** (*bool*) – eager evaluation if True(default = True)

- **appname** (*str*) – (default = 'utool')
- **download_dir** (*str*) – containing downloading directory
- **force_commonprefix** (*bool*) – (default = True)
- **cleanup** (*bool*) – (default = False)
- **redownload** (*bool*) – (default = False)
- **spoof** (*bool*) – (default = False)

CommandLine: python -m utool.util_grabdata –exec-grab_zipped_url –show

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_grabdata import *  # NOQA
>>> import utool as ut
>>> zipped_url = '?'
>>> ensure = True
>>> appname = 'utool'
>>> download_dir = None
>>> force_commonprefix = True
>>> cleanup = False
>>> redownload = False
>>> spoof = False
>>> result = grab_zipped_url(zipped_url, ensure, appname, download_dir,
>>>                          force_commonprefix, cleanup, redownload,
>>>                          spoof)
>>> print(result)
```

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_grabdata import *  # NOQA
>>> zipped_url = 'https://wildbookiarepository.azureedge.net/data/testdata.zip'
>>> zipped_url = 'http://www.spam.com/eggs/data.zip'
```

utool.util_grabdata.**list_remote**(*remote_uri*, *verbose=False*)

remote_uri = 'user@xx.xx.xx.xx'

utool.util_grabdata.**open_url_in_browser**(*url*, *browsername=None*, *fallback=False*)

Opens a url in the specified or default browser

Parameters **url** (*str*) – web url

CommandLine: python -m utool.util_grabdata –test-open_url_in_browser

Ignore:

```
>>> # DISABLE_DOCTEST
>>> # SCRIPT
>>> from utool.util_grabdata import *  # NOQA
>>> url = 'http://www.jrsoftware.org/isdl.php'
>>> open_url_in_browser(url, 'chrome')
```

utool.util_grabdata.**read_s3_contents**(*bucket*, *key*, *auth_access_id=None*, *auth_secret_key=None*, *auth_domain=None*)

utool.util_grabdata.**rsync**(*src_uri*, *dst_uri*, *exclude_dirs=[]*, *port=22*, *dryrun=False*)

Wrapper for rsync

General function to push or pull a directory from a remote server to a local path

Parameters

- **src_uri** (*str*) –
- **dst_uri** (*str*) –
- **exclude_dirs** (*list*) – (default = [])
- **port** (*int*) – (default = 22)
- **dryrun** (*bool*) – (default = False)

References

http://www.tecmint.com/rsync-local-remote-file-synchronization-commands/

CommandLine: python -m utool.util_grabdata rsync

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_grabdata import *  # NOQA
>>> import utool as ut
>>> src_uri = 'local/path/file.txt'
>>> dst_uri = 'host.cs.college.edge:/remove/path/file.txt'
>>> exclude_dirs = []
>>> port = 22
>>> dryrun = False
>>> result = rsync(src_uri, dst_uri, exclude_dirs, port, dryrun)
>>> print(result)
```

utool.util_grabdata.**s3_dict_encode_to_str**(*s3_dict*, *return_local_directory_if_available=False*)

utool.util_grabdata.**s3_str_decode_to_dict**(*s3_str*)

utool.util_grabdata.**scp_pull**(*remote_path*, *local_path='.'*, *remote='localhost'*, *user=None*)

wrapper for scp

utool.util_grabdata.**split_archive_ext**(*path*)

utool.util_grabdata.**unarchive_file**(*archive_fpath*, *force_commonprefix=True*, ***kwargs*)

utool.util_grabdata.**untar_file**(*targz_fpath*, *force_commonprefix=True*)

utool.util_grabdata.**unzip_file**(*zip_fpath*, *force_commonprefix=True*, *output_dir=None*, *prefix=None*, *dryrun=False*, *overwrite=None*)

utool.util_grabdata.**url_read**(*url*, *verbose=True*)

Directly reads data from url

utool.util_grabdata.**url_read_text**(*url*, *verbose=True*)

Directly reads text data from url

1.28 utool.util_graph module

utool.util_graph.**all_multi_paths**(*graph*, *source*, *target*, *data=False*)

Returns specific paths along multi-edges from the source to this table. Multipaths are identified by edge keys.

Returns all paths from source to target. This function treats multi-edges as distinct and returns the key value in each edge tuple that defines a path.

Example

```
>>> # DISABLE_DOCTEST
>>> from dtool.depcache_control import *  # NOQA
>>> from utool.util_graph import *  # NOQA
>>> from dtool.example_depcache import testdata_depc
>>> depc = testdata_depc()
>>> graph = depc.graph
>>> source = depc.root
>>> target = 'notchpair'
>>> path_list1 = ut.all_multi_paths(graph, depc.root, 'notchpair')
>>> path_list2 = ut.all_multi_paths(graph, depc.root, 'spam')
>>> result1 = ('path_list1 = %s' % ut.repr3(path_list1, nl=1))
>>> result2 = ('path_list2 = %s' % ut.repr3(path_list2, nl=2))
>>> result = '\n'.join([result1, result2])
>>> print(result)
path_list1 = [
    [('dummy_annot', 'notch', 0), ('notch', 'notchpair', 0)],
    [('dummy_annot', 'notch', 0), ('notch', 'notchpair', 1)],
]
path_list2 = [
    [
        ('dummy_annot', 'chip', 0),
        ('chip', 'keypoint', 0),
        ('keypoint', 'fgweight', 0),
        ('fgweight', 'spam', 0),
    ],
    [
        ('dummy_annot', 'chip', 0),
        ('chip', 'keypoint', 0),
        ('keypoint', 'spam', 0),
    ],
    [
        ('dummy_annot', 'chip', 0),
        ('chip', 'spam', 0),
    ],
    [
        ('dummy_annot', 'probchip', 0),
        ('probchip', 'fgweight', 0),
        ('fgweight', 'spam', 0),
    ],
]
```

utool.util_graph.**approx_min_num_components**(*nodes*, *negative_edges*)

Find approximate minimum number of connected components possible Each edge represents that two nodes must be separated

This code doesn't solve the problem. The problem is NP-complete and reduces to minimum clique cover (MCC). This is only an approximate solution. Not sure what the approximation ratio is.

CommandLine: python -m utool.util_graph approx_min_num_components

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_graph import *  # NOQA
>>> import utool as ut
>>> import networkx as nx
>>> nodes = [1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> edges = [(1, 2), (2, 3), (3, 1),
>>>          (4, 5), (5, 6), (6, 4),
>>>          (7, 8), (8, 9), (9, 7),
>>>          (1, 4), (4, 7), (7, 1),
>>>         ]
>>> g_pos = nx.Graph()
>>> g_pos.add_edges_from(edges)
>>> g_neg = nx.complement(g_pos)
>>> #import wbia.plottool as pt
>>> #pt.qt4ensure()
>>> #pt.show_nx(g_pos)
>>> #pt.show_nx(g_neg)
>>> negative_edges = g_neg.edges()
>>> nodes = [1, 2, 3, 4, 5, 6, 7]
>>> negative_edges = [(1, 2), (2, 3), (4, 5)]
>>> result = approx_min_num_components(nodes, negative_edges)
>>> print(result)
2
```

utool.util_graph.**bfs_conditional**(*G*, *source*, *reverse=False*, *keys=True*, *data=False*, *yield_nodes=True*, *yield_if=None*, *continue_if=None*, *visited_nodes=None*, *yield_source=False*)

Produce edges in a breadth-first-search starting at source, but only return nodes that satisfiy a condition, and only iterate past a node if it satisfies a different condition.

conditions are callables that take (G, child, edge) and return true or false

CommandLine: python -m utool.util_graph bfs_conditional

Example

```
>>> # DISABLE_DOCTEST
>>> import networkx as nx
>>> import utool as ut
>>> G = nx.Graph()
>>> G.add_edges_from([(1, 2), (1, 3), (2, 3), (2, 4)])
>>> continue_if = lambda G, child, edge: True
>>> result = list(ut.bfs_conditional(G, 1, yield_nodes=False))
>>> print(result)
[(1, 2), (1, 3), (2, 1), (2, 3), (2, 4), (3, 1), (3, 2), (4, 2)]
```

Example

```
>>> # ENABLE_DOCTEST
>>> import networkx as nx
>>> import utool as ut
>>> G = nx.Graph()
>>> continue_if = lambda G, child, edge: (child % 2 == 0)
>>> yield_if = lambda G, child, edge: (child % 2 == 1)
>>> G.add_edges_from([(0, 1), (1, 3), (3, 5), (5, 10),
>>>                   (4, 3), (3, 6),
>>>                   (0, 2), (2, 4), (4, 6), (6, 10)])
>>> result = list(ut.bfs_conditional(G, 0, continue_if=continue_if,
>>>                                  yield_if=yield_if))
>>> print(result)
[1, 3, 5]
```

utool.util_graph.**bfs_multi_edges**(*G*, *source*, *reverse=False*, *keys=True*, *data=False*)
Produce edges in a breadth-first-search starting at source.

Based on http://www.ics.uci.edu/~eppstein/PADS/BFS.py by D. Eppstein, July 2004.

utool.util_graph.**color_nodes**(*graph*, *labelattr='label'*, *brightness=0.878*, *outof=None*, *sat_adjust=None*)
Colors edges and nodes by nid

utool.util_graph.**dag_longest_path**(*graph*, *source*, *target*)
Finds the longest path in a dag between two nodes

utool.util_graph.**dfs_conditional**(*G*, *source*, *state*, *can_cross*)

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_graph import *
>>> G = nx.Graph()
>>> G.add_edges_from([(1, 2), (2, 3), (3, 4), (4, 5)])
>>> G.adj[2][3]['lava'] = True
>>> G.adj[3][4]['lava'] = True
>>> def can_cross(G, edge, state):
>>>     # can only cross lava once, then your lava protection wears off
>>>     data = G.get_edge_data(*edge)
>>>     lava = int(data.get('lava', False))
>>>     if not lava or state == 0:
>>>         return True, state + lava
>>>     return False, lava
>>> assert 5 not in dfs_conditional(G, 1, state=0, can_cross=can_cross)
>>> G.adj[3][4]['lava'] = False
>>> assert 5 in dfs_conditional(G, 1, state=0, can_cross=can_cross)
```

utool.util_graph.**dict_depth**(*dict_*, *accum=0*)

utool.util_graph.**edges_to_adjacency_list**(*edges*)

utool.util_graph.**get_allkeys**(*dict_*)

utool.util_graph.**get_graph_bounding_box**(*graph*)

utool.util_graph.**get_levels**(*dict_*, *n=0*, *levels=None*)
DEPCIRATE

Parameters

- **dict_** (*dict*) – a dictionary
- **n** (*int*) – (default = 0)
- **levels** (*None*) – (default = None)

CommandLine: python -m utool.util_graph –test-get_levels –show python3 -m utool.util_graph –test-get_levels –show

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_graph import *  # NOQA
>>> import utool as ut
>>> from_root = {
>>>     'dummy_annot': {
>>>         'chip': {
>>>                 'keypoint': {
>>>                             'fgweight': None,
>>>                         },
>>>             },
>>>         'probchip': {
>>>                 'fgweight': None,
>>>             },
>>>     },
>>> }
>>> dict_ = from_root
>>> n = 0
>>> levels = None
>>> levels_ = get_levels(dict_, n, levels)
>>> result = ut.repr2(levels_, nl=1)
>>> print(result)
[
    ['dummy_annot'],
    ['chip', 'probchip'],
    ['keypoint', 'fgweight'],
    ['fgweight'],
]
```

utool.util_graph.**graph_info**(*graph*, *ignore=None*, *stats=False*, *verbose=False*)

utool.util_graph.**greedy_mincost_diameter_augment**(*graph*, *max_cost*, *candidates=None*, *weight=None*, *cost=None*)

utool.util_graph.**longest_levels**(*levels_*)

Parameters **levels** (*list*) –

CommandLine: python -m utool.util_graph –exec-longest_levels –show

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_graph import *  # NOQA
>>> import utool as ut
>>> levels_ = [
```

(continues on next page)

(continued from previous page)

```
>>>     ['dummy_annot'],
>>>     ['chip', 'probchip'],
>>>     ['keypoint', 'fgweight'],
>>>     ['fgweight'],
>>> ]
>>> new_levels = longest_levels(levels_)
>>> result = ('new_levels = %s' % (ut.repr2(new_levels, nl=1),))
>>> print(result)
new_levels = [
    ['dummy_annot'],
    ['chip', 'probchip'],
    ['keypoint'],
    ['fgweight'],
]
```

utool.util_graph.**mincost_diameter_augment**(*graph*, *max_cost*, *candidates=None*, *weight=None*, *cost=None*)

PROBLEM: Bounded Cost Minimum Diameter Edge Addition (BCMD)

Parameters

- **graph** (*nx.Graph*) – input graph
- **max_cost** (*float*) – maximum weighted diamter of the graph
- **weight** (*str*) – key of the edge weight attribute
- **cost** (*str*) – key of the edge cost attribute
- **candidates** (*list*) – set of non-edges, optional, defaults to the complement of the graph

Returns if no solution exists list: minimum cost edges if solution exists

Return type None

Notes

We are given a graph G = (V, E) with an edge weight function w, an edge cost function c, an a maximum cost B.

The goal is to find a set of candidate non-edges F.

Let x[e] in {0, 1} denote if a non-edge e is excluded or included.

minimize sum(c(e) * x[e] for e in F) such that weighted_diamter(graph.union({e for e in F if x[e]})) <= B

References

https://www.cse.unsw.edu.au/~sergeg/papers/FratiGGM13isaac.pdf http://www.cis.upenn.edu/~sanjeev/papers/diameter.pdf http://dl.acm.org/citation.cfm?id=2953882

Notes

There is a 4-Approximation of the BCMD problem Running time is O((3 ** B * B ** 3 + n + log(B * n)) * B * n ** 2)

This algorithm usexs a clustering approach to find a set C, of B + 1 cluster centers. Then we create a minimum height rooted tree, T = (U subseteq V, D) so that C subseteq U. This tree T approximates an optimal B-augmentation.

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_graph import *  # NOQA
>>> import utool as ut
>>> import networkx as nx
>>> graph = nx.Graph()
>>> if nx.__version__.startswith('1'):
>>>     nx.add_path = nx.Graph.add_path
>>> nx.add_path(graph, range(6))
>>> #cost_func   = lambda e: e[0] + e[1]
>>> cost_func   = lambda e: 1
>>> weight_func = lambda e: (e[0]) / e[1]
>>> comp_graph = nx.complement(graph)
>>> nx.set_edge_attributes(graph, name='cost', values={e: cost_func(e) for e in␣
↪graph.edges()})
>>> nx.set_edge_attributes(graph, name='weight', values={e: weight_func(e) for e␣
↪in graph.edges()})
>>> nx.set_edge_attributes(comp_graph, name='cost', values={e: cost_func(e) for e␣
↪in comp_graph.edges()})
>>> nx.set_edge_attributes(comp_graph, name='weight', values={e: weight_func(e)␣
↪for e in comp_graph.edges()})
>>> candidates = list(comp_graph.edges(data=True))
>>> max_cost = 2
>>> cost = 'cost'
>>> weight = 'weight'
>>> best_edges = mincost_diameter_augment(graph, max_cost, candidates, weight,␣
↪cost)
>>> print('best_edges = %r' % (best_edges,))
>>> soln_edges = greedy_mincost_diameter_augment(graph, max_cost, candidates,␣
↪weight, cost)
>>> print('soln_edges = %r' % (soln_edges,))
```

utool.util_graph.**nx_all_nodes_between**(*graph*, *source*, *target*, *data=False*)
Find all nodes with on paths between source and target.

utool.util_graph.**nx_all_simple_edge_paths**(*G*, *source*, *target*, *cutoff=None*, *keys=False*, *data=False*)
Returns each path from source to target as a list of edges.

This function is meant to be used with MultiGraphs or MultiDiGraphs. When `keys` is True each edge in the path is returned with its unique key identifier. In this case it is possible to distinguish between different paths along different edges between the same two nodes.

Derived from simple_paths.py in networkx

utool.util_graph.**nx_common_ancestors**(*graph*, *node1*, *node2*)

utool.util_graph.**nx_common_descendants**(*graph*, *node1*, *node2*)

utool.util_graph.**nx_contracted_nodes**(*G*, *u*, *v*, *self_loops=True*, *inplace=False*)
copy of networkx function with inplace modification TODO: commit to networkx

utool.util_graph.**nx_dag_node_rank**(*graph*, *nodes=None*)
Returns rank of nodes that define the "level" each node is on in a topological sort. This is the same as the Graphviz dot rank.

Ignore: simple_graph = ut.simplify_graph(exi_graph) adj_dict = ut.nx_to_adj_dict(simple_graph) import wbia.plottool as pt pt.qt4ensure() pt.show_nx(graph)

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_graph import *  # NOQA
>>> import utool as ut
>>> import networkx as nx
>>> adj_dict = {0: [5], 1: [5], 2: [1], 3: [4], 4: [0], 5: [], 6: [4], 7: [9], 8:␣
↪[6], 9: [1]}
>>> nodes = [2, 1, 5]
>>> f_graph = ut.nx_from_adj_dict(adj_dict, nx.DiGraph)
>>> graph = f_graph.reverse()
>>> #ranks = ut.nx_dag_node_rank(graph, nodes)
>>> ranks = ut.nx_dag_node_rank(graph, nodes)
>>> result = ('ranks = %r' % (ranks,))
>>> print(result)
ranks = [3, 2, 1]
```

utool.util_graph.**nx_delete_None_edge_attr**(*graph*, *edges=None*)

utool.util_graph.**nx_delete_None_node_attr**(*graph*, *nodes=None*)

utool.util_graph.**nx_delete_edge_attr**(*graph*, *name*, *edges=None*)

Removes an attributes from specific edges in the graph

Example

```
>>> from utool.util_graph import *  # NOQA
>>> import utool as ut
>>> import networkx as nx
>>> G = nx.karate_club_graph()
>>> nx.set_edge_attributes(G, name='spam', values='eggs')
>>> nx.set_edge_attributes(G, name='foo', values='bar')
>>> assert len(nx.get_edge_attributes(G, 'spam')) == 78
>>> assert len(nx.get_edge_attributes(G, 'foo')) == 78
>>> ut.nx_delete_edge_attr(G, ['spam', 'foo'], edges=[(1, 2)])
>>> assert len(nx.get_edge_attributes(G, 'spam')) == 77
>>> assert len(nx.get_edge_attributes(G, 'foo')) == 77
>>> ut.nx_delete_edge_attr(G, ['spam'])
>>> assert len(nx.get_edge_attributes(G, 'spam')) == 0
>>> assert len(nx.get_edge_attributes(G, 'foo')) == 77
```

Example

```
>>> from utool.util_graph import *  # NOQA
>>> import utool as ut
>>> import networkx as nx
>>> G = nx.MultiGraph()
>>> G.add_edges_from([(1, 2), (2, 3), (3, 4), (4, 5), (4, 5), (1, 2)])
>>> nx.set_edge_attributes(G, name='spam', values='eggs')
>>> nx.set_edge_attributes(G, name='foo', values='bar')
>>> assert len(nx.get_edge_attributes(G, 'spam')) == 6
>>> assert len(nx.get_edge_attributes(G, 'foo')) == 6
>>> ut.nx_delete_edge_attr(G, ['spam', 'foo'], edges=[(1, 2, 0)])
>>> assert len(nx.get_edge_attributes(G, 'spam')) == 5
>>> assert len(nx.get_edge_attributes(G, 'foo')) == 5
```

(continues on next page)

(continued from previous page)

```
>>> ut.nx_delete_edge_attr(G, ['spam'])
>>> assert len(nx.get_edge_attributes(G, 'spam')) == 0
>>> assert len(nx.get_edge_attributes(G, 'foo')) == 5
```

utool.util_graph.**nx_delete_node_attr**(*graph*, *name*, *nodes=None*)

Removes node attributes

Example

```
>>> from utool.util_graph import *  # NOQA
>>> import utool as ut
>>> import networkx as nx
>>> G = nx.karate_club_graph()
>>> nx.set_node_attributes(G, name='foo', values='bar')
>>> datas = nx.get_node_attributes(G, 'club')
>>> assert len(nx.get_node_attributes(G, 'club')) == 34
>>> assert len(nx.get_node_attributes(G, 'foo')) == 34
>>> ut.nx_delete_node_attr(G, ['club', 'foo'], nodes=[1, 2])
>>> assert len(nx.get_node_attributes(G, 'club')) == 32
>>> assert len(nx.get_node_attributes(G, 'foo')) == 32
>>> ut.nx_delete_node_attr(G, ['club'])
>>> assert len(nx.get_node_attributes(G, 'club')) == 0
>>> assert len(nx.get_node_attributes(G, 'foo')) == 32
```

utool.util_graph.**nx_edges**(*graph*, *keys=False*, *data=False*)

utool.util_graph.**nx_edges_between**(*graph*, *nodes1*, *nodes2=None*, *assume_disjoint=False*, *assume_sparse=True*)

Get edges between two components or within a single component

Parameters

- **graph** (*nx.Graph*) – the graph
- **nodes1** (*set*) – list of nodes
- **nodes2** (*set*) – (default=None) if None it is equivlanet to nodes2=nodes1
- **assume_disjoint** (*bool*) – skips expensive check to ensure edges arnt returned twice (default=False)

CommandLine: python -m utool.util_graph –test-nx_edges_between

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_graph import *  # NOQA
>>> import utool as ut
>>> import networkx as nx
>>> edges = [
>>>     (1, 2), (2, 3), (3, 4), (4, 1), (4, 3),  # cc 1234
>>>     (1, 5), (7, 2), (5, 1),  # cc 567 / 5678
>>>     (7, 5), (5, 6), (8, 7),
>>> ]
>>> digraph = nx.DiGraph(edges)
```

(continues on next page)

(continued from previous page)

```
>>> graph = nx.Graph(edges)
>>> nodes1 = [1, 2, 3, 4]
>>> nodes2 = [5, 6, 7]
>>> n2 = sorted(nx_edges_between(graph, nodes1, nodes2))
>>> n4 = sorted(nx_edges_between(graph, nodes1))
>>> n5 = sorted(nx_edges_between(graph, nodes1, nodes1))
>>> n1 = sorted(nx_edges_between(digraph, nodes1, nodes2))
>>> n3 = sorted(nx_edges_between(digraph, nodes1))
>>> print('n2 == %r' % (n2,))
>>> print('n4 == %r' % (n4,))
>>> print('n5 == %r' % (n5,))
>>> print('n1 == %r' % (n1,))
>>> print('n3 == %r' % (n3,))
>>> assert n2 == ([(1, 5), (2, 7)]), '2'
>>> assert n4 == ([(1, 2), (1, 4), (2, 3), (3, 4)]), '4'
>>> assert n5 == ([(1, 2), (1, 4), (2, 3), (3, 4)]), '5'
>>> assert n1 == ([(1, 5), (5, 1), (7, 2)]), '1'
>>> assert n3 == ([(1, 2), (2, 3), (3, 4), (4, 1), (4, 3)]), '3'
>>> n6 = sorted(nx_edges_between(digraph, nodes1 + [6], nodes2 + [1, 2], assume_
↪sparse=True))
>>> print('n6 = %r' % (n6,))
>>> n6 = sorted(nx_edges_between(digraph, nodes1 + [6], nodes2 + [1, 2], assume_
↪sparse=False))
>>> print('n6 = %r' % (n6,))
>>> assert n6 == ([(1, 2), (1, 5), (2, 3), (4, 1), (5, 1), (5, 6), (7, 2)]), '6'
```

Timeit: from utool.util_graph import * # NOQA # ut.timeit_compare() import networkx as nx import utool as ut graph = nx.fast_gnp_random_graph(1000, .001) list(nx.connected_components(graph)) rng = np.random.RandomState(0) nodes1 = set(rng.choice(list(graph.nodes()), 500, replace=False)) nodes2 = set(graph.nodes()) - nodes1 edges_between = ut.nx_edges_between %timeit list(edges_between(graph, nodes1, nodes2, assume_sparse=False, assume_disjoint=True)) %timeit list(edges_between(graph, nodes1, nodes2, assume_sparse=False, assume_disjoint=False)) %timeit list(edges_between(graph, nodes1, nodes2, assume_sparse=True, assume_disjoint=False)) %timeit list(edges_between(graph, nodes1, nodes2, assume_sparse=True, assume_disjoint=True))

graph = nx.fast_gnp_random_graph(1000, .1) rng = np.random.RandomState(0) print(graph.number_of_edges()) nodes1 = set(rng.choice(list(graph.nodes()), 500, replace=False)) nodes2 = set(graph.nodes()) - nodes1 edges_between = ut.nx_edges_between %timeit list(edges_between(graph, nodes1, nodes2, assume_sparse=True, assume_disjoint=True)) %timeit list(edges_between(graph, nodes1, nodes2, assume_sparse=False, assume_disjoint=True))

Ignore: graph = nx.DiGraph(edges) graph = nx.Graph(edges) nodes1 = [1, 2, 3, 4] nodes2 = nodes1

utool.util_graph.**nx_ensure_agraph_color**(*graph*)
changes colors to hex strings on graph attrs

utool.util_graph.**nx_from_adj_dict**(*adj_dict*, *cls=None*)

utool.util_graph.**nx_from_matrix**(*weight_matrix*, *nodes=None*, *remove_self=True*)

utool.util_graph.**nx_from_node_edge**(*nodes=None*, *edges=None*)

utool.util_graph.**nx_gen_edge_attrs**(*G*, *key*, *edges=None*, *default=NoParam*, *on_missing='error'*, *on_keyerr='default'*)
Improved generator version of nx.get_edge_attributes

Parameters

- **on_missing** (*str*) – Strategy for handling nodes missing from G. Can be {'error', 'default', 'filter'}. defaults to 'error'. is on_missing is not error, then we allow any edge even if the endpoints are not in the graph.
- **on_keyerr** (*str*) – Strategy for handling keys missing from node dicts. Can be {'error', 'default', 'filter'}. defaults to 'default' if default is specified, otherwise defaults to 'error'.

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_graph import *  # NOQA
>>> import utool as ut
>>> import networkx as nx
>>> G = nx.Graph([(1, 2), (2, 3), (3, 4)])
>>> nx.set_edge_attributes(G, name='part', values={(1, 2): 'bar', (2, 3): 'baz'})
>>> edges = [(1, 2), (2, 3), (3, 4), (4, 5)]
>>> func = ut.partial(ut.nx_gen_edge_attrs, G, 'part', default=None)
>>> #
>>> assert len(list(func(on_missing='error', on_keyerr='default'))) == 3
>>> assert len(list(func(on_missing='error', on_keyerr='filter'))) == 2
>>> ut.assert_raises(KeyError, list, func(on_missing='error', on_keyerr='error'))
>>> #
>>> assert len(list(func(edges, on_missing='filter', on_keyerr='default'))) == 3
>>> assert len(list(func(edges, on_missing='filter', on_keyerr='filter'))) == 2
>>> ut.assert_raises(KeyError, list, func(edges, on_missing='filter', on_keyerr=
↪'error'))
>>> #
>>> assert len(list(func(edges, on_missing='default', on_keyerr='default'))) == 4
>>> assert len(list(func(edges, on_missing='default', on_keyerr='filter'))) == 2
>>> ut.assert_raises(KeyError, list, func(edges, on_missing='default', on_keyerr=
↪'error'))
```

utool.util_graph.**nx_gen_edge_values**(*G*, *key*, *edges=None*, *default=NoParam*, *on_missing='error'*, *on_keyerr='default'*)

Generates attributes values of specific edges

Parameters

- **on_missing** (*str*) – Strategy for handling nodes missing from G. Can be {'error', 'default'}. defaults to 'error'.
- **on_keyerr** (*str*) – Strategy for handling keys missing from node dicts. Can be {'error', 'default'}. defaults to 'default' if default is specified, otherwise defaults to 'error'.

utool.util_graph.**nx_gen_node_attrs**(*G*, *key*, *nodes=None*, *default=NoParam*, *on_missing='error'*, *on_keyerr='default'*)

Improved generator version of nx.get_node_attributes

Parameters

- **on_missing** (*str*) – Strategy for handling nodes missing from G. Can be {'error', 'default', 'filter'}. defaults to 'error'.
- **on_keyerr** (*str*) – Strategy for handling keys missing from node dicts. Can be {'error', 'default', 'filter'}. defaults to 'default' if default is specified, otherwise defaults to 'error'.

Notes

strategies are: error - raises an error if key or node does not exist default - returns node, but uses value specified by default filter - skips the node

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_graph import *  # NOQA
>>> import utool as ut
>>> import networkx as nx
>>> G = nx.Graph([(1, 2), (2, 3)])
>>> nx.set_node_attributes(G, name='part', values={1: 'bar', 3: 'baz'})
>>> nodes = [1, 2, 3, 4]
>>> #
>>> assert len(list(ut.nx_gen_node_attrs(G, 'part', default=None, on_missing=
↪'error', on_keyerr='default'))) == 3
>>> assert len(list(ut.nx_gen_node_attrs(G, 'part', default=None, on_missing=
↪'error', on_keyerr='filter'))) == 2
>>> ut.assert_raises(KeyError, list, ut.nx_gen_node_attrs(G, 'part', on_missing=
↪'error', on_keyerr='error'))
>>> #
>>> assert len(list(ut.nx_gen_node_attrs(G, 'part', nodes, default=None, on_
↪missing='filter', on_keyerr='default'))) == 3
>>> assert len(list(ut.nx_gen_node_attrs(G, 'part', nodes, default=None, on_
↪missing='filter', on_keyerr='filter'))) == 2
>>> ut.assert_raises(KeyError, list, ut.nx_gen_node_attrs(G, 'part', nodes, on_
↪missing='filter', on_keyerr='error'))
>>> #
>>> assert len(list(ut.nx_gen_node_attrs(G, 'part', nodes, default=None, on_
↪missing='default', on_keyerr='default'))) == 4
>>> assert len(list(ut.nx_gen_node_attrs(G, 'part', nodes, default=None, on_
↪missing='default', on_keyerr='filter'))) == 2
>>> ut.assert_raises(KeyError, list, ut.nx_gen_node_attrs(G, 'part', nodes, on_
↪missing='default', on_keyerr='error'))
```

Example

```
>>> # DISABLE_DOCTEST
>>> # ALL CASES
>>> from utool.util_graph import *  # NOQA
>>> import utool as ut
>>> import networkx as nx
>>> G = nx.Graph([(1, 2), (2, 3)])
>>> nx.set_node_attributes(G, name='full', values={1: 'A', 2: 'B', 3: 'C'})
>>> nx.set_node_attributes(G, name='part', values={1: 'bar', 3: 'baz'})
>>> nodes = [1, 2, 3, 4]
>>> attrs = dict(ut.nx_gen_node_attrs(G, 'full'))
>>> input_grid = {
>>>     'nodes': [None, (1, 2, 3, 4)],
>>>     'key': ['part', 'full'],
>>>     'default': [util_const.NoParam, None],
>>> }
>>> inputs = ut.all_dict_combinations(input_grid)
```

(continues on next page)

(continued from previous page)

```
>>> kw_grid = {
>>>     'on_missing': ['error', 'default', 'filter'],
>>>     'on_keyerr': ['error', 'default', 'filter'],
>>> }
>>> kws = ut.all_dict_combinations(kw_grid)
>>> for in_ in inputs:
>>>     for kw in kws:
>>>         kw2 = ut.dict_union(kw, in_)
>>>         #print(kw2)
>>>         on_missing = kw['on_missing']
>>>         on_keyerr = kw['on_keyerr']
>>>         if on_keyerr == 'default' and in_['default'] is util_const.NoParam:
>>>             on_keyerr = 'error'
>>>         will_miss = False
>>>         will_keyerr = False
>>>         if on_missing == 'error':
>>>             if in_['key'] == 'part' and in_['nodes'] is not None:
>>>                 will_miss = True
>>>             if in_['key'] == 'full' and in_['nodes'] is not None:
>>>                 will_miss = True
>>>         if on_keyerr == 'error':
>>>             if in_['key'] == 'part':
>>>                 will_keyerr = True
>>>             if on_missing == 'default':
>>>                 if in_['key'] == 'full' and in_['nodes'] is not None:
>>>                     will_keyerr = True
>>>         want_error = will_miss or will_keyerr
>>>         gen = ut.nx_gen_node_attrs(G, **kw2)
>>>         try:
>>>             attrs = list(gen)
>>>         except KeyError:
>>>             if not want_error:
>>>                 raise AssertionError('should not have errored')
>>>         else:
>>>             if want_error:
>>>                 raise AssertionError('should have errored')
```

utool.util_graph.**nx_gen_node_values**(*G*, *key*, *nodes*, *default=NoParam*)
Generates attributes values of specific nodes

utool.util_graph.**nx_get_default_edge_attributes**(*graph*, *key*, *default=None*)

utool.util_graph.**nx_get_default_node_attributes**(*graph*, *key*, *default=None*)

utool.util_graph.**nx_make_adj_matrix**(*G*)

utool.util_graph.**nx_mincut_edges_weighted**(*G*, *s*, *t*, *capacity='weight'*)

utool.util_graph.**nx_minimum_weight_component**(*graph*, *weight='weight'*)
A minimum weight component is an MST + all negative edges

utool.util_graph.**nx_node_dict**(*G*)

utool.util_graph.**nx_set_default_edge_attributes**(*graph*, *key*, *val*)

utool.util_graph.**nx_set_default_node_attributes**(*graph*, *key*, *val*)

utool.util_graph.**nx_sink_nodes**(*graph*)

utool.util_graph.**nx_source_nodes**(*graph*)

utool.util_graph.**nx_to_adj_dict**(*graph*)

utool.util_graph.**nx_topsort_nodes**(*graph*, *nodes*)

utool.util_graph.**nx_topsort_rank**(*graph*, *nodes=None*)
graph = inputs.exi_graph.reverse() nodes = flat_node_order_

utool.util_graph.**nx_transitive_reduction**(*G*, *mode=1*)

References

https://en.wikipedia.org/wiki/Transitive_reduction#Computing_the_reduction_using_the_closure
http://dept-info.labri.fr/~thibault/tmp/0201008.pdf http://stackoverflow.com/questions/17078696/transitive-reduction-of-directed-graph-in-python

CommandLine: python -m utool.util_graph nx_transitive_reduction –show

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_graph import *  # NOQA
>>> import utool as ut
>>> import networkx as nx
>>> G = nx.DiGraph([('a', 'b'), ('a', 'c'), ('a', 'e'),
>>>                 ('a', 'd'), ('b', 'd'), ('c', 'e'),
>>>                 ('d', 'e'), ('c', 'e'), ('c', 'd')])
>>> G = testdata_graph()[1]
>>> G_tr = nx_transitive_reduction(G, mode=1)
>>> G_tr2 = nx_transitive_reduction(G, mode=1)
>>> ut.quit_if_noshow()
>>> try:
>>>     import wbia.plottool as pt
>>> except ImportError:
>>>     import wbia.plottool as pt
>>> G_ = nx.dag.transitive_closure(G)
>>> pt.show_nx(G    , pnum=(1, 5, 1), fnum=1)
>>> pt.show_nx(G_tr , pnum=(1, 5, 2), fnum=1)
>>> pt.show_nx(G_tr2 , pnum=(1, 5, 3), fnum=1)
>>> pt.show_nx(G_   , pnum=(1, 5, 4), fnum=1)
>>> pt.show_nx(nx.dag.transitive_closure(G_tr), pnum=(1, 5, 5), fnum=1)
>>> ut.show_if_requested()
```

utool.util_graph.**paths_to_root**(*tablename*, *root*, *child_to_parents*)

CommandLine: python -m utool.util_graph –exec-paths_to_root:0 python -m utool.util_graph –exec-paths_to_root:1

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_graph import *  # NOQA
>>> import utool as ut
>>> import networkx as nx
>>> child_to_parents = {
>>>     'chip': ['dummy_annot'],
```

(continues on next page)

(continued from previous page)

```
>>>     'chipmask': ['dummy_annot'],
>>>     'descriptor': ['keypoint'],
>>>     'fgweight': ['keypoint', 'probchip'],
>>>     'keypoint': ['chip'],
>>>     'notch': ['dummy_annot'],
>>>     'probchip': ['dummy_annot'],
>>>     'spam': ['fgweight', 'chip', 'keypoint']
>>> }
>>> root = 'dummy_annot'
>>> tablename = 'fgweight'
>>> to_root = paths_to_root(tablename, root, child_to_parents)
>>> result = ut.repr3(to_root)
>>> print(result)
{
    'keypoint': {
        'chip': {
            'dummy_annot': None,
        },
    },
    'probchip': {
        'dummy_annot': None,
    },
}
```

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_graph import *  # NOQA
>>> import utool as ut
>>> root = u'annotations'
>>> tablename = u'Notch_Tips'
>>> child_to_parents = {
>>>     'Block_Curvature': [
>>>         'Trailing_Edge',
>>>     ],
>>>     'Has_Notch': [
>>>         'annotations',
>>>     ],
>>>     'Notch_Tips': [
>>>         'annotations',
>>>     ],
>>>     'Trailing_Edge': [
>>>         'Notch_Tips',
>>>     ],
>>> }
>>> to_root = paths_to_root(tablename, root, child_to_parents)
>>> result = ut.repr3(to_root)
>>> print(result)
```

utool.util_graph.**reverse_path**(*dict_*, *root*, *child_to_parents*)

CommandLine: python -m utool.util_graph –exec-reverse_path –show

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_graph import *  # NOQA
>>> import utool as ut
>>> child_to_parents = {
>>>     'chip': ['dummy_annot'],
>>>     'chipmask': ['dummy_annot'],
>>>     'descriptor': ['keypoint'],
>>>     'fgweight': ['keypoint', 'probchip'],
>>>     'keypoint': ['chip'],
>>>     'notch': ['dummy_annot'],
>>>     'probchip': ['dummy_annot'],
>>>     'spam': ['fgweight', 'chip', 'keypoint']
>>> }
>>> to_root = {
>>>     'fgweight': {
>>>         'keypoint': {
>>>             'chip': {
>>>                 'dummy_annot': None,
>>>             },
>>>         },
>>>         'probchip': {
>>>             'dummy_annot': None,
>>>         },
>>>     },
>>> }
>>> reversed_ = reverse_path(to_root, 'dummy_annot', child_to_parents)
>>> result = ut.repr3(reversed_)
>>> print(result)
{
    'dummy_annot': {
        'chip': {
            'keypoint': {
                'fgweight': None,
            },
        },
        'probchip': {
            'fgweight': None,
        },
    },
}
```

utool.util_graph.**reverse_path_edges**(*edge_list*)

utool.util_graph.**shortest_levels**(*levels_*)

Parameters **levels** (*list*) –

CommandLine: python -m utool.util_graph –exec-shortest_levels –show

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_graph import *  # NOQA
>>> import utool as ut
```

(continues on next page)

(continued from previous page)

```
>>> levels_ = [
>>>     ['dummy_annot'],
>>>     ['chip', 'probchip'],
>>>     ['keypoint', 'fgweight'],
>>>     ['fgweight'],
>>> ]
>>> new_levels = shortest_levels(levels_)
>>> result = ('new_levels = %s' % (ut.repr2(new_levels, nl=1),))
>>> print(result)
new_levels = [
    ['dummy_annot'],
    ['chip', 'probchip'],
    ['keypoint', 'fgweight'],
]
```

utool.util_graph.**simplify_graph**(*graph*)

strips out everything but connectivity

Parameters **graph** (*nx.Graph*) –

Returns new_graph

Return type nx.Graph

CommandLine: python3 -m utool.util_graph simplify_graph –show python2 -m utool.util_graph simplify_graph –show

python2 -c "import networkx as nx; print(nx.__version__)" python3 -c "import networkx as nx; print(nx.__version__)"

Ignore:

```
>>> # ENABLE_DOCTEST
>>> from utool.util_graph import *  # NOQA
>>> import utool as ut
>>> import networkx as nx
>>> graph = nx.DiGraph([('a', 'b'), ('a', 'c'), ('a', 'e'),
>>>                     ('a', 'd'), ('b', 'd'), ('c', 'e'),
>>>                     ('d', 'e'), ('c', 'e'), ('c', 'd')])
>>> new_graph = simplify_graph(graph)
>>> result = ut.repr2(list(new_graph.edges()))
>>> #adj_list = sorted(list(nx.generate_adjlist(new_graph)))
>>> #result = ut.repr2(adj_list)
>>> print(result)
[(0, 1), (0, 2), (0, 3), (0, 4), (1, 3), (2, 3), (2, 4), (3, 4)]
```

['0 1 2 3 4', '1 3 4', '2 4', '3', '4 3']

utool.util_graph.**stack_graphs**(*graph_list*, *vert=False*, *pad=None*)

utool.util_graph.**subgraph_from_edges**(*G*, *edge_list*, *ref_back=True*)

Creates a networkx graph that is a subgraph of G defined by the list of edges in edge_list.

Requires G to be a networkx MultiGraph or MultiDiGraph edge_list is a list of edges in either (u,v) or (u,v,d) form where u and v are nodes comprising an edge, and d would be a dictionary of edge attributes

ref_back determines whether the created subgraph refers to back to the original graph and therefore changes to the subgraph's attributes also affect the original graph, or if it is to create a new copy of the original graph.

References

http://stackoverflow.com/questions/16150557/nx-subgraph-from-edges

utool.util_graph.**testdata_graph**()

Returns (graph, G)

Return type tuple

CommandLine: python -m utool.util_graph –exec-testdata_graph –show

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_graph import *  # NOQA
>>> import utool as ut
>>> import networkx as nx
>>> (graph, G) = testdata_graph()
>>> import wbia.plottool as pt
>>> ut.ensureqt()
>>> pt.show_nx(G, layout='agraph')
>>> ut.show_if_requested()
```

utool.util_graph.**translate_graph**(*graph*, *t_xy*)

utool.util_graph.**translate_graph_to_origin**(*graph*)

utool.util_graph.**traverse_path**(*start*, *end*, *seen_*, *allkeys*, *mat*)

utool.util_graph.**weighted_diamter**(*graph*, *weight=None*)

1.29 utool.util_gridsearch module

module for gridsearch helper

class utool.util_gridsearch.**CountstrParser**(*lhs_dict*, *prop2_nid2_aids*)

Bases: object

Parses a statement like '#primary>0&#primary1>1' and returns a filtered set.

FIXME: make generalizable beyond wbia

compare_op_map = {'!=': <built-in function ne>, '<': <built-in function lt>, '<=':

numop = '#'

parse_countstr_binop(*part*)

parse_countstr_expr(*countstr*)

rrr(*verbose=True*, *reload_module=True*)

special class reloading function This function is often injected as rrr of classes

class utool.util_gridsearch.**DimensionBasis**(*dimension_name*, *dimension_point_list*)

Bases: tuple

dimension_name

Alias for field number 0

dimension_point_list
Alias for field number 1

class utool.util_gridsearch.**GridSearch**(*grid_basis*, *label=None*)
Bases: object

helper for executing iterations and analyzing the results of a grid search

Example

```
>>> # ENABLE_DOCTEST
>>> import utool as ut
>>> grid_basis = [
...     ut.DimensionBasis('p', [.5, .8, .9, 1.0]),
...     ut.DimensionBasis('K', [2, 3, 4, 5]),
...     ut.DimensionBasis('dcvs_clip_max', [.1, .2, .5, 1.0]),
... ]
>>> gridsearch = ut.GridSearch(grid_basis, label='testdata_gridsearch')
>>> for cfgdict in gridsearch:
...     tp_score = cfgdict['p'] + (cfgdict['K'] ** .5)
...     tn_score = (cfgdict['p'] * (cfgdict['K'])) / cfgdict['dcvs_clip_max']
...     gridsearch.append_result(tp_score, tn_score)
```

append_result(*tp_score*, *tn_score*)
for use in iteration

get_csv_results(*max_lines=None*, *score_lbl='score_diff'*)
Make csv text describing results

Parameters

- **max_lines** (*int*) – add top num lines to the csv. No limit if None.
- **score_lbl** (*str*) – score label to sort by

Returns result data in csv format

Return type str

CommandLine: python -m utool.util_gridsearch –test-get_csv_results

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_gridsearch import *  # NOQA
>>> import utool as ut
>>> import wbia.plottool as pt
>>> # build test data
>>> score_lbl = 'score_diff'
>>> gridsearch = testdata_grid_search()
>>> csvtext = gridsearch.get_csv_results(10, score_lbl)
>>> print(csvtext)
>>> result = ut.hashstr(csvtext)
>>> print(result)
60yptleiwo@lk@24
```

get_dimension_stats(*param_lbl*, *score_lbl='score_diff'*)
Returns result stats about a specific parameter

Parameters

- **param_lbl** (*str*) –
- **score_lbl** (*str*) – score label to sort by

Returns param2_score_stats

Return type dict

get_dimension_stats_str(*param_lbl*, *score_lbl='score_diff'*)
Returns a result stat string about a specific parameter

get_param_lbls(*exclude_unvaried_dimension=True*)

get_param_list_and_lbls()
returns input data

get_rank_cfgdict(*rank=0*, *score_lbl='score_diff'*)

get_score_list_and_lbls()
returns result data

get_sorted_columns_and_labels(*score_lbl='score_diff'*)
returns sorted input and result data

plot_dimension(*param_lbl*, *score_lbl='score_diff'*, ***kwargs*)
Plots result statistics about a specific parameter

Parameters

- **param_lbl** (*str*) –
- **score_lbl** (*str*) –

CommandLine: python -m utool.util_gridsearch –test-plot_dimension python -m utool.util_gridsearch –test-plot_dimension –show

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_gridsearch import *  # NOQA
>>> import wbia.plottool as pt
>>> # build test data
>>> gridsearch = testdata_grid_search()
>>> param_lbl = 'p'
>>> score_lbl = 'score_diff'
>>> self = gridsearch
>>> self.plot_dimension('p', score_lbl, fnum=1, pnum=(1, 3, 1))
>>> self.plot_dimension('K', score_lbl, fnum=1, pnum=(1, 3, 2))
>>> self.plot_dimension('dcvs_clip_max', score_lbl, fnum=1, pnum=(1, 3, 3))
>>> pt.show_if_requested()
```

rrr(*verbose=True*, *reload_module=True*)
special class reloading function This function is often injected as rrr of classes

class utool.util_gridsearch.**Nesting**(*type_*, *value*)
Bases: object

x = Nesting.from_nestings(nestings) list(x.itertype('nonNested'))

as_list()

classmethod from_nestings(*nestings*, *type_='root'*)

classmethod from_text(*text*)

itertype(*type_*)

recombine()

class utool.util_gridsearch.**ParamInfo**(*varname=None*, *default=None*, *shortprefix=None*, *type_=NoParam*, *varyvals=[]*, *varyslice=None*, *hideif=NoParam*, *help_=None*, *valid_values=None*, *max_=None*, *min_=None*, *step_=None*, *none_ok=True*)

Bases: *utool.util_dev.NiceRepr*

small class for individual paramater information

Configuration objects should use these for default / printing / type information. NOTE: the actual value of the parameter for any specific configuration is not stored here.

CommandLine: python -m utool.util_gridsearch –test-ParamInfo

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_gridsearch import *  # NOQA
>>> import utool as ut
>>> pi = ParamInfo(varname='foo', default='bar')
>>> cfg = ut.DynStruct()
>>> cfg.foo = 5
>>> result = pi.get_itemstr(cfg)
>>> print(result)
foo=5
```

append_hideif(*hideif*)

cast_to_valid_type(*value*)
Checks if a *value* for this param is valid and attempts to cast it if it is close

error_if_invalid_value(*value*)
Checks if a *value* for this param is valid

get_itemstr(*cfg*)

Parameters **cfg** (*dict-like*) – Parent object (like a dtool.Config)

is_enabled(*cfg*)
alternative to is_hidden. a bit hacky. only returns false if another config hides you. self hiding is ok

is_hidden(*cfg*)
Checks if this param is hidden by the parent context of *cfg*

is_type_enforced()

rrr(*verbose=True*, *reload_module=True*)
special class reloading function This function is often injected as rrr of classes

class utool.util_gridsearch.**ParamInfoBool**(*varname*, *default=False*, *shortprefix=None*, *type_=<class 'bool'>*, *varyvals=[]*, *varyslice=None*, *hideif=False*, *help_=None*)

Bases: *utool.util_gridsearch.ParamInfo*

param info for booleans

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_gridsearch import *  # NOQA
>>> import utool as ut
>>> pi = ParamInfoBool('cheese_on', hideif=util_const.NoParam)
>>> cfg = ut.DynStruct()
>>> cfg.cheese_on = False
>>> result = pi.get_itemstr(cfg)
>>> print(result)
nocheese
```

rrr(*verbose=True*, *reload_module=True*)
special class reloading function This function is often injected as rrr of classes

class utool.util_gridsearch.**ParamInfoList**(*name*, *param_info_list=[]*, *constraint_func=None*, *hideif=None*)

Bases: object

small class for ut.Pref-less configurations

append_hideif(*hideif*)

aslist(*hideif=None*)

get_grid_basis()
DEPRICATE

get_gridsearch_input(*defaultslice=slice(0, 1, None)*)
for gridsearch

get_slicedict()
for gridsearch

get_varnames()

get_varydict()
for gridsearch

rrr(*verbose=True*, *reload_module=True*)
special class reloading function This function is often injected as rrr of classes

updated_cfgdict(*dict_*)

utool.util_gridsearch.**constrain_cfgdict_list**(*cfgdict_list_*, *constraint_func*)
constrains configurations and removes duplicates

utool.util_gridsearch.**customize_base_cfg**(*cfgname*, *cfgopt_strs*, *base_cfg*, *cfgtype*, *alias_keys=None*, *valid_keys=None*, *offset=0*, *strict=True*)

Parameters

- **cfgname** (*str*) – config name
- **cfgopt_strs** (*str*) – mini-language defining key variations
- **base_cfg** (*dict*) – specifies the default cfg to customize
- **cfgtype** –
- **alias_keys** (*None*) – (default = None)
- **valid_keys** (*None*) – if base_cfg is not specied, this defines the valid keys (default = None)

- **offset** (*int*) – (default = 0)
- **strict** (*bool*) – (default = True)

Returns

cfg_combo - list of config dicts defining customized configs based on cfgopt_strs. customized configs always are given an _cfgindex, _cfgstr, and _cfgname key.

Return type list

CommandLine: python -m utool.util_gridsearch –test-customize_base_cfg:0

Ignore:

```
>>> cfgname = 'default'
>>> cfgopt_strs = 'dsize=1000,per_name=[1,2]'
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_gridsearch import *  # NOQA
>>> import utool as ut
>>> cfgname = 'name'
>>> cfgopt_strs = 'b=[1,2]'
>>> base_cfg = {}
>>> alias_keys = None
>>> cfgtype = None
>>> offset = 0
>>> valid_keys = None
>>> strict = False
>>> cfg_combo = customize_base_cfg(cfgname, cfgopt_strs, base_cfg, cfgtype,
>>>                                alias_keys, valid_keys, offset, strict)
>>> result = ('cfg_combo = %s' % (ut.repr2(cfg_combo, nl=1),))
>>> print(result)
cfg_combo = [
    {'_cfgindex': 0, '_cfgname': 'name', '_cfgstr': 'name:b=[1,2]', '_cfgtype':␣
↪None, 'b': 1},
    {'_cfgindex': 1, '_cfgname': 'name', '_cfgstr': 'name:b=[1,2]', '_cfgtype':␣
↪None, 'b': 2},
]
```

utool.util_gridsearch.**get_cfg_lbl**(*cfg, name=None, nonlbl_keys=['_cfgstr', '_cfgname', '_cfgtype', '_cfgindex'], key_order=None, with_name=True, default_cfg=None, sep=''*)

Formats a flat configuration dict into a short string label. This is useful for re-creating command line strings.

Parameters

- **cfg** (*dict*) –
- **name** (*str*) – (default = None)
- **nonlbl_keys** (*list*) – (default = INTERNAL_CFGKEYS)

Returns cfg_lbl

Return type str

CommandLine: python -m utool.util_gridsearch get_cfg_lbl

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_gridsearch import *  # NOQA
>>> import utool as ut
>>> cfg = {'_cfgname': 'test', 'var1': 'val1', 'var2': 'val2'}
>>> name = None
>>> nonlbl_keys = ['_cfgstr', '_cfgname', '_cfgtype', '_cfgindex']
>>> cfg_lbl = get_cfg_lbl(cfg, name, nonlbl_keys)
>>> result = ('cfg_lbl = %s' % (six.text_type(cfg_lbl),))
>>> print(result)
cfg_lbl = test:var1=val1,var2=val2
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_gridsearch import *  # NOQA
>>> import utool as ut
>>> cfg = {'var1': 'val1', 'var2': 'val2'}
>>> default_cfg = {'var2': 'val1', 'var1': 'val1'}
>>> name = None
>>> cfg_lbl = get_cfg_lbl(cfg, name, default_cfg=default_cfg)
>>> result = ('cfg_lbl = %s' % (six.text_type(cfg_lbl),))
>>> print(result)
cfg_lbl = :var2=val2
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_gridsearch import *  # NOQA
>>> import utool as ut
>>> cfg = {'_cfgname': 'test:K=[1,2,3]', 'K': '1'}
>>> name = None
>>> nonlbl_keys = ['_cfgstr', '_cfgname', '_cfgtype', '_cfgindex']
>>> cfg_lbl = get_cfg_lbl(cfg, name, nonlbl_keys)
>>> result = ('cfg_lbl = %s' % (six.text_type(cfg_lbl),))
>>> print(result)
cfg_lbl = test:K=1
```

utool.util_gridsearch.**get_cfgdict_lbl_list_subset**(*cfgdict_list*, *varied_dict*)

utool.util_gridsearch.**get_cfgdict_list_subset**(*cfgdict_list*, *keys*)

returns list of unique dictionaries only with keys specified in keys

Parameters

- **cfgdict_list** (*list*) –
- **keys** (*list*) –

Returns cfglbl_list

Return type list

CommandLine: python -m utool.util_gridsearch –test-get_cfgdict_list_subset

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_gridsearch import *  # NOQA
>>> import utool as ut
>>> # build test data
>>> cfgdict_list = [
...     {'K': 3, 'dcvs_clip_max': 0.1, 'p': 0.1},
...     {'K': 5, 'dcvs_clip_max': 0.1, 'p': 0.1},
...     {'K': 5, 'dcvs_clip_max': 0.1, 'p': 0.2},
...     {'K': 3, 'dcvs_clip_max': 0.2, 'p': 0.1},
...     {'K': 5, 'dcvs_clip_max': 0.2, 'p': 0.1},
...     {'K': 3, 'dcvs_clip_max': 0.2, 'p': 0.1}]
>>> keys = ['K', 'dcvs_clip_max']
>>> # execute function
>>> cfgdict_sublist = get_cfgdict_list_subset(cfgdict_list, keys)
>>> # verify results
>>> result = ut.repr4(cfgdict_sublist)
>>> print(result)
[
    {'K': 3, 'dcvs_clip_max': 0.1},
    {'K': 5, 'dcvs_clip_max': 0.1},
    {'K': 3, 'dcvs_clip_max': 0.2},
    {'K': 5, 'dcvs_clip_max': 0.2},
]
```

utool.util_gridsearch.**get_nonvaried_cfg_lbls**(*cfg_list*, *default_cfg=None*, *mainkey='_cfgname'*)

TODO: this might only need to return a single value. Maybe not if the names are different.

Parameters

- **cfg_list** (*list*) –
- **default_cfg** (*None*) – (default = None)

Returns cfglbl_list

Return type list

CommandLine: python -m utool.util_gridsearch –exec-get_nonvaried_cfg_lbls

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_gridsearch import *  # NOQA
>>> import utool as ut
>>> cfg_list = [{'_cfgname': 'test', 'f': 1, 'b': 1},
>>>             {'_cfgname': 'test', 'f': 2, 'b': 1},
>>>             {'_cfgname': 'test', 'f': 3, 'b': 1, 'z': 4}]
>>> default_cfg = None
>>> cfglbl_list = get_nonvaried_cfg_lbls(cfg_list, default_cfg)
>>> result = ('cfglbl_list = %s' % (ut.repr2(cfglbl_list),))
>>> print(result)
cfglbl_list = ['test:b=1', 'test:b=1', 'test:b=1']
```

utool.util_gridsearch.**get_varied_cfg_lbls**(*cfg_list*, *default_cfg=None*, *mainkey='_cfgname'*, *checkname=False*)

Parameters

- **cfg_list** (*list*) –
- **default_cfg** (*None*) – (default = None)
- **checkname** (*bool*) – if True removes names if they are all the same.

Returns cfglbl_list

Return type list

CommandLine: python -m utool.util_gridsearch –exec-get_varied_cfg_lbls

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_gridsearch import *  # NOQA
>>> import utool as ut
>>> cfg_list = [{'_cfgname': 'test', 'f': 1, 'b': 1},
>>>             {'_cfgname': 'test', 'f': 2, 'b': 1},
>>>             {'_cfgname': 'test', 'f': 3, 'b': 1, 'z': 4}]
>>> default_cfg = None
>>> cfglbl_list = get_varied_cfg_lbls(cfg_list, default_cfg)
>>> result = ('cfglbl_list = %s' % (ut.repr2(cfglbl_list),))
>>> print(result)
cfglbl_list = ['test:f=1', 'test:f=2', 'test:f=3,z=4']
```

utool.util_gridsearch.**grid_search_generator**(*grid_basis=[]*, **args*, ***kwargs*)

Iteratively yeilds individual configuration points inside a defined basis.

Parameters **grid_basis** (*list*) – a list of 2-component tuple. The named tuple looks like this:

CommandLine: python -m utool.util_gridsearch –test-grid_search_generator

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_gridsearch import *  # NOQA
>>> import utool as ut
>>> # build test data
>>> grid_basis = [
...     DimensionBasis('dim1', [.1, .2, .3]),
...     DimensionBasis('dim2', [.1, .4, .5]),
... ]
>>> args = tuple()
>>> kwargs = {}
>>> # execute function
>>> point_list = list(grid_search_generator(grid_basis))
>>> # verify results
>>> column_lbls = ut.get_list_column(grid_basis, 0)
>>> column_list  = ut.get_list_column(grid_basis, 1)
>>> first_vals = ut.get_list_column(ut.get_list_column(grid_basis, 1), 0)
>>> column_types = list(map(type, first_vals))
```

(continues on next page)

(continued from previous page)

```
>>> header = 'grid search'
>>> result = ut.make_csv_table(column_list, column_lbls, header, column_types)
>>> print(result)
grid search
# num_rows=3
#   dim1,  dim2
    0.10,  0.10
    0.20,  0.40
    0.30,  0.50
```

utool.util_gridsearch.**gridsearch_timer**(*func_list*, *args_list*, *niters=None*, ***searchkw*)

Times a series of functions on a series of inputs

args_list is a list should vary the input sizes can also be a func that take a count param

items in args_list list or returned by the func should be a tuple so it can be unpacked

CommandLine: python -m wbia.annotmatch_funcs –exec-get_annotmatch_rowids_from_aid2 –show python -m wbia.annotmatch_funcs –exec-get_annotmatch_rowids_from_aid:1 –show

Parameters

- **func_list** (*list*) –
- **args_list** (*list*) –
- **niters** (*None*) – (default = None)

Returns time_result

Return type dict

CommandLine: python -m utool.util_gridsearch –exec-gridsearch_timer –show

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_gridsearch import *  # NOQA
>>> import utool as ut
>>> func_list = [ut.fibonacci_recursive, ut.fibonacci_iterative]
>>> args_list = list(range(1, 35))
>>> niters = None
>>> searchkw = {}
>>> time_result = gridsearch_timer(func_list, args_list, niters, **searchkw)
>>> result = ('time_result = %s' % (six.text_type(time_result),))
>>> print(result)
>>> time_result['plot_timings']()
>>> ut.show_if_requested()
```

utool.util_gridsearch.**interact_gridsearch_result_images**(*show_result_func*, *cfgdict_list*, *cfglbl_list*, *cfgresult_list*, *score_list=None*, *fnum=None*, *figtitle=''*, *unpack=False*, *max_plots=25*, *verbose=True*, *precision=3*, *scorelbl='score'*, *onclick_func=None*)

helper function for visualizing results of gridsearch

utool.util_gridsearch.**lookup_base_cfg_list**(*cfgname*, *named_defaults_dict*, *meta-data=None*)

utool.util_gridsearch.**make_cfglbls**(*cfgdict_list*, *varied_dict*)
Show only the text in labels that mater from the cfgdict

utool.util_gridsearch.**make_constrained_cfg_and_lbl_list**(*varied_dict*, *constraint_func=None*, *slice_dict=None*, *defaultslice=slice(0, 1, None)*)

Parameters

- **varied_dict** (*dict*) – parameters to vary with possible variations
- **constraint_func** (*func*) – function to restirct parameter variations
- **slice_dict** (*dict*) – dict of slices for each param of valid possible values
- **defaultslice** (*slice*) – default slice used if slice is not specified in slice_dict

Returns (cfgdict_list, cfglbl_list)

Return type tuple

CommandLine: python -m utool.util_gridsearch –test-make_constrained_cfg_and_lbl_list

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_gridsearch import *  # NOQA
>>> # build test data
>>> varied_dict = {
...     'p': [.1, .3, 1.0, 2.0],
...     'dcvs_clip_max': [.1, .2, .5],
...     'K': [3, 5],
... }
>>> constraint_func = None
>>> # execute function
>>> (cfgdict_list, cfglbl_list) = make_constrained_cfg_and_lbl_list(varied_dict,␣
↪constraint_func)
>>> # verify results
>>> result = six.text_type((cfgdict_list, cfglbl_list))
>>> print(result)
```

utool.util_gridsearch.**noexpand_parse_cfgstrs**(*cfgopt_strs*, *alias_keys=None*)
alias_keys = None cfgopt_strs = ‘f=2,c=[(1,2),(3,4)],d=1’ string = cfgopt_strs

cfgopt_strs = ‘f=2,c=[1,2,3,4]’

CommandLine: python -m utool.util_gridsearch noexpand_parse_cfgstrs

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_gridsearch import *  # NOQA
>>> import utool as ut
>>> cfgopt_strs = 'b=[1,2]'
>>> alias_keys = None
>>> cfg_combo = noexpand_parse_cfgstrs(cfgopt_strs, alias_keys)
>>> result = ('cfg_combo = %s' % (ut.repr2(cfg_combo, nl=0),))
>>> print(result)
cfg_combo = {'b': [1, 2]}
```

utool.util_gridsearch.**parse_argv_cfg**(*argname, default=[''], named_defaults_dict=None, valid_keys=None, alias_keys=None*)

simple configs

Parameters

- **argname** (*str*) –
- **default** (*list*) – (default = [])
- **named_defaults_dict** (*dict*) – (default = None)
- **valid_keys** (*None*) – (default = None)

Returns cfg_list

Return type list

CommandLine: python -m utool.util_gridsearch –exec-parse_argv_cfg –filt :foo=bar python -m utool.util_gridsearch –exec-parse_argv_cfg

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_gridsearch import *  # NOQA
>>> import utool as ut
>>> argname = '--filt'
>>> cfg_list = parse_argv_cfg(argname)
>>> result = ('cfg_list = %s' % (six.text_type(cfg_list),))
>>> print(result)
```

utool.util_gridsearch.**parse_cfgstr3**(*string*, *debug=None*)

http://stackoverflow.com/questions/4801403/how-can-i-use-pyparsing-to-parse-nested-expressions-that-have-mutiple-opener-clo

Ignore:

```
>>> from utool.util_gridsearch import *  # NOQA
cfgopt_strs = 'f=2,c=[(1,2),(3,4)],d=1,j=[[1,2],[3,4]],foobar,x="[fdsfds",y=
↪"]fdsfds",e=[[1,2],[3,4]],[]'
string  =  cfgopt_strs
parse_cfgstr3(string)
```

CommandLine: python -m utool.util_gridsearch parse_cfgstr3 –show

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_gridsearch import *  # NOQA
>>> import utool as ut
>>> cfgopt_strs = 'b=[1,2]'
>>> cfgdict = parse_cfgstr3(cfgopt_strs)
>>> result = ('cfgdict = %s' % (ut.repr2(cfgdict),))
>>> print(result)
cfgdict = {'b': [1, 2]}
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_gridsearch import *  # NOQA
>>> import utool as ut
>>> cfgopt_strs = 'myprefix=False,sentence_break=False'
>>> cfgdict = parse_cfgstr3(cfgopt_strs, debug=True)
>>> print('cfgopt_strs = %r' % (cfgopt_strs,))
>>> result = ('cfgdict = %s' % (ut.repr2(cfgdict),))
>>> print(result)
cfgdict = {'myprefix': False, 'sentence_break': False}
```

utool.util_gridsearch.**parse_cfgstr_list2**(*cfgstr_list*, *named_defaults_dict=None*, *cfgtype=None*, *alias_keys=None*, *valid_keys=None*, *expand_nested=True*, *strict=True*, *special_join_dict=None*, *is_nestedcfgtype=False*, *metadata=None*)

Parses config strings. By looking up name in a dict of configs

Parameters

- **cfgstr_list** (*list*) –
- **named_defaults_dict** (*dict*) – (default = None)
- **cfgtype** (*None*) – (default = None)
- **alias_keys** (*None*) – (default = None)
- **valid_keys** (*None*) – (default = None)
- **expand_nested** (*bool*) – (default = True)
- **strict** (*bool*) – (default = True)
- **- used for annot configs so special joins arent geometrically combined** (*is_nestedcfgtype*) –

Note:

Normal Case: –flag name

Custom Arugment Cases: –flag name:custom_key1=custom_val1,custom_key2=custom_val2

Multiple Config Case: –flag name1:custom_args1 name2:custom_args2

Multiple Config (special join) Case: (here name2 and name3 have some special interaction) –flag name1:custom_args1 name2:custom_args2::name3:custom_args3

Varied Argument Case: –flag name:key1=[val1,val2]

Returns cfg_combos_list

Return type list

CommandLine: python -m utool.util_gridsearch –test-parse_cfgstr_list2 python -m utool.util_gridsearch –test-parse_cfgstr_list2:0 python -m utool.util_gridsearch –test-parse_cfgstr_list2:1 python -m utool.util_gridsearch –test-parse_cfgstr_list2:2

Setup:

```
>>> from utool.util_gridsearch import *  # NOQA
>>> import utool as ut
>>> named_defaults_dict = None
>>> cfgtype, alias_keys, valid_keys, metadata = None, None, None, None
>>> expand_nested, is_nestedcfgtypel, strict = True, False, False
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_gridsearch import *  # NOQA
>>> import utool as ut
>>> named_defaults_dict = None
>>> cfgtype, alias_keys, valid_keys, metadata = None, None, None, None
>>> expand_nested, is_nestedcfgtypel, strict = True, False, False
>>> cfgstr_list = ['name', 'name:f=1', 'name:b=[1,2]', 'name1:f=1::name2:f=1,b=2']
>>> #cfgstr_list = ['name', 'name1:f=1::name2:f=1,b=2']
>>> special_join_dict = {'joined': True}
>>> cfg_combos_list = parse_cfgstr_list2(
>>>     cfgstr_list, named_defaults_dict, cfgtype, alias_keys, valid_keys,
>>>     expand_nested, strict, special_join_dict)
>>> print('b' in cfg_combos_list[2][0])
>>> print('cfg_combos_list = %s' % (ut.repr4(cfg_combos_list, nl=2),))
>>> assert 'b' in cfg_combos_list[2][0], 'second cfg[2] should vary b'
>>> assert 'b' in cfg_combos_list[2][1], 'second cfg[2] should vary b'
>>> print(ut.depth_profile(cfg_combos_list))
>>> cfg_list = ut.flatten(cfg_combos_list)
>>> cfg_list = ut.flatten([cfg if isinstance(cfg, list) else [cfg] for cfg in cfg_
↪list])
>>> result = ut.repr2(ut.get_varied_cfg_lbls(cfg_list))
>>> print(result)
['name:', 'name:f=1', 'name:b=1', 'name:b=2', 'name1:f=1,joined=True', 'name2:b=2,
↪f=1,joined=True']
```

Example

```
>>> # ENABLE_DOCTEST
>>> # Allow for definition of a named default on the fly
>>> from utool.util_gridsearch import *  # NOQA
>>> import utool as ut
>>> named_defaults_dict = None
>>> cfgtype, alias_keys, valid_keys, metadata = None, None, None, None
```

(continues on next page)

(continued from previous page)

```
>>> expand_nested, is_nestedcfgtypel, strict = True, False, False
>>> cfgstr_list = ['base=:f=2,c=[1,2]', 'base:f=1', 'base:b=[1,2]']
>>> special_join_dict = None
>>> cfg_combos_list = parse_cfgstr_list2(
>>>     cfgstr_list, named_defaults_dict, cfgtype, alias_keys, valid_keys,
>>>     expand_nested, strict, special_join_dict)
>>> print('cfg_combos_list = %s' % (ut.repr4(cfg_combos_list, nl=2),))
>>> print(ut.depth_profile(cfg_combos_list))
>>> cfg_list = ut.flatten(cfg_combos_list)
>>> cfg_list = ut.flatten([cfg if isinstance(cfg, list) else [cfg] for cfg in cfg_
↪list])
>>> result = ut.repr2(ut.get_varied_cfg_lbls(cfg_list))
>>> print(result)
['base:c=1,f=1', 'base:c=2,f=1', 'base:b=1,c=1,f=2', 'base:b=1,c=2,f=2',
↪'base:b=2,c=1,f=2', 'base:b=2,c=2,f=2']
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_gridsearch import *  # NOQA
>>> import utool as ut
>>> named_defaults_dict = None
>>> cfgtype, alias_keys, valid_keys, metadata = None, None, None, None
>>> expand_nested, is_nestedcfgtypel, strict = True, False, False
>>> cfgstr_list = ['base:f=2,c=[(1,2),(3,4)]']
>>> special_join_dict = None
>>> cfg_combos_list = parse_cfgstr_list2(
>>>     cfgstr_list, named_defaults_dict, cfgtype, alias_keys, valid_keys,
>>>     expand_nested, strict, special_join_dict)
>>> print('cfg_combos_list = %s' % (ut.repr4(cfg_combos_list, nl=2),))
>>> print(ut.depth_profile(cfg_combos_list))
>>> cfg_list = ut.flatten(cfg_combos_list)
>>> cfg_list = ut.flatten([cfg if isinstance(cfg, list) else [cfg] for cfg in cfg_
↪list])
>>> result = ut.repr2(ut.get_varied_cfg_lbls(cfg_list))
>>> print(result)
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_gridsearch import *  # NOQA
>>> import utool as ut
>>> named_defaults_dict = None
>>> cfgtype, alias_keys, valid_keys, metadata = None, None, None, None
>>> expand_nested, is_nestedcfgtypel, strict = True, False, False
>>> # test simplest case
>>> cfgstr_list = ['name:b=[1,2]']
>>> special_join_dict = {'joined': True}
>>> cfg_combos_list = parse_cfgstr_list2(
>>>     cfgstr_list, named_defaults_dict, cfgtype, alias_keys, valid_keys,
>>>     expand_nested, strict, special_join_dict)
>>> print('b' in cfg_combos_list[0][0])
>>> print('cfg_combos_list = %s' % (ut.repr4(cfg_combos_list, nl=2),))
```

(continues on next page)

(continued from previous page)

```
>>> assert 'b' in cfg_combos_list[0][0], 'second cfg[2] should vary b'
>>> assert 'b' in cfg_combos_list[0][1], 'second cfg[2] should vary b'
>>> print(ut.depth_profile(cfg_combos_list))
>>> cfg_list = ut.flatten(cfg_combos_list)
>>> cfg_list = ut.flatten([cfg if isinstance(cfg, list) else [cfg] for cfg in cfg_
↪list])
>>> result = ut.repr2(ut.get_varied_cfg_lbls(cfg_list))
>>> print(result)
```

utool.util_gridsearch.**parse_cfgstr_name_options**(*cfgstr*)

Parameters **cfgstr** (*str*) –

Returns (cfgname, cfgopt_strs, subx)

Return type tuple

CommandLine: python -m utool.util_gridsearch –test-parse_cfgstr_name_options

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_gridsearch import *  # NOQA
>>> import utool as ut
>>> cfgstr = 'default' + NAMEVARSEP + 'myvar1=myval1,myvar2=myval2'
>>> (cfgname, cfgopt_strs, subx) = parse_cfgstr_name_options(cfgstr)
>>> result = ('(cfgname, cfg_optstrs, subx) = %s' % (ut.repr2((cfgname, cfgopt_
↪strs, subx)),))
>>> print(result)
(cfgname, cfg_optstrs, subx) = ('default', 'myvar1=myval1,myvar2=myval2', None)
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_gridsearch import *  # NOQA
>>> import utool as ut
>>> cfgstr = 'default[0:1]' + NAMEVARSEP + 'myvar1=myval1,myvar2=myval2'
>>> (cfgname, cfgopt_strs, subx) = parse_cfgstr_name_options(cfgstr)
>>> result = ('(cfgname, cfg_optstrs, subx) = %s' % (ut.repr2((cfgname, cfgopt_
↪strs, subx)),))
>>> print(result)
(cfgname, cfg_optstrs, subx) = ('default', 'myvar1=myval1,myvar2=myval2', slice(0,
↪ 1, None))
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_gridsearch import *  # NOQA
>>> import utool as ut
>>> cfgstr = 'default[0]' + NAMEVARSEP + 'myvar1=myval1,myvar2=myval2'
>>> (cfgname, cfgopt_strs, subx) = parse_cfgstr_name_options(cfgstr)
>>> result = ('(cfgname, cfg_optstrs, subx) = %s' % (ut.repr2((cfgname, cfgopt_
↪strs, subx)),))
```

(continues on next page)

(continued from previous page)

```
>>> print(result)
(cfgname, cfg_optstrs, subx) = ('default', 'myvar1=myval1,myvar2=myval2', [0])
```

utool.util_gridsearch.**parse_nestings**(*string*, *only_curl=False*)

References

http://stackoverflow.com/questions/4801403/pyparsing-nested-mutiple-opener-clo

CommandLine: python -m utool.util_gridsearch parse_nestings:1 –show

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_gridsearch import *  # NOQA
>>> import utool as ut
>>> string = r'lambda u: sign(u) * abs(u)**3.0 * greater(u, 0)'
>>> parsed_blocks = parse_nestings(string)
>>> recombined = recombine_nestings(parsed_blocks)
>>> print('PARSED_BLOCKS = ' + ut.repr3(parsed_blocks, nl=1))
>>> print('recombined = %r' % (recombined,))
>>> print('orig       = %r' % (string,))
PARSED_BLOCKS = [
    ('nonNested', 'lambda u: sign'),
    ('paren', [('ITEM', '('), ('nonNested', 'u'), ('ITEM', ')')]),
    ('nonNested', '* abs'),
    ('paren', [('ITEM', '('), ('nonNested', 'u'), ('ITEM', ')')]),
    ('nonNested', '**3.0 * greater'),
    ('paren', [('ITEM', '('), ('nonNested', 'u, 0'), ('ITEM', ')')]),
]
```

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_gridsearch import *  # NOQA
>>> import utool as ut
>>> string = r'\chapter{Identification \textbf{foobar} workflow}\label
↪{chap:application}'
>>> parsed_blocks = parse_nestings(string)
>>> print('PARSED_BLOCKS = ' + ut.repr3(parsed_blocks, nl=1))
PARSED_BLOCKS = [
    ('nonNested', '\\chapter'),
    ('curl', [('ITEM', '{'), ('nonNested', 'Identification \\textbf'), ('curl', [(
↪'ITEM', '{'), ('nonNested', 'foobar'), ('ITEM', '}')]), ('nonNested', 'workflow
↪'), ('ITEM', '}')]),
    ('nonNested', '\\label'),
    ('curl', [('ITEM', '{'), ('nonNested', 'chap:application'), ('ITEM', '}')]),
]
```

utool.util_gridsearch.**parse_nestings2**(*string*, *nesters=['()', '[]', '<>', '"', '"']*, *escape='\\'*)

References

http://stackoverflow.com/questions/4801403/pyparsing-nested-mutiple-opener-clo

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_gridsearch import *  # NOQA
>>> import utool as ut
>>> string = r'lambda u: sign(u) * abs(u)**3.0 * greater(u, 0)'
>>> parsed_blocks = parse_nestings2(string)
>>> print('parsed_blocks = {!r}'.format(parsed_blocks))
>>> string = r'lambda u: sign("\"u(\'fdfds\')") * abs(u)**3.0 * greater(u, 0)'
>>> parsed_blocks = parse_nestings2(string)
>>> print('parsed_blocks = {!r}'.format(parsed_blocks))
>>> recombined = recombine_nestings(parsed_blocks)
>>> print('PARSED_BLOCKS = ' + ut.repr3(parsed_blocks, nl=1))
>>> print('recombined = %r' % (recombined,))
>>> print('orig       = %r' % (string,))
```

utool.util_gridsearch.**partition_varied_cfg_list**(*cfg_list*, *default_cfg=None*, *recursive=False*)

Separates varied from non-varied parameters in a list of configs

TODO: partition nested configs

CommandLine: python -m utool.util_gridsearch –exec-partition_varied_cfg_list:0

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_gridsearch import *  # NOQA
>>> import utool as ut
>>> cfg_list = [{'f': 1, 'b': 1}, {'f': 2, 'b': 1}, {'f': 3, 'b': 1, 'z': 4}]
>>> nonvaried_cfg, varied_cfg_list = partition_varied_cfg_list(cfg_list)
>>> result = ut.repr4({'nonvaried_cfg': nonvaried_cfg,
>>>                    'varied_cfg_list': varied_cfg_list}, explicit=1, nobr=True,
↪ nl=1)
>>> print(result)
nonvaried_cfg={'b': 1},
varied_cfg_list=[{'f': 1}, {'f': 2}, {'f': 3, 'z': 4}],
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_gridsearch import *  # NOQA
>>> import utool as ut
>>> cfg_list = [{'q1': 1, 'f1': {'a2': {'x3': 1, 'y3': 2}, 'b2': 1}}, {'q1': 1,
↪'f1': {'a2': {'x3': 1, 'y3': 1}, 'b2': 1}, 'e1': 1}]
>>> print(ut.repr4(cfg_list, nl=1))
>>> nonvaried_cfg, varied_cfg_list = partition_varied_cfg_list(cfg_list,␣
↪recursive=True)
>>> result = ut.repr4({'nonvaried_cfg': nonvaried_cfg,
>>>                    'varied_cfg_list': varied_cfg_list}, explicit=1, nobr=True,
↪ nl=1)
```

(continues on next page)

(continued from previous page)

```
>>> print(result)
nonvaried_cfg={'f1': {'a2': {'x3': 1}, 'b2': 1}, 'q1': 1},
varied_cfg_list=[{'f1': {'a2': {'y3': 2}}}, {'e1': 1, 'f1': {'a2': {'y3': 1}}}],
```

utool.util_gridsearch.**recombine_nestings**(*parsed_blocks*)

utool.util_gridsearch.**testdata_grid_search**()
test data function for doctests

1.30 utool.util_hash module

Hashing convinience functions

You should opt to use a hash*27 function over a hash* function.

TODO: the same hashing algorithm should be used everywhere Currently there is a mix of sha1, sha256, and sha512 in different places.

utool.util_hash.**augment_uuid**(*uuid_*, **hashables*)

utool.util_hash.**b**(*x*)

utool.util_hash.**combine_hashes**(*bytes_list*, *hasher=None*)
Only works on bytes

Example

```
>>> # DISABLE_DOCTEST
>>> x = [b('1111'), b('2222')]
>>> y = [b('11'), b('11'), b('22'), b('22')]
>>> bytes_list = y
>>> out1 = ut.combine_hashes(x, hashlib.sha1())
>>> hasher = hashlib.sha1()
>>> out2 = ut.combine_hashes(y, hasher)
>>> bytes_ = out2
>>> assert hasher.hexdigest() == freeze_hash_bytes(hasher.digest())
>>> assert convert_bytes_to_bigbase(hasher.digest()) == convert_hexstr_to_
↪bigbase(hasher.hexdigest())
>>> assert out1 != out2
>>> print('out1 = %r' % (out1,))
>>> print('out2 = %r' % (out2,))
```

utool.util_hash.**combine_uuids**(*uuids*, *ordered=True*, *salt=''*)
Creates a uuid that specifies a group of UUIDS

Parameters

- **uuids** (*list*) – list of uuid objects
- **ordered** (*bool*) – if False uuid order changes the resulting combined uuid otherwise the uuids are considered an orderless set
- **salt** (*str*) – salts the resulting hash

Returns combined uuid

Return type uuid.UUID

CommandLine: python -m utool.util_hash –test-combine_uuids

Ignore:

```
>>> # ENABLE_DOCTEST
>>> from utool.util_hash import *  # NOQA
>>> import utool as ut
>>> uuids = [hashable_to_uuid('one'), hashable_to_uuid('two'),
>>>          hashable_to_uuid('three')]
>>> combo1 = combine_uuids(uuids, ordered=True)
>>> combo2 = combine_uuids(uuids[::-1], ordered=True)
>>> combo3 = combine_uuids(uuids, ordered=False)
>>> combo4 = combine_uuids(uuids[::-1], ordered=False)
>>> result = ut.repr4([combo1, combo2, combo3, combo4], nobr=True)
>>> print(result)
UUID('83ee781f-8646-ccba-0ed8-13842825c12a'),
UUID('52bbb33f-612e-2ab8-a62c-2f46e5b1edc8'),
UUID('945cadab-e834-e581-0f74-62f106d20d81'),
UUID('945cadab-e834-e581-0f74-62f106d20d81'),
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_hash import *  # NOQA
>>> import utool as ut
>>> uuids = [uuid.UUID('5ff6b34e-7d8f-ef32-5fad-489266acd2ae'),
>>>          uuid.UUID('f2400146-ec12-950b-1489-668228e155a8'),
>>>          uuid.UUID('037d6f31-8c73-f961-1fe4-d616442a1e86'),
>>>          uuid.UUID('ca45d6e2-e648-09cc-a49e-e71c6fa3b3f3')]
>>> ordered = True
>>> salt = u''
>>> result = combine_uuids(uuids, ordered, salt)
>>> print(result)
1dabc66b-b564-676a-99b4-5cae7a9e7294
```

utool.util_hash.**convert_bytes_to_bigbase**(*bytes_*, *alphabet=['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']*)

Parameters **bytes** (*bytes*) –

Returns

Return type str

Ignore:

CommandLine: python -m utool.util_hash convert_bytes_to_bigbase

Ignore:

```
>>> # DISABLE_DOCTEST
>>> from utool.util_hash import *  # NOQA
>>> import utool as ut
>>> bytes_ = b('99999999999999999999999999999999')
>>> alphabet = ALPHABET_27
>>> result = convert_bytes_to_bigbase(bytes_, alphabet)
>>> print(result)
fervudwhpustklnptklklcgswbmvtustqocdpgiwkgrvwytvneardkpytd
```

utool.util_hash.**convert_hexstr_to_bigbase**(*hexstr, alphabet=['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z'], bigbase=26*)

Packs a long hexstr into a shorter length string with a larger base

Ignore:

```
>>> # Determine the length savings with lossless conversion
>>> import sympy as sy
>>> consts = dict(hexbase=16, hexlen=256, bigbase=27)
>>> symbols = sy.symbols('hexbase, hexlen, bigbase, newlen')
>>> haexbase, hexlen, bigbase, newlen = symbols
>>> eqn = sy.Eq(16 ** hexlen,  bigbase ** newlen)
>>> newlen_ans = sy.solve(eqn, newlen)[0].subs(consts).evalf()
>>> print('newlen_ans = %r' % (newlen_ans,))
>>>
>>> # for a 27 char alphabet we can get 216
>>> print('Required length for lossless conversion len2 = %r' % (len2,))
>>>
>>> def info(base, len):
>>>     bits = base ** len
>>>     print('base = %r' % (base,))
>>>     print('len = %r' % (len,))
>>>     print('bits = %r' % (bits,))
>>> info(16, 256)
>>> info(27, 16)
>>> info(27, 64)
>>> info(27, 216)
```

utool.util_hash.**digest_data**(*data*, *alg='sha256'*)

utool.util_hash.**freeze_hash_bytes**(*bytes_*)

utool.util_hash.**get_file_hash**(*fpath*, *blocksize=65536*, *hasher=None*, *stride=1*, *hexdigest=False*)

For better hashes use hasher=hashlib.sha256, and keep stride=1

Parameters

- **fpath** (*str*) – file path string
- **blocksize** (*int*) – 2 ** 16. Affects speed of reading file
- **hasher** (*None*) – defaults to sha1 for fast (but insecure) hashing
- **stride** (*int*) – strides > 1 skip data to hash, useful for faster hashing, but less accurate, also makes hash dependant on blocksize.

References

http://stackoverflow.com/questions/3431825/generating-a-md5-checksum-of-a-file http://stackoverflow.com/questions/5001893/when-should-i-use-sha-1-and-when-should-i-use-sha-2

CommandLine: python -m utool.util_hash –test-get_file_hash python -m utool.util_hash –test-get_file_hash:0 python -m utool.util_hash –test-get_file_hash:1

Ignore:

```
>>> # DISABLE_DOCTEST
>>> from utool.util_hash import *  # NOQA
```

(continues on next page)

(continued from previous page)

```
>>> fpath = ut.grab_test_imgpath('patsy.jpg')
>>> #blocksize = 65536  # 2 ** 16
>>> blocksize = 2 ** 16
>>> hasher = None
>>> stride = 1
>>> hashbytes_20 = get_file_hash(fpath, blocksize, hasher, stride)
>>> result = repr(hashbytes_20)
>>> print(result)
'7\x07B\x0eX<sRu\xa2\x90P\xda\xb2\x84?\x81?\xa9\xd9'
```

‘x13x9bxf6x0fxa3QQ xd7”$xe9mx05x9ex81xf6xf2vxe4’

‘x16x00x80Xxx8c-HxcdPxf6x02x9frlxbfx99VQxb5’

Ignore:

```
>>> # DISABLE_DOCTEST
>>> from utool.util_hash import *  # NOQA
>>> #fpath = ut.grab_file_url('http://en.wikipedia.org/wiki/List_of_comets_by_
↪type')
>>> fpath = ut.unixjoin(ut.ensure_app_resource_dir('utool'), 'tmp.txt')
>>> ut.write_to(fpath, ut.lorium_ipsum())
>>> blocksize = 2 ** 3
>>> hasher = None
>>> stride = 2
>>> hashbytes_20 = get_file_hash(fpath, blocksize, hasher, stride)
>>> result = repr(hashbytes_20)
>>> print(result)
'5KP\xcf>R\xf6\xffO:L\xac\x9c\xd3V+\x0e\xf6\xe1n'
```

Ignore: file_ = open(fpath, ‘rb’)

utool.util_hash.**get_file_uuid**(*fpath*, *hasher=None*, *stride=1*)
Creates a uuid from the hash of a file

utool.util_hash.**get_zero_uuid**()

utool.util_hash.**hash_data**(*data*, *hashlen=None*, *alphabet=None*)
Get a unique hash depending on the state of the data.

Parameters

- **data** (*object*) – any sort of loosely organized data
- **hashlen** (*None*) – (default = None)
- **alphabet** (*None*) – (default = None)

Returns text - hash string

Return type str

CommandLine: python -m utool.util_hash hash_data

Ignore:

```
>>> # ENABLE_DOCTEST
>>> from utool.util_hash import *  # NOQA
>>> import utool as ut
>>> counter = [0]
```

(continues on next page)

(continued from previous page)

```
>>> failed = []
>>> def check_hash(input_, want=None):
>>>     count = counter[0] = counter[0] + 1
>>>     got = ut.hash_data(input_)
>>>     print('({}) {}'.format(count, got))
>>>     if want is not None and not got.startswith(want):
>>>         failed.append((got, input_, count, want))
>>> check_hash('1', 'wuvrng')
>>> check_hash(['1'], 'dekbfpby')
>>> check_hash(tuple(['1']), 'dekbfpby')
>>> check_hash(b'12', 'marreflbv')
>>> check_hash([b'1', b'2'], 'nwfs')
>>> check_hash(['1', '2', '3'], 'arfrp')
>>> check_hash(['1', np.array([1,2,3]), '3'], 'uyqwcq')
>>> check_hash('123', 'ehkgxk')
>>> check_hash(zip([1, 2, 3], [4, 5, 6]), 'mjcpwa')
>>> import numpy as np
>>> rng = np.random.RandomState(0)
>>> check_hash(rng.rand(100000), 'bdwosuey')
>>> for got, input_, count, want in failed:
>>>     print('failed {} on {}'.format(count, input_))
>>>     print('got={}, want={}'.format(got, want))
>>> assert not failed
```

utool.util_hash.**hashable_to_uuid**(*hashable_*)

TODO: ensure that python2 and python3 agree on hashes of the same information

Parameters **hashable** (*hashable*) – hashables are bytes-like objects An object that supports the Buffer Protocol, like bytes, bytearray or memoryview. Bytes-like objects can be used for various operations that expect binary data, such as compression, saving to a binary file or sending over a socket. Some operations need the binary data to be mutable, in which case not all bytes-like objects can apply.

Returns UUID

CommandLine: python -m utool.util_hash –test-hashable_to_uuid

Ignore:

```
>>> # ENABLE_DOCTEST
>>> from utool.util_hash import *  # NOQA
>>> import utool as ut
>>> hashables = [
>>>     'foobar',
>>>     'foobar'.encode('utf-8'),
>>>     u'foobar',
>>>     10,
>>>     [1, 2, 3],
>>> ]
>>> uuids = []
>>> for hashable_ in hashables:
>>>     uuid_ = hashable_to_uuid(hashable_)
>>>     uuids.append(uuid_)
>>> result = ut.repr4(ut.lmap(str, uuids), strvals=True, nobr=True)
>>> print(result)
8843d7f9-2416-211d-e9eb-b963ff4ce281,
```

(continues on next page)

(continued from previous page)

```
8843d7f9-2416-211d-e9eb-b963ff4ce281,
8843d7f9-2416-211d-e9eb-b963ff4ce281,
e864ece8-8880-43b6-7277-c8b2cefe96ad,
a01eda32-e4e0-b139-3274-e91d1b3e9ecf,
```

utool.util_hash.**hashid_arr**(*arr*, *label='arr'*, *hashlen=16*)
newer version of hashstr_arr2

utool.util_hash.**hashstr**(*data, hashlen=16, alphabet=['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']*)
python -c "import utool as ut; print(ut.hashstr('abcd'))"

Parameters

- **data** (*hashable*) –
- **hashlen** (*int*) – (default = 16)
- **alphabet** (*list*) – list of characters:

Returns hashstr

Return type str

CommandLine: python -m utool.util_hash –test-hashstr python3 -m utool.util_hash –test-hashstr python3 -m utool.util_hash –test-hashstr:2 python -m utool.util_hash hashstr:3 python3 -m utool.util_hash hashstr:3

Ignore:

```
>>> # ENABLE_DOCTEST
>>> from utool.util_hash import *  # NOQA
>>> data = 'foobar'
>>> hashlen = 16
>>> alphabet = ALPHABET_41
>>> text = hashstr(data, hashlen, alphabet)
>>> result = ('text = %s' % (str(text),))
>>> print(result)
text = mi5yum60mbxhyp+x
```

Ignore:

```
>>> # ENABLE_DOCTEST
>>> from utool.util_hash import *  # NOQA
>>> data = ''
>>> hashlen = 16
>>> alphabet = ALPHABET_41
>>> text = hashstr(data, hashlen, alphabet)
>>> result = ('text = %s' % (str(text),))
>>> print(result)
text = 0000000000000000
```

Ignore:

```
>>> # ENABLE_DOCTEST
>>> from utool.util_hash import *  # NOQA
>>> import numpy as np
>>> data = np.array([1, 2, 3])
>>> hashlen = 16
```

(continues on next page)

(continued from previous page)

```
>>> alphabet = ALPHABET_41
>>> text = hashstr(data, hashlen, alphabet)
>>> result = ('text = %s' % (str(text),))
>>> print(result)
text = z5lqw0bzt4dmb9yy
```

Ignore:

```
>>> # ENABLE_DOCTEST
>>> from utool.util_hash import *  # NOQA
>>> import numpy as np
>>> from uuid import UUID
>>> data = (UUID('7cd0197b-1394-9d16-b1eb-0d8d7a60aedc'), UUID('c76b54a5-adb6-
↪7f16-f0fb-190ab99409f8'))
>>> hashlen = 16
>>> alphabet = ALPHABET_41
>>> text = hashstr_arr(data, 'label')
>>> result = ('text = %s' % (str(text),))
>>> print(result)
```

Ignore:

```
>>> # DISABLE_DOCTEST
>>> # UNSTABLE_DOCTEST
>>> from utool.util_hash import *  # NOQA
>>> import numpy as np
>>> data = np.array(['a', 'b'], dtype=object)
>>> text = hashstr(data, alphabet=ALPHABET_27)
>>> result = ('text = %s' % (str(text),))
>>> print(result)
```

Ignore: data = np.array([‘a’, ‘b’], dtype=object) data.tobytes() data = np.array([‘a’, ‘b’]) data = [‘a’, ‘b’] data = np.array([1, 2, 3]) import hashlib from six.moves import cPickle as pickle pickle.dumps(data, protocol=2)

python2 -c “import hashlib, numpy; print(hashlib.sha1(‘ab’).hexdigest())” python3 -c “import hashlib, numpy; print(hashlib.sha1(‘ab’.encode(‘utf8’)).hexdigest())”

python2 -c “import hashlib, numpy; print(hashlib.sha1(‘ab’).hexdigest())” python3 -c “import hashlib, numpy; print(hashlib.sha1(b’ab’).hexdigest())”

python2 -c “import hashlib, numpy; print(hashlib.sha1(numpy.array([1, 2])).hexdigest())” python3 -c “import hashlib, numpy; print(hashlib.sha1(numpy.array([1, 2])).hexdigest())”

TODO: numpy arrays of strings must be encoded to bytes first in python3 python2 -c “import hashlib, numpy; print(hashlib.sha1(numpy.array([‘a’, ‘b’])).hexdigest())” python3 -c “import hashlib, numpy; print(hashlib.sha1(numpy.array([b’a’, b’b’])).hexdigest())”

python -c “import hashlib, numpy; print(hashlib.sha1(numpy.array([‘a’, ‘b’], dtype=object)).hexdigest())” python -c “import hashlib, numpy; print(hashlib.sha1(numpy.array([‘a’, ‘b’], dtype=object)).hexdigest())”

utool.util_hash.**hashstr27**(*data*, *alphabet=['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']*, ***kwargs*)

utool.util_hash.**hashstr_arr**(*arr*, *lbl='arr'*, *pathsafe=False*, ***kwargs*)

Parameters

- **arr** (*ndarray*) –
- **lbl** (*str*) – (default = ‘arr’)

- **pathsafe** (*bool*) – (default = False)

Returns arr_hashstr

Return type str

CommandLine: python -m utool.util_hash –test-hashstr_arr python -m utool.util_hash hashstr_arr:2

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_hash import *  # NOQA
>>> import numpy as np
>>> arr = np.array([[1, 2, 3], [4, 5, 6]], dtype=np.float64)
>>> lbl = 'arr'
>>> kwargs = {}
>>> pathsafe = False
>>> arr_hashstr = hashstr_arr(arr, lbl, pathsafe, alphabet=ALPHABET_27)
>>> result = ('arr_hashstr = %s' % (str(arr_hashstr),))
>>> print(result)
arr_hashstr = arr((2,3)daukyreqnhfejkfs)
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_hash import *  # NOQA
>>> import numpy as np
>>> arr = np.array([[1, 2, 3], [4, 5, 6]], dtype=np.float64)
>>> kwargs = {}
>>> lbl = 'arr'
>>> pathsafe = True
>>> arr_hashstr = hashstr_arr(arr, lbl, pathsafe, alphabet=ALPHABET_27)
>>> result = ('arr_hashstr = %s' % (str(arr_hashstr),))
>>> print(result)
arr_hashstr = arr-_2,3_daukyreqnhfejkfs-
```

utool.util_hash.**hashstr_arr27**(*arr*, *lbl*, *alphabet=['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']*, ***kwargs*)

utool.util_hash.**hashstr_md5**(*data*)

Ignore:

```
>>> from utool.util_hash import *  # NOQA
>>> fpath = ut.grab_test_imgpath('patsy.jpg')
>>> %timeit ut.get_file_hash(fpath, hasher=hashlib.sha1())
>>> %timeit ut.get_file_hash(fpath, hasher=hashlib.md5())
```

utool.util_hash.**hashstr_sha1**(*data*, *base10=False*)

utool.util_hash.**image_uuid**(*pil_img*)

UNSAFE: DEPRICATE: JPEG IS NOT GAURENTEED TO PRODUCE CONSITENT VALUES ON MULTIPLE MACHINES image global unique id

References

http://stackoverflow.com/questions/23565889/jpeg-images-have-different-pixel-values-across-multiple-devices

utool.util_hash.**make_hash**(*o*)
Makes a hash from a dictionary, list, tuple or set to any level, that contains only other hashable types (including any lists, tuples, sets, and dictionaries). In the case where other kinds of objects (like classes) need to be hashed, pass in a collection of object attributes that are pertinent. For example, a class can be hashed in this fashion:

make_hash([cls.__dict__, cls.__name__])

A function can be hashed like so:

make_hash([fn.__dict__, fn.__code__])

References

http://stackoverflow.com/questions/5884066/hashing-a-python-dictionary

utool.util_hash.**random_nonce**(*length=64*, *alphabet=None*)
returns a random string of len=<length> from <alphabet> I have no idea why this is named random_nonce

utool.util_hash.**random_uuid**()

utool.util_hash.**write_hash_file**(*fpath*, *hash_tag='md5'*, *recompute=False*)
Creates a hash file for each file in a path

CommandLine: python -m utool.util_hash –test-write_hash_file

Ignore:

```
>>> # DISABLE_DOCTEST
>>> import utool as ut
>>> from utool.util_hash import *  # NOQA
>>> fpath = ut.grab_test_imgpath('patsy.jpg')
>>> write_hash_file(fpath, 'md5')
```

utool.util_hash.**write_hash_file_for_path**(*path*, *recompute=False*)
Creates a hash file for each file in a path

CommandLine: python -m utool.util_hash –test-write_hash_file_for_path

Ignore:

```
>>> # DISABLE_DOCTEST
>>> import os
>>> import utool as ut
>>> from utool.util_hash import *  # NOQA
>>> fpath = ut.grab_test_imgpath('patsy.jpg')
>>> path, _ = os.path.split(fpath)
>>> hash_fpath_list = write_hash_file_for_path(path)
>>> for hash_fpath in hash_fpath_list:
>>>     assert os.path.exists(hash_fpath)
>>>     ut.delete(hash_fpath)
```

1.31 utool.util_import module

SeeAlso: utool._internal.util_importer

TODO: http://code.activestate.com/recipes/473888-lazy-module-imports/ https://pypi.python.org/pypi/zope.deferredimport/3.5.2

utool.util_import.**check_module_installed**(*modname*)

Check if a python module is installed without attempting to import it. Note, that if `modname` indicates a child module, the parent module is always loaded.

Parameters **modname** (*str*) – module name

Returns found

Return type bool

References

http://stackoverflow.com/questions/14050281/module-exists-without-importing

CommandLine: python -m utool.util_import check_module_installed –show –verbimp –modname=this python -m utool.util_import check_module_installed –show –verbimp –modname=wbia.scripts.iccv

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_import import *  # NOQA
>>> import utool as ut
>>> modname = ut.get_argval('--modname', default='this')
>>> is_installed = check_module_installed(modname)
>>> is_imported = modname in sys.modules
>>> print('module(%r).is_installed = %r' % (modname, is_installed))
>>> print('module(%r).is_imported = %r' % (modname, is_imported))
>>> assert 'this' not in sys.modules, 'module(this) should not have ever been␣
↪imported'
```

utool.util_import.**get_modpath_from_modname**(*modname*, *prefer_pkg=False*, *prefer_main=False*)

Same as get_modpath but doesnt import directly

SeeAlso: get_modpath

utool.util_import.**import_modname**(*modname*)

Parameters **modname** (*str*) – module name

Returns module

Return type module

CommandLine: python -m utool.util_import –test-import_modname

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_import import *  # NOQA
>>> modname_list = [
>>>     'utool',
>>>     'utool._internal',
>>>     'utool._internal.meta_util_six',
```

(continues on next page)

(continued from previous page)

```
>>>     'utool.util_path',
>>>     #'utool.util_path.checkpath',
>>> ]
>>> modules = [import_modname(modname) for modname in modname_list]
>>> result = ([m.__name__ for m in modules])
>>> assert result == modname_list
```

utool.util_import.**import_module_from_fpath**(*module_fpath*)
imports module from a file path

Parameters **module_fpath** (*str*) –

Returns module

Return type module

CommandLine: python -m utool.util_import –test-import_module_from_fpath

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_import import *  # NOQA
>>> import utool
>>> module_fpath = utool.__file__
>>> module = import_module_from_fpath(module_fpath)
>>> result = ('module = %s' % (str(module),))
>>> print(result)
```

Ignore:

```
>>> import shutil
>>> import ubelt as ub
>>> test_root = ub.ensure_app_cache_dir('test_fpath_import')
>>> # Clear the directory
>>> shutil.rmtree(test_root)
>>> test_root = ub.ensure_app_cache_dir('test_fpath_import')
>>>
>>> # -----
>>> # Define two temporary modules with the same name that are not in sys.path
>>> import sys, os, os.path
>>> from os.path import join
>>>
>>> # Even though they have the same name they have different values
>>> mod1_fpath = ub.ensuredir((test_root, 'path1', 'testmod'))
>>> ub.writeto(join(mod1_fpath, '__init__.py'), 'version = 1\nfrom . import␣
↪sibling\na1 = 1')
>>> ub.writeto(join(mod1_fpath, 'sibling.py'), 'spam = \"ham\"\nb1 = 2')
>>>
>>> # Even though they have the same name they have different values
>>> mod2_fpath = ub.ensuredir((test_root, 'path2', 'testmod'))
>>> ub.writeto(join(mod2_fpath, '__init__.py'), 'version = 2\nfrom . import␣
↪sibling\na2 = 3')
>>> ub.writeto(join(mod2_fpath, 'sibling.py'), 'spam = \"jam\"\nb2 = 4')
>>>
>>> # -----
```

(continues on next page)

(continued from previous page)

```
>>> # Neither module should be importable through the normal mechanism
>>> try:
>>>     import testmod
>>>     assert False, 'should fail'
>>> except ImportError as ex:
>>>     pass
>>>
>>> mod1 = ut.import_module_from_fpath(mod1_fpath)
>>> print('mod1.version = {!r}'.format(mod1.version))
>>> print('mod1.version = {!r}'.format(mod1.version))
>>> print(mod1.version == 1, 'mod1 version is 1')
>>> print('mod1.a1 = {!r}'.format(mod1.a1))
>>>
>>> mod2 = ut.import_module_from_fpath(mod2_fpath)
>>> print('mod2.version = {!r}'.format(mod2.version))
>>> print(mod2.version == 2, 'mod2 version is 2')
>>> print('mod2.a2 = {!r}'.format(mod1.a2))
>>>
>>> # BUT Notice how mod1 is mod2
>>> print(mod1 is mod2)
>>>
>>> # mod1 has attributes from mod1 and mod2
>>> print('mod1.a1 = {!r}'.format(mod1.a1))
>>> print('mod1.a2 = {!r}'.format(mod1.a2))
>>> print('mod2.a1 = {!r}'.format(mod2.a1))
>>> print('mod2.a2 = {!r}'.format(mod2.a2))
>>>
>>> # Both are version 2
>>> print('mod1.version = {!r}'.format(mod1.version))
>>> print('mod2.version = {!r}'.format(mod2.version))
>>>
>>> # However sibling always remains at version1 (ham)
>>> print('mod2.sibling.spam = {!r}'.format(mod2.sibling.spam))
>>>
>>> # now importing testmod works because it reads from sys.modules
>>> import testmod
>>>
>>> # reloading mod1 overwrites attrs again
>>> mod1 = ut.import_module_from_fpath(mod1_fpath)
>>>
>>> # Removing both from sys.modules
>>> del sys.modules['testmod']
>>> del sys.modules['testmod.sibling']
>>> mod2 = ut.import_module_from_fpath(mod2_fpath)
>>>
>>> print(not hasattr(mod2, 'a1'),
>>>     'mod2 no longer has a1 and it reloads itself correctly')
>>>
>>>
>>> # -------
>>>
>>> del sys.modules['testmod']
>>> del sys.modules['testmod.sibling']
>>> mod1 = ut.import_module_from_fpath(mod1_fpath)
>>>
>>>
>>> # third test
```

(continues on next page)

(continued from previous page)

```
>>> mod3_fpath = ub.ensuredir((test_root, 'path3', 'testmod'))
>>> ub.writeto(join(mod3_fpath, '__init__.py'), 'version = 3')
>>>
>>> module_fpath = mod3_fpath
>>> modname = 'testmod'
>>>
>>> # third test
>>> mod4_fpath = ub.ensuredir((test_root, 'path3', 'novelmod'))
>>> ub.writeto(join(mod4_fpath, '__init__.py'), 'version = 4')
```

utool.util_import.**import_star**(*modname*, *parent=None*)

Parameters **modname** (*str*) – module name

Tutorial: Replacement for from modname import *

Usage is like this globals().update(ut.import_star('<module>')) OR ut.import_star('<module>', __name__)

Ignore:

```
>>> from utool.util_import import *  # NOQA
modname = 'opengm'
submod = None
parent = __name__
```

utool.util_import.**import_star_execstr**(*modname*, *parent=None*)
print(ut.import_star_execstr('opengm.inference'))

utool.util_import.**package_contents**(*package*, *with_pkg=False*, *with_mod=True*, *ignore_prefix=[]*, *ignore_suffix=[]*)

References

http://stackoverflow.com/questions/1707709/list-all-the-modules-that-are-part-of-a-python-package

Parameters

- **package** –
- **with_pkg** (*bool*) – (default = False)
- **with_mod** (*bool*) – (default = True)

CommandLine: python -m utool.util_import –exec-package_contents

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_import import *  # NOQA
>>> import utool as ut
>>> import wbia
>>> package = wbia
>>> ignore_prefix = ['wbia.tests', 'wbia.control.__SQLITE3__',
>>>                  '_autogen_explicit_controller']
```

(continues on next page)

(continued from previous page)

```
>>> ignore_suffix = ['_grave']
>>> with_pkg = False
>>> with_mod = True
>>> result = package_contents(package, with_pkg, with_mod,
>>>                           ignore_prefix, ignore_suffix)
>>> print(ut.repr2(result))
```

utool.util_import.**possible_import_patterns**(*modname*)
does not support from x import * does not support from x import z, y

Example

```
>>> # DISABLE_DOCTEST
>>> import utool as ut
>>> modname = 'package.submod.submod2.module'
>>> result = ut.repr3(ut.possible_import_patterns(modname))
>>> print(result)
[
    'import\spackage.submod.submod2.module',
    'from\spackage\.submod\.submod2\simportmodule',
]
```

utool.util_import.**tryimport**(*modname*, *pipiname=None*, *ensure=False*)

CommandLine: python -m utool.util_import –test-tryimport

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_tests import *  # NOQA
>>> import utool as ut
>>> modname = 'pyfiglet'
>>> pipiname = 'git+https://github.com/pwaller/pyfiglet'
>>> pyfiglet = ut.tryimport(modname, pipiname)
>>> assert pyfiglet is None or isinstance(pyfiglet, types.ModuleType), 'unknown␣
↪error'
```

Example

```
>>> # UNSTABLE_DOCTEST
>>> # disabled because not everyone has access to being a super user
>>> from utool.util_tests import *  # NOQA
>>> import utool as ut
>>> modname = 'lru'
>>> pipiname = 'git+https://github.com/amitdev/lru-dict'
>>> lru = ut.tryimport(modname, pipiname, ensure=True)
>>> assert isinstance(lru, types.ModuleType), 'did not ensure lru'
```

1.32 utool.util_inject module

Injects code into live modules or into text source files.

Basic use case is to extend the print function into a logging function

utool.util_inject.**DUMMYPROF_FUNC**(*func*)
dummy profiling func. does nothing

utool.util_inject.**PROFILE_FUNC**(*func*)
dummy profiling func. does nothing

utool.util_inject.**TIMERPROF_FUNC**(*func*)

utool.util_inject.**colored_pygments_excepthook**(*type_*, *value*, *tb*)

References

https://stackoverflow.com/questions/14775916/color-exceptions-python

CommandLine: python -m utool.util_inject –test-colored_pygments_excepthook

utool.util_inject.**get_injected_modules**()

utool.util_inject.**inject**(*module_name=None*, *module_prefix='[???]'*, *DEBUG=False*, *module=None*, *N=1*)
Injects your module with utool magic

Utool magic is not actually magic. It just turns your `print` statments into logging statments, allows for your module to be used with the utool.Indent context manager and the and utool.indent_func decorator. `printDBG` will soon be deprecated as will `print\_`. The function rrr is a developer convinience for reloading the module dynamically durring runtime. The profile decorator is a no-op if not using kernprof.py, otherwise it is kernprof.py's profile decorator.

Parameters

- **module_name** (*str*) – the __name__ varaible in your module
- **module_prefix** (*str*) – a user defined module prefix
- **DEBUG** (*bool*) –
- **module** (*None*) – the actual module (optional)

Returns (print, print_, printDBG, rrr, profile_)

Return type tuple

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_inject import *  # NOQA
>>> from __future__ import absolute_import, division, print_function, unicode_
↪literals
>>> from util.util_inject import inject
>>> print, rrr, profile = inject2(__name__, '[mod]')
```

utool.util_inject.**inject2**(*module_name=None*, *module_prefix=None*, *DEBUG=False*, *module=None*, *N=1*)
wrapper that depricates print_ and printDBG

utool.util_inject.**inject_colored_exceptions**()
Causes exceptions to be colored if not already

Hooks into sys.excepthook

CommandLine: python -m utool.util_inject –test-inject_colored_exceptions

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_inject import *  # NOQA
>>> print('sys.excepthook = %r ' % (sys.excepthook,))
>>> #assert sys.excepthook is colored_pygments_excepthook, 'bad excepthook'
>>> raise Exception('should be in color')
```

utool.util_inject.**inject_print_functions**(*module_name=None*, *module_prefix='[???]'*, *DEBUG=False*, *module=None*)
makes print functions to be injected into the module

utool.util_inject.**inject_python_code**(*fpath*, *patch_code*, *tag=None*, *inject_location='after_imports'*)
DEPRICATE puts code into files on disk

utool.util_inject.**inject_python_code2**(*fpath*, *patch_code*, *tag*)
Does autogeneration stuff

utool.util_inject.**make_module_print_func**(*module*)

utool.util_inject.**make_module_profile_func**(*module_name=None*, *module_prefix='[???]'*, *module=None*)

utool.util_inject.**make_module_reload_func**(*module_name=None*, *module_prefix='[???]'*, *module=None*)
Injects dynamic module reloading

utool.util_inject.**make_module_write_func**(*module*)

utool.util_inject.**memprof**(*func*)
requires memory_profiler pip install memory_profiler

References

https://pypi.python.org/pypi/memory_profiler

utool.util_inject.**noinject**(*module_name=None*, *module_prefix='[???]'*, *DEBUG=False*, *module=None*, *N=0*, *via=None*)
Use in modules that do not have inject in them

Does not inject anything into the module. Just lets utool know that a module is being imported so the import order can be debuged

utool.util_inject.**print**(**args*, ***kwargs*)
logging builtins.print

utool.util_inject.**profile**(*func*)

utool.util_inject.**reload_module**(*module*, *verbose=None*)

utool.util_inject.**rrr**(*verbose=True*)
Dynamic module reloading

utool.util_inject.**split_python_text_into_lines**(*text*)
TODO: make it so this function returns text so one statment is on one # line that means no splitting up things like function definitions into # multiple lines

1.33 utool.util_inspect module

class utool.util_inspect.**BaronWraper**(*sourcecode*)
Bases: object

defined_functions(*recursive=True*)

find_func(*name*)

find_root_function(*node*)

find_usage(*name*)

classmethod from_fpath(*fpath*)

internal_call_graph(*with_doctests=False*)

print_diff(*fpath=None*)

rrr(*verbose=True*, *reload_module=True*)
special class reloading function This function is often injected as rrr of classes

to_string()

write(*fpath=None*)

class utool.util_inspect.**KWReg**(*enabled=False*)
Bases: object

Helper to register keywords for complex keyword parsers

defaultkw

print_defaultkw()

utool.util_inspect.**argparse_funckw**(*func*, *defaults={}*, ***kwargs*)
allows kwargs to be specified on the commandline from testfuncs

Parameters **func** (*function*) –

Kwargs: lbl, verbose, only_specified, force_keys, type_hint, alias_dict

Returns funckw

Return type dict

CommandLine: python -m utool.util_inspect argparse_funckw

SeeAlso: exec_funckw recursive_parse_kwargs parse_kwarg_keys

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_inspect import *  # NOQA
>>> import utool as ut
>>> func = get_instance_attrnames
>>> funckw = argparse_funckw(func)
>>> result = ('funckw = %s' % (ut.repr3(funckw),))
>>> print(result)
funckw = {
    'default': True,
```

(continues on next page)

(continued from previous page)

```
    'with_methods': True,
    'with_properties': True,
}
```

utool.util_inspect.**check_module_usage**(*modpath_patterns*)

FIXME: not fully implmented

Desired behavior is — Given a set of modules specified by a list of patterns, returns how the functions defined in the modules are called: a) with themselves and b) by other files in the project not in the given set.

Parameters **modpath_patterns** (*list*) –

CommandLine: python -m utool.util_inspect check_module_usage –show utprof.py -m utool.util_inspect check_module_usage –show python -m utool.util_inspect check_module_usage –pat="['auto*', 'user_dialogs.py', 'special_query.py', 'qt_inc_automatch.py', 'devcases.py']" python -m utool.util_inspect check_module_usage –pat="preproc_detectimg.py" python -m utool.util_inspect check_module_usage –pat="neighbor_index.py" python -m utool.util_inspect check_module_usage –pat="manual_chip_funcs.py" python -m utool.util_inspect check_module_usage –pat="preproc_probchip.py" python -m utool.util_inspect check_module_usage –pat="guiback.py"

python -m utool.util_inspect check_module_usage –pat="util_str.py"

Ignore:

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_inspect import *  # NOQA
>>> import utool as ut
>>> modpath_patterns = ['_grave*']
>>> modpath_patterns = ['auto*', 'user_dialogs.py', 'special_query.py', 'qt_inc_
↪automatch.py', 'devcases.py']
>>> modpath_patterns = ['neighbor_index.py']
>>> modpath_patterns = ['manual_chip_funcs.py']
>>> modpath_patterns = ut.get_argval('--pat', type_=list, default=['*'])
>>> result = check_module_usage(modpath_patterns)
>>> print(result)
```

utool.util_inspect.**check_static_member_vars**(*class_*, *fpath=None*, *only_init=True*)

class_ can either be live object or a classname

fpath = ut.truepath('~/code/ibeis/ibeis/viz/viz_graph2.py') # classname = 'AnnotGraphWidget'

utool.util_inspect.**dummy_func**(*arg1, arg2, arg3=None, arg4=[1, 2, 3], arg5={}, **kwargs*)

test func for kwargs parseing

utool.util_inspect.**exec_func_doctest**(*func*, *start_sentinal=None*, *end_sentinal=None*, *num=0*, *globals_=None*, *locals_=None*)

execs a func doctest and returns requested local vars.

func = encoder.learn_threshold2 num = 0 start_sentinal = 'import wbia.plottool as pt' end_sentinal = 'pnum_ = pt.make_pnum_nextgen'

utool.util_inspect.**exec_func_src**(*func*, *globals_=None*, *locals_=None*, *key_list=None*, *sentinal=None*, *update=None*, *keys=None*, *verbose=False*, *start=None*, *stop=None*)

execs a func and returns requested local vars.

Does not modify globals unless update=True (or in IPython)

SeeAlso: ut.execstr_funckw

utool.util_inspect.**exec_func_src2**(*func*, *globals_=None*, *locals_=None*, *sentinal=None*, *verbose=False*, *start=None*, *stop=None*)

execs a func and returns requested local vars.

Does not modify globals unless update=True (or in IPython)

SeeAlso: ut.execstr_funckw

utool.util_inspect.**exec_func_src3**(*func*, *globals_*, *sentinal=None*, *verbose=False*, *start=None*, *stop=None*)

execs a func and returns requested local vars.

Does not modify globals unless update=True (or in IPython)

SeeAlso: ut.execstr_funckw

utool.util_inspect.**execstr_func_doctest**(*func*, *num=0*, *start_sentinal=None*, *end_sentinal=None*)

execs a func doctest and returns requested local vars.

```
>>> from utool.util_inspect import *  # NOQA
```

func = encoder.learn_threshold2 num = 0 start_sentinal = 'import wbia.plottool as pt' end_sentinal = 'pnum_ = pt.make_pnum_nextgen'

utool.util_inspect.**filter_valid_kwargs**(*func*, *dict_*)

utool.util_inspect.**find_funcs_called_with_kwargs**(*sourcecode*, *target_kwargs_name='kwargs'*)

Finds functions that are called with the keyword *kwargs* variable

CommandLine: python3 -m utool.util_inspect find_funcs_called_with_kwargs

Example

```
>>> # ENABLE_DOCTEST
>>> import utool as ut
>>> sourcecode = ut.codeblock(
        '''
        x, y = list(zip(*ut.ichunks(data, 2)))
        somecall(arg1, arg2, arg3=4, **kwargs)
        import sys
        sys.badcall(**kwargs)
        def foo():
            bar(**kwargs)
            ut.holymoly(**kwargs)
            baz()
            def biz(**kwargs):
                foo2(**kwargs)
        ''')
>>> child_funcnamess = ut.find_funcs_called_with_kwargs(sourcecode)
>>> print('child_funcnamess = %r' % (child_funcnamess,))
>>> assert 'foo2' not in child_funcnamess, 'foo2 should not be found'
>>> assert 'bar' in child_funcnamess, 'bar should be found'
```

utool.util_inspect.**get_argnames**(*func*)

utool.util_inspect.**get_dev_hints**()

utool.util_inspect.**get_docstr**(*func_or_class*)
Get the docstring from a live object

utool.util_inspect.**get_func_argspec**(*func*)
wrapper around inspect.getfullargspec but takes into account utool decorators

utool.util_inspect.**get_func_docblocks**(*func_or_class*)

utool.util_inspect.**get_func_kwargs**(*func*, *recursive=True*)
func = wbia.run_experiment

SeeAlso: argparse_funckw recursive_parse_kwargs parse_kwarg_keys parse_func_kwarg_keys get_func_kwargs

utool.util_inspect.**get_func_modname**(*func*)

utool.util_inspect.**get_func_sourcecode**(*func*, *stripdef=False*, *stripret=False*, *strip_docstr=False*, *strip_comments=False*, *remove_linenums=None*)
wrapper around inspect.getsource but takes into account utool decorators strip flags are very hacky as of now

Parameters

- **func** (*function*) –
- **stripdef** (*bool*) –
- **stripret** (*bool*) – (default = False)
- **strip_docstr** (*bool*) – (default = False)
- **strip_comments** (*bool*) – (default = False)
- **remove_linenums** (*None*) – (default = None)

CommandLine: python -m utool.util_inspect –test-get_func_sourcecode

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_inspect import *  # NOQA
>>> # build test data
>>> func = get_func_sourcecode
>>> stripdef = True
>>> stripret = True
>>> sourcecode = get_func_sourcecode(func, stripdef)
>>> # verify results
>>> print(result)
```

utool.util_inspect.**get_funcdoc**(*func*)

utool.util_inspect.**get_funcfpath**(*func*)

utool.util_inspect.**get_funcglobals**(*func*)

utool.util_inspect.**get_funckw**(*func*, *recursive=True*)

utool.util_inspect.**get_funcname**(*func*)

utool.util_inspect.**get_funcnames_from_modpath**(*modpath*, *include_methods=True*)
Get all functions defined in module

utool.util_inspect.**get_instance_attrnames**(*obj*, *default=True*, ***kwargs*)

utool.util_inspect.**get_internal_call_graph**(*fpath*, *with_doctests=False*)

CommandLine: python -m utool.util_inspect get_internal_call_graph –show –modpath=~/code/ibeis/ibeis/init/main_helpers.py –show python -m utool.util_inspect get_internal_call_graph –show –modpath=~/code/dtool/dtool/depcache_table.py –show

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_inspect import *  # NOQA
>>> import utool as ut
>>> fpath = ut.get_argval('--modpath', default='.')
>>> with_doctests = ut.get_argflag('--with_doctests')
>>> G = get_internal_call_graph(fpath, with_doctests)
>>> ut.quit_if_noshow()
>>> import wbia.plottool as pt
>>> pt.qt4ensure()
>>> pt.show_nx(G, fontsize=8, as_directed=False)
>>> z = pt.zoom_factory()
>>> p = pt.pan_factory()
>>> ut.show_if_requested()
```

utool.util_inspect.**get_kwargs**(*func*)

Parameters **func** (*function*) –

Returns keys, is_arbitrary keys (list): kwargs keys is_arbitrary (bool): has generic **kwargs

Return type tuple

CommandLine: python -m utool.util_inspect –test-get_kwargs

Ignore:

```
>>> def func1(a, b, c):
>>>     pass
>>> def func2(a, b, c, *args):
>>>     pass
>>> def func3(a, b, c, *args, **kwargs):
>>>     pass
>>> def func4(a, b=1, c=2):
>>>     pass
>>> def func5(a, b=1, c=2, *args):
>>>     pass
>>> def func6(a, b=1, c=2, **kwargs):
>>>     pass
>>> def func7(a, b=1, c=2, *args, **kwargs):
>>>     pass
>>> for func in [locals()['func' + str(x)] for x in range(1, 8)]:
>>>     print(inspect.getfullargspec(func))
```

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_inspect import *  # NOQA
>>> # build test data
>>> func = '?'
>>> result = get_kwargs(func)
>>> # verify results
>>> print(result)
```

utool.util_inspect.**get_kwdefaults**(*func*, *parse_source=False*)

Parameters **func** (*func*) –

Returns

Return type dict

CommandLine: python -m utool.util_inspect get_kwdefaults

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_inspect import *  # NOQA
>>> import utool as ut
>>> func = dummy_func
>>> parse_source = True
>>> kwdefaults = get_kwdefaults(func, parse_source)
>>> print('kwdefaults = %s' % (ut.repr4(kwdefaults),))
```

utool.util_inspect.**get_kwdefaults2**(*func*, *parse_source=False*)

utool.util_inspect.**get_method_func**(*func*)

utool.util_inspect.**get_module_from_class**(*class_*)

utool.util_inspect.**get_module_owned_functions**(*module*)
Replace with iter_module_doctesable (but change that name to be something better)

returns functions actually owned by the module module = vtool.distance

utool.util_inspect.**get_object_methods**(*obj*)
Returns all methods belonging to an object instance specified in by the __dir__ function

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_inspect import *  # NOQA
>>> import utool as ut
>>> obj = ut.NiceRepr()
>>> methods1 = ut.get_object_methods()
>>> ut.inject_func_as_method(obj, ut.get_object_methods)
>>> methods2 = ut.get_object_methods()
>>> assert ut.get_object_methods in methods2
```

utool.util_inspect.**get_unbound_args**(*argspec*)

utool.util_inspect.**help_members**(*obj*, *use_other=False*)
Inspects members of a class

Parameters **obj** (*class or module*) –

CommandLine: python -m utool.util_inspect help_members

Ignore:

```
>>> # ENABLE_DOCTEST
>>> from utool.util_inspect import *  # NOQA
>>> import utool as ut
>>> obj = ut.DynStruct
>>> result = help_members(obj)
>>> print(result)
```

utool.util_inspect.**infer_arg_types_and_descriptions**(*argname_list*, *defaults*)

Parameters

- **argname_list** (*list*) –
- **defaults** (*list*) –

Returns (argtype_list, argdesc_list)

Return type tuple

CommandLine: python -m utool.util_inspect –test-infer_arg_types_and_descriptions

Ignore: python -c "import utool; print(utool.auto_docstr('wbia.algo.hots.pipeline', 'build_chipmatches'))"

Example

```
>>> # ENABLE_DOCTEST
>>> import utool
>>> argname_list = ['ibs', 'qaid', 'fdKfds', 'qfx2_foo']
>>> defaults = None
>>> tup = utool.infer_arg_types_and_descriptions(argname_list, defaults)
>>> argtype_list, argdesc_list, argdefault_list, hasdefault_list = tup
```

utool.util_inspect.**infer_function_info**(*func*)

Infers information for make_default_docstr # TODO: Interleave old documentation with new documentation

Parameters **func** (*function*) – live python function

CommandLine: python -m utool –tf infer_function_info:0 python -m utool –tf infer_function_info:1 –funcname=wbia_cnn.models.siam.ignore_hardest_cases

Ignore:

```
>>> # ENABLE_DOCTEST
>>> from utool.util_inspect import *  # NOQA
>>> import utool as ut
>>> func = ut.infer_function_info
>>> #func = ut.Timer.tic
>>> func = ut.make_default_docstr
>>> funcinfo = infer_function_info(func)
>>> result = ut.repr4(funcinfo.__dict__)
>>> print(result)
```

Ignore:

```
>>> # DISABLE_DOCTEST
>>> # SCRIPT
>>> from utool.util_inspect import *  # NOQA
>>> import utool as ut
>>> funcname = ut.get_argval('--funcname')
>>> # Parse out custom function
>>> modname = '.'.join(funcname.split('.')[0:-1])
>>> script = 'import {modname}\nfunc = {funcname}'.format(
>>>     modname=modname, funcname=funcname)
>>> globals_, locals_ = {}, {}
>>> exec(script, globals_, locals_)
>>> func = locals_['func']
>>> funcinfo = infer_function_info(func)
>>> result = ut.repr4(funcinfo.__dict__)
>>> print(result)
```

utool.util_inspect.**inherit_kwargs**(*inherit_func*)

TODO move to util_decor inherit_func = inspect_pdfs func = encoder.visualize.im_func

utool.util_inspect.**is_bateries_included**(*item*)

Returns if a value is a python builtin function

Parameters **item** (*object*) –

Returns flag

Return type bool

References

http://stackoverflow.com/questions/23149218/check-if-a-python-function-is-builtin

CommandLine: python -m utool._internal.meta_util_six is_builtin

Example

```
>>> # DISABLE_DOCTEST
>>> from utool._internal.meta_util_six import *  # NOQA
>>> item = zip
>>> flag = is_bateries_included(item)
>>> result = ('flag = %s' % (str(flag),))
>>> print(result)
```

utool.util_inspect.**is_defined_by_module**(*item*, *module*, *parent=None*)

Check if item is directly defined by a module. This check may be prone to errors.

utool.util_inspect.**is_defined_by_module2**(*item*, *module*)

utool.util_inspect.**is_valid_python**(*code*, *reraise=True*, *ipy_magic_workaround=False*)

References

http://stackoverflow.com/questions/23576681/python-check-syntax

utool.util_inspect.**iter_module_doctestable**(*module*, *include_funcs=True*, *include_classes=True*, *include_methods=True*, *include_builtin=True*, *include_inherited=False*, *debug_key=None*)

Yeilds doctestable live object form a modules

TODO: change name to iter_module_members Replace with iter_module_doctesable (but change that name to be something better)

Parameters

- **module** (*module*) – live python module
- **include_funcs** (*bool*) –
- **include_classes** (*bool*) –
- **include_methods** (*bool*) –
- **include_builtin** (*bool*) – (default = True)
- **include_inherited** (*bool*) – (default = False)

Yeilds: tuple (str, callable): (funcname, func) doctestable

CommandLine:

python -m utool –tf iter_module_doctestable –modname=wbia.algo.hots.chip_match –modname=wbia.control.IBEISControl –modname=wbia.control.SQLDatabaseControl –modname=wbia.control.manual_annot_funcs –modname=wbia.control.manual_annot_funcs –modname=wbia.expt.test_result –modname=utool.util_progress –debug-key=build_msg_fmtstr_time2 –modname=utool.util_progress –debug-key=ProgressIter

Debug: # fix profile with doctest utprof.py -m utool –tf iter_module_doctestable –modname=utool.util_inspect –debugkey=zzz_profiled_is_yes utprof.py -m utool –tf iter_module_doctestable –modname=wbia.algo.hots.chip_match –debugkey=to_json

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_inspect import *   # NOQA
>>> import utool as ut
>>> modname = ut.get_argval('--modname', type_=str, default=None)
>>> kwargs = ut.argparse_funckw(iter_module_doctestable)
>>> module = ut.util_tests if modname is None else ut.import_modname(modname)
>>> debug_key = ut.get_argval('--debugkey', type_=str, default=None)
>>> kwargs['debug_key'] = debug_key
>>> kwargs['include_inherited'] = True
>>> doctestable_list = list(iter_module_doctestable(module, **kwargs))
>>> func_names = sorted(ut.take_column(doctestable_list, 0))
>>> print(ut.repr4(func_names))
```

utool.util_inspect.**list_class_funcnames**(*fname, blank_pats=[' #']*)

Parameters

- **fname** (*str*) – filepath
- **blank_pats** (*list*) – defaults to ' #'

Returns funcname_list

Return type list

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_inspect import *  # NOQA
>>> fname = 'util_class.py'
>>> blank_pats = ['    #']
>>> funcname_list = list_class_funcnames(fname, blank_pats)
>>> print(funcname_list)
```

utool.util_inspect.**list_global_funcnames**(*fname, blank_pats=[' #']*)

Parameters

- **fname** (*str*) – filepath
- **blank_pats** (*list*) – defaults to ' #'

Returns funcname_list

Return type list

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_inspect import *  # NOQA
>>> fname = 'util_class.py'
>>> blank_pats = ['    #']
>>> funcname_list = list_global_funcnames(fname, blank_pats)
>>> print(funcname_list)
```

utool.util_inspect.**lookup_attribute_chain**(*attrname*, *namespace*)

Ignore:

```
>>> attrname = funcname
>>> namespace = mod.__dict__
```

```
>>> import utool as ut
>>> globals_ = ut.util_inspect.__dict__
>>> attrname = 'KWReg.print_defaultkw'
```

utool.util_inspect.**parse_func_kwarg_keys**(*func*, *with_vals=False*)

hacky inference of kwargs keys

SeeAlso: argparse_funckw recursive_parse_kwargs parse_kwarg_keys parse_func_kwarg_keys get_func_kwargs

utool.util_inspect.**parse_function_names**(*sourcecode*, *top_level=True*, *ignore_condition=1*)

Finds all function names in a file without importing it

Parameters **sourcecode** (*str*) –

Returns func_names

Return type list

CommandLine: python -m utool.util_inspect parse_function_names

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_inspect import *  # NOQA
>>> import utool as ut
>>> fpath = ut.util_inspect.__file__.replace('.pyc', '.py')
>>> #fpath = ut.truepath('~/code/bintrees/bintrees/avltree.py')
>>> sourcecode = ut.readfrom(fpath)
>>> func_names = parse_function_names(sourcecode)
>>> result = ('func_names = %s' % (ut.repr2(func_names),))
>>> print(result)
```

utool.util_inspect.**parse_import_names**(*sourcecode*, *top_level=True*, *fpath=None*, *branch=False*)

Finds all function names in a file without importing it

Parameters **sourcecode** (*str*) –

Returns func_names

Return type list

CommandLine: python -m utool.util_inspect parse_import_names

References

https://stackoverflow.com/questions/20445733/how-to-tell-which-modules-have-been-imported-in-some-source-code

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_inspect import *  # NOQA
>>> import utool as ut
>>> fpath = ut.util_inspect.__file__.replace('.pyc', '.py')
>>> #fpath = ut.truepath('~/code/bintrees/bintrees/avltree.py')
>>> sourcecode = ut.readfrom(fpath)
>>> func_names = parse_import_names(sourcecode)
>>> result = ('func_names = %s' % (ut.repr2(func_names),))
>>> print(result)
```

utool.util_inspect.**parse_kwarg_keys**(*source*, *keywords='kwargs'*, *with_vals=False*)

Parses the source code to find keys used by the ***kwargs* keywords dictionary variable. if *with_vals* is True, we also attempt to infer the default values.

Parameters **source** (*str*) –

Returns kwarg_keys

Return type list

CommandLine: python -m utool.util_inspect parse_kwarg_keys

python -m utool.util_inspect parse_kwarg_keys

SeeAlso: argparse_funckw recursive_parse_kwargs parse_kwarg_keys parse_func_kwarg_keys get_func_kwargs

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_inspect import *  # NOQA
>>> import utool as ut
>>> source = (
>>>           "\n x = 'hidden_x'"
>>>           "\n y = 3 # hidden val"
>>>           "\n kwargs.get(x, y)"
>>>           "\n kwargs.get('foo', None)\n kwargs.pop('bar', 3)"
>>>           "\n kwargs.pop('str', '3fd')\n kwargs.pop('str', '3f\\'d')"
>>>           "\n \"kwargs.get('baz', None)\"\n kwargs['foo2']"
>>>           "\n #kwargs.get('biz', None)\""
>>>           "\n kwargs['bloop']"
>>>           "\n x = 'bop' in kwargs"
>>>           )
>>> print('source = %s\n' % (source,))
>>> ut.exec_funckw(parse_kwarg_keys, globals())
>>> with_vals = True
>>> kwarg_items = parse_kwarg_keys(source, with_vals=with_vals)
>>> result = ('kwarg_items = %s' % (ut.repr2(kwarg_items, nl=1),))
>>> kwarg_keys = ut.take_column(kwarg_items, 0)
>>> assert 'baz' not in kwarg_keys
>>> assert 'foo' in kwarg_keys
>>> assert 'bloop' in kwarg_keys
>>> assert 'bop' not in kwarg_keys
>>> print(result)
kwarg_items = [
    ('foo', None),
    ('bar', 3),
    ('str', '3fd'),
    ('str', "3f'd"),
    ('foo2', None),
    ('bloop', None),
]
```

utool.util_inspect.**parse_project_imports**(*dpath*)

dpath = ub.truepath('~/code/clab/clab')

Script:

```
>>> dpath = ut.get_argval('--dpath')
>>> parse_project_imports()
```

utool.util_inspect.**parse_return_type**(*sourcecode*)

Parameters **sourcecode** –

Returns (return_type, return_name, return_header)

Return type tuple

Ignore: testcase automated_helpers query_vsone_verified

CommandLine: python -m utool.util_inspect parse_return_type python -m utool.util_inspect –test-parse_return_type

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_inspect import *  # NOQA
>>> import utool as ut
>>> sourcecode = ut.codeblock(
...     'def foo(tmp=False):\n'
...     '    bar = True\n'
...     '    return bar\n'
... )
>>> returninfo = parse_return_type(sourcecode)
>>> result = ut.repr2(returninfo)
>>> print(result)
('?', 'bar', 'Returns', '')
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_inspect import *  # NOQA
>>> import utool as ut
>>> sourcecode = ut.codeblock(
...     'def foo(tmp=False):\n'
...     '    return True\n'
... )
>>> returninfo = parse_return_type(sourcecode)
>>> result = ut.repr2(returninfo)
>>> print(result)
('bool', 'True', 'Returns', '')
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_inspect import *  # NOQA
>>> import utool as ut
>>> sourcecode = ut.codeblock(
...     'def foo(tmp=False):\n'
...     '    for i in range(2): \n'
...     '        yield i\n'
... )
>>> returninfo = parse_return_type(sourcecode)
>>> result = ut.repr2(returninfo)
>>> print(result)
('?', 'i', 'Yields', '')
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_inspect import *  # NOQA
>>> import utool as ut
>>> sourcecode = ut.codeblock(
...     'def foo(tmp=False):\n'
...     '    if tmp is True:\n'
```

(continues on next page)

(continued from previous page)

```
...     '        return (True, False)\n'
...     '    elif tmp is False:\n'
...     '        return 1\n'
...     '    else:\n'
...     '        bar = baz()\n'
...     '        return bar\n'
... )
>>> returninfo = parse_return_type(sourcecode)
>>> result = ut.repr2(returninfo)
>>> print(result)
('tuple', '(True, False)', 'Returns', '')
```

utool.util_inspect.**prettyprint_parsetree**(*pt*)
pip install astdump pip install codegen

utool.util_inspect.**recursive_parse_kwargs**(*root_func*, *path_=None*, *verbose=None*)
recursive kwargs parser TODO: rectify with others FIXME: if docstr indentation is off, this fails

SeeAlso: argparse_funckw recursive_parse_kwargs parse_kwarg_keys parse_func_kwarg_keys get_func_kwargs

Parameters

- **root_func** (*function*) – live python function
- **path** (*None*) – (default = None)

Returns

Return type list

CommandLine: python -m utool.util_inspect recursive_parse_kwargs:0 python -m utool.util_inspect recursive_parse_kwargs:0 –verbinspect python -m utool.util_inspect recursive_parse_kwargs:1

python -m utool.util_inspect recursive_parse_kwargs:2 –mod vtool –func ScoreNormalizer.visualize

python -m utool.util_inspect recursive_parse_kwargs:2 –mod wbia.viz.viz_matches –func show_name_matches –verbinspect python -m utool.util_inspect recursive_parse_kwargs:2 –mod wbia.expt.experiment_drawing –func draw_rank_cmc –verbinspect

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_inspect import *  # NOQA
>>> import utool as ut
>>> root_func = iter_module_doctestable
>>> path_ = None
>>> result = ut.repr2(recursive_parse_kwargs(root_func), nl=1)
>>> print(result)
[
    ('include_funcs', True),
    ('include_classes', True),
    ('include_methods', True),
    ('include_builtin', True),
    ('include_inherited', False),
    ('debug_key', None),
]
```

Example

```
>>> # xdoctest: +REQUIRES(module:wbia)
>>> from utool.util_inspect import *  # NOQA
>>> from wbia.algo.hots import chip_match
>>> import utool as ut
>>> recursive_parse_kwargs(chip_match.ChipMatch.show_ranked_matches)
>>> recursive_parse_kwargs(chip_match.ChipMatch)
```

import wbia import utool as ut ibs = wbia.opendb(defaultdb='testdb1') kwkeys1 = ibs.parse_annot_stats_filter_kws() ut.recursive_parse_kwargs(ibs.get_annotconfig_stats, verbose=1) kwkeys2 = list(ut.recursive_parse_kwargs(ibs.get_annotconfig_stats).keys())

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_inspect import *  # NOQA
>>> import utool as ut
>>> modname = ut.get_argval('--mod', type_=str, default='plottool')
>>> funcname = ut.get_argval('--func', type_=str, default='draw_histogram')
>>> mod = ut.import_modname(modname)
>>> root_func = lookup_attribute_chain(funcname, mod.__dict__)
>>> path_ = None
>>> parsed = recursive_parse_kwargs(root_func)
>>> flags = ut.unique_flags(ut.take_column(parsed, 0))
>>> unique = ut.compress(parsed, flags)
>>> print('parsed = %s' % (ut.repr4(parsed),))
>>> print('unique = %s' % (ut.repr4(unique),))
```

utool.util_inspect.**set_funcdoc**(*func*, *newdoc*)

utool.util_inspect.**set_funcname**(*func*, *newname*)

utool.util_inspect.**special_parse_process_python_code**(*sourcecode*)

pip install redbaron http://stackoverflow.com/questions/7456933/python-ast-with-preserved-comments

CommandLine: python -m utool.util_inspect special_parse_process_python_code –show

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_inspect import *  # NOQA
>>> import utool as ut
>>> sourcecode = ut.read_from(ut.util_inspect.__file__)
>>> result = special_parse_process_python_code(sourcecode)
>>> print(result)
```

utool.util_inspect.**zzz_profiled_is_no**()

utool.util_inspect.**zzz_profiled_is_yes**()

1.34 utool.util_io module

class utool.util_io.**FixRenamedUnpickler**
Bases: _pickle.Unpickler

find_class(*module*, *name*)
Return an object from a specified module.

If necessary, the module will be imported. Subclasses may override this method (e.g. to restrict unpickling of arbitrary classes and functions).

This method is called whenever a class or a function object is needed. Both arguments passed are str objects.

utool.util_io.**load_cPkl**(*fpath*, *verbose=None*, *n=None*)
Loads a pickled file with optional verbosity. Aims for compatibility between python2 and python3.

Ignore:

```
>>> import utool as ut
>>> def makedata_simple():
>>>     data = np.empty((500, 2 ** 20), dtype=np.uint8) + 1
>>>     return data
>>> memtrack = ut.MemoryTracker()
>>> # create a large amount of data
>>> data = makedata_simple()
>>> memtrack.report()
>>> print(ut.get_object_size_str(data))
>>> fpath = 'tmp.pkl'
>>> ut.save_cPkl(fpath, data)
>>> print(ut.get_file_nBytes_str('tmp.pkl'))
>>> #del data
>>> memtrack.collect()
>>> memtrack.report()
>>> data = ut.load_cPkl(fpath)
>>> memtrack.report()
```

Ignore:

```
>>> def makedata_complex():
>>>     rng = np.random.RandomState(42)
>>>     item1 = np.empty((100, 2 ** 20), dtype=np.uint8) + 1
>>>     item2 = [np.empty((10, 2 ** 10), dtype=np.uint8) + 1
>>>              for a in range(1000)]
>>>     item3 = {a: np.empty(int(rng.rand() * 10), dtype=np.int16) + 1
>>>                  for a in range(100)}
>>>     item4 = {np.int32(a): np.empty((int(rng.rand() * 10), 2), dtype=np.
↪float64) + 1
>>>                  for a in range(200)}
>>>     data = {'item1': item1, 'item2': item2,
>>>             'item3': item3, 'item4': item4}
>>>     return data
>>> memtrack = ut.MemoryTracker()
>>> # create a large amount of data
>>> data = makedata_complex()
>>> memtrack.report()
>>> print(ut.get_object_size_str(data))
>>> fpath = 'tmp.pkl'
```

(continues on next page)

(continued from previous page)

```
>>> ut.save_cPkl(fpath, data)
>>> print(ut.get_file_nBytes_str('tmp.pkl'))
>>> #del data
>>> memtrack.collect()
>>> memtrack.report()
>>> data2 = ut.load_cPkl(fpath)
>>> memtrack.report()
```

Ignore:

```
>>> import utool as ut
>>> memtrack = ut.MemoryTracker()
>>> cacher = ut.Cacher('tmp', cache_dir='.', cfgstr='foo')
>>> data3 = cacher.ensure(makedata_complex)
>>> memtrack.report()
>>> data4 = cacher.ensure(makedata_complex)
>>> memtrack.report()
>>> import utool as ut
>>> memtrack = ut.MemoryTracker()
>>> fpath = '/home/joncrall/Desktop/smkcache/inva_PZ_
↪Master1VUUIDS((5616)vxihbjwtggyovrto)_vpgwpcafbjkkpjdf.cPkl'
>>> print(ut.get_file_nBytes_str(fpath))
>>> data = ut.load_cPkl(fpath)
>>> memtrack.report()
```

def makedata_complex(): data = np.empty((1000, 2 ** 20), dtype=np.uint8) data[:] = 0 return data

utool.util_io.**load_data**(*fpath*, ***kwargs*)
More generic interface to load data

utool.util_io.**load_hdf5**(*fpath*, *verbose=None*)

utool.util_io.**load_json**(*fpath*)

utool.util_io.**load_numpy**(*fpath*, *mmap_mode=None*, *verbose=None*)

utool.util_io.**load_pytables**(*fpath*, *verbose=False*)

utool.util_io.**load_text**(*fpath*, *verbose=None*, *aslines=False*, *strict=True*, *n=None*, *errors='replace'*)
Reads text from a file. Automatically returns utf8.

Parameters

- **fpath** (*str*) – file path
- **aslines** (*bool*) – if True returns list of lines
- **verbose** (*bool*) – verbosity flag

Returns text from fpath (this is unicode)

Return type str

Ignore: x = b"'/whaleshark_003_forsxc3xb8g.wmv" />rn"' ut.writeto('foo.txt', x) y = ut.readfrom('foo.txt') y.encode('utf8') == x

utool.util_io.**lock_and_load_cPkl**(*fpath*, *verbose=False*)

utool.util_io.**lock_and_save_cPkl**(*fpath*, *data*, *verbose=False*)

utool.util_io.**read_from**(*fpath*, *verbose=None*, *aslines=False*, *strict=True*, *n=None*, *errors='replace'*)
Reads text from a file. Automatically returns utf8.

Parameters

- **fpath** (*str*) – file path
- **aslines** (*bool*) – if True returns list of lines
- **verbose** (*bool*) – verbosity flag

Returns text from fpath (this is unicode)

Return type str

Ignore: x = b"'/whaleshark_003_forsxc3xb8g.wmv" />rn"' ut.writeto('foo.txt', x) y = ut.readfrom('foo.txt') y.encode('utf8') == x

utool.util_io.**read_lines_from**(*fpath*, *num_lines=None*, *verbose=None*, *n=None*)

utool.util_io.**readfrom**(*fpath*, *verbose=None*, *aslines=False*, *strict=True*, *n=None*, *errors='replace'*)
Reads text from a file. Automatically returns utf8.

Parameters

- **fpath** (*str*) – file path
- **aslines** (*bool*) – if True returns list of lines
- **verbose** (*bool*) – verbosity flag

Returns text from fpath (this is unicode)

Return type str

Ignore: x = b"'/whaleshark_003_forsxc3xb8g.wmv" />rn"' ut.writeto('foo.txt', x) y = ut.readfrom('foo.txt') y.encode('utf8') == x

utool.util_io.**save_cPkl**(*fpath*, *data*, *verbose=None*, *n=None*)
Saves data to a pickled file with optional verbosity

utool.util_io.**save_data**(*fpath*, *data*, ***kwargs*)
More generic interface to write data

utool.util_io.**save_hdf5**(*fpath*, *data*, *verbose=None*, *compression='lzf'*)
Restricted save of data using hdf5. Can only save ndarrays and dicts of ndarrays.

Parameters

- **fpath** (*str*) –
- **data** (*ndarray*) –
- **compression** (*str*) – DEFLATE/GZIP - standard LZF - fast SHUFFLE - compression ratio FLETCHER32 - error detection Scale-offset - integer / float scaling and truncation SZIP - fast and patented

CommandLine: python -m utool.util_io –test-save_hdf5

References

http://docs.h5py.org/en/latest/quick.html http://docs.h5py.org/en/latest/mpi.html

Ignore:

```
>>> # DISABLE_DOCTEST
>>> from utool.util_io import *  # NOQA
>>> import numpy as np
>>> import utool as ut
>>> rng = np.random.RandomState(0)
>>> data = (rng.rand(100000, 128) * 255).astype(np.uint8).copy()
>>> verbose = True
>>> fpath = ut.unixjoin(ut.ensure_app_resource_dir('utool'), 'myfile.hdf5')
>>> compression = 'lzf'
>>> ut.delete(fpath)
>>> save_hdf5(fpath, data, verbose, compression)
>>> data2 = load_hdf5(fpath, verbose)
>>> assert data is not data2
>>> assert np.all(data == data2)
>>> assert ut.delete(fpath)
```

Ignore:

```
>>> # DISABLE_DOCTEST
>>> from utool.util_io import *  # NOQA
>>> import numpy as np
>>> import utool as ut
>>> rng = np.random.RandomState(0)
>>> data = {'name': 'foobar', 'x': [1, 2, 3], 'y': np.array([3, 2, 1])}
>>> ut.exec_funckw(save_hdf5, globals())
>>> fpath = ut.unixjoin(ut.ensure_app_resource_dir('utool'), 'myfile2.hdf5')
>>> ut.delete(fpath)
>>> save_hdf5(fpath, data, verbose, compression)
>>> data2 = load_hdf5(fpath, verbose)
>>> assert data is not data2
>>> assert all([np.all(data[key] == data2[key]) for key in data.keys()])
>>> assert ut.delete(fpath)
```

Ignore:

```
>>> # DISABLE_DOCTEST
>>> # cPkl / numpy seems to be faster with this initial implementation
>>> import utool as ut
>>> data = (rng.rand(1000000, 128) * 255).astype(np.uint8).copy()
>>> print(ut.get_object_size_str(data))
>>> del data
>>> setup = ut.codeblock(
>>>     '''
        import numpy as np
        import utool as ut
        rng = np.random.RandomState(0)
        fpath = ut.unixjoin(ut.ensure_app_resource_dir('utool'), 'io_test_data
↪')
        data = (rng.rand(1000000, 128) * 255).astype(np.uint8).copy()
        #print(ut.get_object_size_str(data))
        ''')
>>> # Test load time
```

(continues on next page)

(continued from previous page)

```
>>> stmt_list1 = ut.codeblock(
>>>     '''
        ut.save_hdf5(fpath + '.hdf5', data, verbose=False, compression='gzip')
        ut.save_hdf5(fpath + '.hdf5', data, verbose=False, compression='lzf')
        ut.save_cPkl(fpath + '.cPkl', data, verbose=False)
        ut.save_numpy(fpath + '.npy', data, verbose=False)
        ut.save_pytables(fpath + '.tables', data, verbose=False)
        ''').split('\n')
>>> ut.util_dev.timeit_compare(stmt_list1, setup, int(10))
>>> # Test save time
>>> stmt_list2 = ut.codeblock(
>>>     '''
        ut.load_hdf5(fpath + '.hdf5', verbose=False)
        ut.load_cPkl(fpath + '.cPkl', verbose=False)
        ut.load_numpy(fpath + '.npy', verbose=False)
        ut.load_pytables(fpath + '.tables', verbose=False)
        ''').split('\n')
>>> ut.util_dev.timeit_compare(stmt_list2, setup, int(10))
>>> print('finished timeing')
+----------------
| TIMEIT COMPARE
+----------------
| iterations = 10
| Input:
|     | num | stmt
|     |   0 | u"ut.save_hdf5(fpath + '.hdf5', data, verbose=False,␣
↪compression='gzip')"
|     |   1 | u"ut.save_hdf5(fpath + '.hdf5', data, verbose=False,␣
↪compression='lzf')"
|     |   2 | u"ut.save_cPkl(fpath + '.cPkl', data, verbose=False)"
|     |   3 | u"ut.save_numpy(fpath + '.npy', data, verbose=False)"
|     |   4 | u"ut.save_pytables(fpath + '.tables', data, verbose=False)"
...
| Output:
|    * PASSED: each statement produced the same result
|    | num | total time | per loop | stmt
|    |   0 |    0.03 ks |   3.15 s | ut.save_hdf5(fpath + '.hdf5', data,␣
↪verbose=False, compression='gzip')
|    |   1 |    0.01 ks |   1.25 s | ut.save_hdf5(fpath + '.hdf5', data,␣
↪verbose=False, compression='lzf')
|    |   2 |     5.30 s |   0.53 s | ut.save_cPkl(fpath + '.cPkl', data,␣
↪verbose=False)
|    |   3 |     4.97 s |   0.50 s | ut.save_numpy(fpath + '.npy', data,␣
↪verbose=False)
|    |   4 |     9.23 s |   0.92 s | ut.save_pytables(fpath + '.tables', data,
↪ verbose=False)
L________________
+----------------
| TIMEIT COMPARE
+----------------
| iterations = 10
| Input:
|     | num | stmt
|     |   0 | u"ut.load_hdf5(fpath + '.hdf5', verbose=False)"
|     |   1 | u"ut.load_cPkl(fpath + '.cPkl', verbose=False)"
|     |   2 | u"ut.load_numpy(fpath + '.npy', verbose=False)"
|     |   3 | u"ut.load_pytables(fpath + '.tables', verbose=False)"
```

(continues on next page)

(continued from previous page)

```
...
| Output:
|     * PASSED: each statement produced the same result
|     | num | total time | per loop | stmt
|     |   0 |     2.39 s |   0.24 s | ut.load_hdf5(fpath + '.hdf5',␣
↪verbose=False)
|     |   1 |     0.39 s |   0.04 s | ut.load_cPkl(fpath + '.cPkl',␣
↪verbose=False)
|     |   2 |     0.19 s |   0.02 s | ut.load_numpy(fpath + '.npy',␣
↪verbose=False)
|     |   3 |     0.33 s |   0.03 s | ut.load_pytables(fpath + '.tables',␣
↪verbose=False)
L_____________________
```

Ignore: %timeit save_hdf5(fpath, data, verbose=False, compression='gzip') %timeit save_hdf5(fpath, data, verbose=False, compression='lzf') %timeit save_cPkl(fpath + '.cPkl', data, verbose=False) %timeit save_pytables(fpath + '.tables', data, verbose=False) 1 loops, best of 3: 258 ms per loop 10 loops, best of 3: 111 ms per loop 10 loops, best of 3: 53.1 ms per loop 10 loops, best of 3: 96.5 ms per loop

save_hdf5(fpath, data, verbose=False, compression='gzip') %timeit load_hdf5(fpath, verbose=False) save_hdf5(fpath, data, verbose=False, compression='lzf') %timeit load_hdf5(fpath, verbose=False) %timeit load_cPkl(fpath + '.cPkl', verbose=False) %timeit load_pytables(fpath + '.tables', verbose=False) 100 loops, best of 3: 19.4 ms per loop 100 loops, best of 3: 14.4 ms per loop 100 loops, best of 3: 3.92 ms per loop 100 loops, best of 3: 6.22 ms per loop

Notes

pip install mpi4py

utool.util_io.**save_json**(*fpath*, *data*, ***kwargs*)

utool.util_io.**save_numpy**(*fpath*, *data*, *verbose=None*, ***kwargs*)

utool.util_io.**save_pytables**(*fpath*, *data*, *verbose=False*)

sudo pip install numexpr sudo pip install tables

References

https://pytables.github.io/cookbook/py2exe_howto.html https://gist.github.com/andrewgiessel/7515520 http://stackoverflow.com/questions/8843062/python-how-to-store-a-numpy-multidimensional-array-in-pytables http://pytables.github.io/usersguide/tutorials.html#creating-new-array-objects

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_io import *  # NOQA
>>> import numpy as np
>>> import utool as ut
>>> # build test data
>>> verbose = True
>>> fpath = 'myfile.pytables.hdf5'
>>> np.random.seed(0)
>>> compression = 'gzip'
```

(continues on next page)

(continued from previous page)

```
>>> data = (np.random.rand(100000, 128) * 255).astype(np.uint8).copy()
>>> # execute function
>>> ut.delete(fpath)
>>> save_pytables(fpath, data, verbose)
>>> data2 = load_pytables(fpath, verbose)
>>> assert data is not data2
>>> assert np.all(data == data2)
>>> assert ut.delete(fpath)
```

utool.util_io.**save_text**(*fpath*, *to_write*, *aslines=False*, *verbose=None*, *onlyifdiff=False*, *mode='w'*, *n=None*)

Writes text to a file. Automatically encodes text as utf8.

Parameters

- **fpath** (*str*) – file path
- **to_write** (*str*) – text to write (must be unicode text)
- **aslines** (*bool*) – if True to_write is assumed to be a list of lines
- **verbose** (*bool*) – verbosity flag
- **onlyifdiff** (*bool*) – only writes if needed! checks hash of to_write vs the hash of the contents of fpath
- **mode** (*unicode*) – (default = u'w')
- **n** (*int*) – (default = 2)

CommandLine: python -m utool.util_io –exec-write_to –show

Ignore:

```
>>> # DISABLE_DOCTEST
>>> from utool.util_io import *  # NOQA
>>> import utool as ut
>>> fpath = ut.unixjoin(ut.get_app_resource_dir('utool'), 'testwrite.txt')
>>> ut.delete(fpath)
>>> to_write = 'utf-8 symbols Δ, , , , , , , , and .'
>>> aslines = False
>>> verbose = True
>>> onlyifdiff = False
>>> mode = u'w'
>>> n = 2
>>> write_to(fpath, to_write, aslines, verbose, onlyifdiff, mode, n)
>>> read_ = ut.read_from(fpath)
>>> print('read_    = ' + read_)
>>> print('to_write = ' + to_write)
>>> assert read_ == to_write
```

utool.util_io.**try_decode**(*x*)

utool.util_io.**write_to**(*fpath*, *to_write*, *aslines=False*, *verbose=None*, *onlyifdiff=False*, *mode='w'*, *n=None*)

Writes text to a file. Automatically encodes text as utf8.

Parameters

- **fpath** (*str*) – file path

- **to_write** (*str*) – text to write (must be unicode text)
- **aslines** (*bool*) – if True to_write is assumed to be a list of lines
- **verbose** (*bool*) – verbosity flag
- **onlyifdiff** (*bool*) – only writes if needed! checks hash of to_write vs the hash of the contents of fpath
- **mode** (*unicode*) – (default = u'w')
- **n** (*int*) – (default = 2)

CommandLine: python -m utool.util_io –exec-write_to –show

Ignore:

```
>>> # DISABLE_DOCTEST
>>> from utool.util_io import *  # NOQA
>>> import utool as ut
>>> fpath = ut.unixjoin(ut.get_app_resource_dir('utool'), 'testwrite.txt')
>>> ut.delete(fpath)
>>> to_write = 'utf-8 symbols Δ, , , , , , , , and .'
>>> aslines = False
>>> verbose = True
>>> onlyifdiff = False
>>> mode = u'w'
>>> n = 2
>>> write_to(fpath, to_write, aslines, verbose, onlyifdiff, mode, n)
>>> read_ = ut.read_from(fpath)
>>> print('read_    = ' + read_)
>>> print('to_write = ' + to_write)
>>> assert read_ == to_write
```

utool.util_io.**writeto**(*fpath*, *to_write*, *aslines=False*, *verbose=None*, *onlyifdiff=False*, *mode='w'*, *n=None*)

Writes text to a file. Automatically encodes text as utf8.

Parameters

- **fpath** (*str*) – file path
- **to_write** (*str*) – text to write (must be unicode text)
- **aslines** (*bool*) – if True to_write is assumed to be a list of lines
- **verbose** (*bool*) – verbosity flag
- **onlyifdiff** (*bool*) – only writes if needed! checks hash of to_write vs the hash of the contents of fpath
- **mode** (*unicode*) – (default = u'w')
- **n** (*int*) – (default = 2)

CommandLine: python -m utool.util_io –exec-write_to –show

Ignore:

```
>>> # DISABLE_DOCTEST
>>> from utool.util_io import *  # NOQA
>>> import utool as ut
```

(continues on next page)

(continued from previous page)

```
>>> fpath = ut.unixjoin(ut.get_app_resource_dir('utool'), 'testwrite.txt')
>>> ut.delete(fpath)
>>> to_write = 'utf-8 symbols Δ, , , , , , , , and .'
>>> aslines = False
>>> verbose = True
>>> onlyifdiff = False
>>> mode = u'w'
>>> n = 2
>>> write_to(fpath, to_write, aslines, verbose, onlyifdiff, mode, n)
>>> read_ = ut.read_from(fpath)
>>> print('read_    = ' + read_)
>>> print('to_write = ' + to_write)
>>> assert read_ == to_write
```

1.35 utool.util_ipynb module

Utilities for IPython/Jupyter Notebooks

CommandLine: # to connect to a notebook on a remote machine that does not have the # appropriate port exposed you must start an SSH tunnel. # Typically a jupyter-notebook runs on port 8888. # Run this command on your local machine. ssh -N -f -L localhost:8888:localhost:8889 <remote_user>@<remote_host>

class utool.util_ipynb.**IPYNBCell**(*header*, *code*, *footer*)

Bases: tuple

code

Alias for field number 1

footer

Alias for field number 2

header

Alias for field number 0

utool.util_ipynb.**code_cell**(*sourcecode*)

Parameters **sourcecode** (*str*) –

Returns json formatted ipython notebook code cell

Return type str

CommandLine: python -m wbia.templates.generate_notebook –exec-code_cell

Example

```
>>> # DISABLE_DOCTEST
>>> from wbia.templates.generate_notebook import *  # NOQA
>>> sourcecode = notebook_cells.timestamp_distribution[1]
>>> sourcecode = notebook_cells.initialize[1]
>>> result = code_cell(sourcecode)
>>> print(result)
```

utool.util_ipynb.**export_notebook**(*run_nb*, *fname*)

utool.util_ipynb.**format_cells**(*block*, *locals_*=*None*)

utool.util_ipynb.**make_autogen_str**()

Returns

Return type str

CommandLine: python -m utool.util_ipynb –exec-make_autogen_str –show

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_ipynb import *  # NOQA
>>> import utool as ut
>>> result = make_autogen_str()
>>> print(result)
```

utool.util_ipynb.**make_notebook**(*cell_list*)

References

Change cell width http://stackoverflow.com/questions/21971449/how-do-i-increase-the-cell-width-of-the-ipython-notebook-in-24207353#24207353

utool.util_ipynb.**markdown_cell**(*markdown*)

Parameters **markdown** (*str*) –

Returns json formatted ipython notebook markdown cell

Return type str

CommandLine: python -m wbia.templates.generate_notebook –exec-markdown_cell

Example

```
>>> # DISABLE_DOCTEST
>>> from wbia.templates.generate_notebook import *  # NOQA
>>> markdown = '# Title'
>>> result = markdown_cell(markdown)
>>> print(result)
```

utool.util_ipynb.**normalize_cells**(*block*)

utool.util_ipynb.**repr_single_for_md**(*s*)

Parameters **s** (*str*) –

Returns str_repr

Return type str

CommandLine: python -m wbia.templates.generate_notebook –exec-repr_single_for_md –show

Example

```
>>> # DISABLE_DOCTEST
>>> from wbia.templates.generate_notebook import *  # NOQA
>>> s = '#HTML(\'<iframe src="%s" width=700 height=350></iframe>\' % pdf_fpath)'
>>> result = repr_single_for_md(s)
>>> print(result)
```

utool.util_ipynb.**run_ipython_notebook**(*notebook_str*)

References

https://github.com/paulgb/runipy >>> from utool.util_ipynb import * # NOQA

1.36 utool.util_iter module

utool.util_iter.**and_iters**(**args*)

utool.util_iter.**ichunk_slices**(*total*, *chunksize*)

utool.util_iter.**ichunks**(*iterable*, *chunksize*, *bordermode=None*)
generates successive n-sized chunks from `iterable`.

Parameters

- **iterable** (*list*) – input to iterate over
- **chunksize** (*int*) – size of sublist to return
- **bordermode** (*str*) – None, ‘cycle’, or ‘replicate’

References

http://stackoverflow.com/questions/434287/iterate-over-a-list-in-chunks

SeeAlso: util_progress.get_num_chunks

CommandLine: python -m utool.util_iter –exec-ichunks –show

Timeit:

```
>>> import utool as ut
>>> setup = ut.codeblock('''
        from utool.util_iter import *  # NOQA
        iterable = list(range(100))
        chunksize = 8
        ''')
>>> stmt_list = [
...     'list(ichunks(iterable, chunksize))',
...     'list(ichunks_noborder(iterable, chunksize))',
...     'list(ichunks_list(iterable, chunksize))',
... ]
>>> (passed, times, results) = ut.timeit_compare(stmt_list, setup)
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_iter import *  # NOQA
>>> iterable = [1, 2, 3, 4, 5, 6, 7]
>>> chunksize = 3
>>> genresult = ichunks(iterable, chunksize)
>>> result = list(genresult)
>>> print(result)
[[1, 2, 3], [4, 5, 6], [7]]
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_iter import *  # NOQA
>>> iterable = (1, 2, 3, 4, 5, 6, 7)
>>> chunksize = 3
>>> bordermode = 'cycle'
>>> genresult = ichunks(iterable, chunksize, bordermode)
>>> result = list(genresult)
>>> print(result)
[[1, 2, 3], [4, 5, 6], [7, 1, 2]]
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_iter import *  # NOQA
>>> iterable = (1, 2, 3, 4, 5, 6, 7)
>>> chunksize = 3
>>> bordermode = 'replicate'
>>> genresult = ichunks(iterable, chunksize, bordermode)
>>> result = list(genresult)
>>> print(result)
[[1, 2, 3], [4, 5, 6], [7, 7, 7]]
```

utool.util_iter.**ichunks_cycle**(*iterable*, *chunksize*)

utool.util_iter.**ichunks_list**(*list_*, *chunksize*)
input must be a list.

SeeAlso: ichunks

References

http://stackoverflow.com/questions/434287/iterate-over-a-list-in-chunks

utool.util_iter.**ichunks_noborder**(*iterable*, *chunksize*)

utool.util_iter.**ichunks_replicate**(*iterable*, *chunksize*)

utool.util_iter.**ifilter_Nones**(*iter_*)
Removes any nones from the iterable

utool.util_iter.**ifilter_items**(*item_iter*, *flag_iter*)
iter_compress - like numpy compress

Parameters

- **item_iter** (*list*) –
- **flag_iter** (*list*) – of bools

Returns true_items

Return type list

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_iter import *  # NOQA
>>> item_iter = [1, 2, 3, 4, 5]
>>> flag_iter = [False, True, True, False, True]
>>> true_items = iter_compress(item_iter, flag_iter)
>>> result = list(true_items)
>>> print(result)
[2, 3, 5]
```

utool.util_iter.**ifilterfalse_items**(*item_iter*, *flag_iter*)

Parameters

- **item_iter** (*list*) –
- **flag_iter** (*list*) – of bools

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_iter import *  # NOQA
>>> item_iter = [1, 2, 3, 4, 5]
>>> flag_iter = [False, True, True, False, True]
>>> false_items = ifilterfalse_items(item_iter, flag_iter)
>>> result = list(false_items)
>>> print(result)
[1, 4]
```

utool.util_iter.**iflatten**(*list_*)
flattens a list iteratively

utool.util_iter.**iget_list_column**(*list_*, *colx*)
iterator version of get_list_column

utool.util_iter.**iget_list_column_slice**(*list_*, *start=None*, *stop=None*, *stride=None*)
iterator version of get_list_column

utool.util_iter.**interleave**(*args*)
zip followed by flatten

Parameters **args** (*tuple*) – tuple of lists to interleave

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_iter import *  # NOQA
>>> import utool as ut
>>> args = ([1, 2, 3, 4, 5], ['A', 'B', 'C', 'D', 'E', 'F', 'G'])
>>> genresult = interleave(args)
>>> result = ut.repr4(list(genresult), nl=False)
>>> print(result)
[1, 'A', 2, 'B', 3, 'C', 4, 'D', 5, 'E']
```

utool.util_iter.**itake_column**(*list_*, *colx*)
iterator version of get_list_column

utool.util_iter.**iter_compress**(*item_iter*, *flag_iter*)
iter_compress - like numpy compress

Parameters

- **item_iter** (*list*) –
- **flag_iter** (*list*) – of bools

Returns true_items

Return type list

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_iter import *  # NOQA
>>> item_iter = [1, 2, 3, 4, 5]
>>> flag_iter = [False, True, True, False, True]
>>> true_items = iter_compress(item_iter, flag_iter)
>>> result = list(true_items)
>>> print(result)
[2, 3, 5]
```

utool.util_iter.**iter_multichunks**(*iterable*, *chunksizes*, *bordermode=None*)

CommandLine: python -m utool.util_iter –test-iter_multichunks

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_iter import *  # NOQA
>>> import utool as ut
>>> iterable = list(range(20))
>>> chunksizes = (3, 2, 3)
>>> bordermode = 'cycle'
>>> genresult = iter_multichunks(iterable, chunksizes, bordermode)
>>> multichunks = list(genresult)
>>> depthprofile = ut.depth_profile(multichunks)
>>> assert depthprofile[1:] == chunksizes, 'did not generate chunks correctly'
>>> result = ut.repr4(list(map(str, multichunks)), nobr=True)
>>> print(result)
'[[[0, 1, 2], [3, 4, 5]], [[6, 7, 8], [9, 10, 11]], [[12, 13, 14], [15, 16, 17]]]
↪',
'[[[18, 19, 0], [1, 2, 3]], [[4, 5, 6], [7, 8, 9]], [[10, 11, 12], [13, 14, 15]]]
↪',
```

(continues on next page)

(continued from previous page)

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_iter import *  # NOQA
>>> import utool as ut
>>> iterable = list(range(7))
>>> # when chunksizes is len == 1, then equlivalent to ichunks
>>> chunksizes = (3,)
>>> bordermode = 'cycle'
>>> genresult = iter_multichunks(iterable, chunksizes, bordermode)
>>> multichunks = list(genresult)
>>> depthprofile = ut.depth_profile(multichunks)
>>> assert depthprofile[1:] == chunksizes, 'did not generate chunks correctly'
>>> result = str(multichunks)
>>> print(result)
[[0, 1, 2], [3, 4, 5], [6, 0, 1]]
```

utool.util_iter.**iter_window**(*iterable*, *size=2*, *step=1*, *wrap=False*)

iterates through iterable with a window size generalizeation of itertwo

Parameters

- **iterable** (*iter*) – an iterable sequence
- **size** (*int*) – window size (default = 2)
- **wrap** (*bool*) – wraparound (default = False)

Returns returns windows in a sequence

Return type iter

CommandLine: python -m utool.util_iter –exec-iter_window

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_iter import *  # NOQA
>>> iterable = [1, 2, 3, 4, 5, 6]
>>> size, step, wrap = 3, 1, True
>>> window_iter = iter_window(iterable, size, step, wrap)
>>> window_list = list(window_iter)
>>> result = ('window_list = %r' % (window_list,))
>>> print(result)
window_list = [(1, 2, 3), (2, 3, 4), (3, 4, 5), (4, 5, 6), (5, 6, 1), (6, 1, 2)]
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_iter import *  # NOQA
>>> iterable = [1, 2, 3, 4, 5, 6]
>>> size, step, wrap = 3, 2, True
```

(continues on next page)

(continued from previous page)

```
>>> window_iter = iter_window(iterable, size, step, wrap)
>>> window_list = list(window_iter)
>>> result = ('window_list = %r' % (window_list,))
>>> print(result)
window_list = [(1, 2, 3), (3, 4, 5), (5, 6, 1)]
```

utool.util_iter.**itertwo**(*iterable*, *wrap=False*)

equivalent to iter_window(iterable, 2, 1, wrap)

Parameters

- **iterable** (*iter*) – an iterable sequence
- **wrap** (*bool*) – if True, returns with wraparound

Returns returns edges in a sequence

Return type iter

CommandLine: python -m utool.util_iter –test-itertwo

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_iter import *  # NOQA
>>> iterable = [1, 2, 3, 4]
>>> wrap = False
>>> edges = list(itertwo(iterable, wrap))
>>> result = ('edges = %r' % (edges,))
>>> print(result)
edges = [(1, 2), (2, 3), (3, 4)]
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_iter import *  # NOQA
>>> iterable = [1, 2, 3, 4]
>>> wrap = True
>>> edges = list(itertwo(iterable, wrap))
>>> result = ('edges = %r' % (edges,))
>>> print(result)
edges = [(1, 2), (2, 3), (3, 4), (4, 1)]
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_iter import *  # NOQA
>>> import utool as ut
>>> iterable = iter([1, 2, 3, 4])
>>> wrap = False
>>> edge_iter = itertwo(iterable, wrap)
>>> edges = list(edge_iter)
>>> result = ('edges = %r' % (edges,))
```

(continues on next page)

(continued from previous page)

```
>>> ut.assert_eq(len(list(iterable)), 0, 'iterable should have been used up')
>>> print(result)
edges = [(1, 2), (2, 3), (3, 4)]
```

Ignore:

```
>>> # BENCHMARK
>>> import random
>>> import numpy as np
>>> rng = random.Random(0)
>>> iterable = [rng.randint(0, 2000) for _ in range(100000)]
>>> iterable2 = np.array(iterable)
>>> #
>>> import ubelt as ub
>>> ti = ub.Timerit(100, bestof=10, verbose=2)
>>> #
>>> for timer in ti.reset('list-zip'):
>>>     with timer:
>>>         list(zip(iterable, iterable[1:]))
>>> #
>>> for timer in ti.reset('list-itertwo'):
>>>     with timer:
>>>         list(itertwo(iterable))
>>> #
>>> for timer in ti.reset('iter_window(2)'):
>>>     with timer:
>>>         list(ub.iter_window(iterable, 2))
>>> #
>>> for timer in ti.reset('list-zip-numpy'):
>>>     with timer:
>>>         list(zip(iterable2, iterable2[1:]))
>>> #
>>> for timer in ti.reset('list-zip-numpy.tolist'):
>>>     with timer:
>>>         list(zip(iterable2.tolist(), iterable2.tolist()[1:]))
>>> #
>>> for timer in ti.reset('list-itertwo-numpy'):
>>>     with timer:
>>>         list(itertwo(iterable2))
>>> #
>>> for timer in ti.reset('list-itertwo-numpy-tolist'):
>>>     with timer:
>>>         list(itertwo(iterable2.tolist()))
>>> #
>>> print(ub.repr2(ti.rankings))
>>> print(ub.repr2(ti.consistency))
```

utool.util_iter.**next_counter**(*start=0*, *step=1*)

Parameters

- **start** (*int*) – (default = 0)
- **step** (*int*) – (default = 1)

Returns func

CommandLine: python -m utool.util_iter –test-next_counter

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_iter import *  # NOQA
>>> start = 1
>>> step = 1
>>> next_ = next_counter(start, step)
>>> result = str([next_(), next_(), next_()])
>>> print(result)
[1, 2, 3]
```

utool.util_iter.**random_combinations**(*items*, *size*, *num=None*, *rng=None*)

Yields *num* combinations of length *size* from items in random order

Parameters

- **items** –
- **size** –
- **num** (*None*) – (default = None)
- **rng** (*RandomState*) – random number generator(default = None)

Yields *tuple* – combo

CommandLine: python -m utool.util_iter random_combinations

Ignore:

```
>>> # ENABLE_DOCTEST
>>> from utool.util_iter import *  # NOQA
>>> import utool as ut
>>> items = list(range(10))
>>> size = 3
>>> num = 5
>>> rng = 0
>>> combos = list(random_combinations(items, size, num, rng))
>>> result = ('combos = %s' % (ut.repr2(combos),))
>>> print(result)
```

Ignore:

```
>>> # ENABLE_DOCTEST
>>> from utool.util_iter import *  # NOQA
>>> import utool as ut
>>> items = list(zip(range(10), range(10)))
>>> size = 3
>>> num = 5
>>> rng = 0
>>> combos = list(random_combinations(items, size, num, rng))
>>> result = ('combos = %s' % (ut.repr2(combos),))
>>> print(result)
```

utool.util_iter.**random_product**(*items*, *num=None*, *rng=None*)

Yields *num* items from the cartesian product of items in a random order.

Parameters **items** (*list of sequences*) – items to get caresian product of packed in a list or tuple. (note this deviates from api of it.product)

Example

import utool as ut items = [(1, 2, 3), (4, 5, 6, 7)] rng = 0 list(ut.random_product(items, rng=0)) list(ut.random_product(items, num=3, rng=0))

utool.util_iter.**wrap_iterable**(*obj*)

Returns wrapped_obj, was_scalar

1.37 utool.util_latex module

TODO: box and whisker http://tex.stackexchange.com/questions/115210/boxplot-in-latex

utool.util_latex.**compile_latex_text**(*input_text*, *dpath=None*, *fname=None*, *verbose=True*, *move=True*, *nest_in_doc=None*, *title=None*, *preamb_extra=None*)

CommandLine: python -m utool.util_latex –test-compile_latex_text –show

Ignore:

pdflatex -shell-escape –synctex=-1 -src-specials -interaction=nonstopmode ~/code/ibeis/tmptex/latex_formatter_temp.tex

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_latex import *  # NOQA
>>> import utool as ut
>>> verbose = True
>>> #dpath = '/home/joncrall/code/ibeis/aidchallenge'
>>> dpath = dirname(ut.grab_test_imgpath())
>>> #ut.vd(dpath)
>>> orig_fpaths = ut.list_images(dpath, fullpath=True)
>>> figure_str = ut.get_latex_figure_str(orig_fpaths, width_str='2.4in', nCols=2)
>>> input_text = figure_str
>>> pdf_fpath = ut.compile_latex_text(input_text, dpath=dpath,
>>>                                   verbose=verbose)
>>> output_pdf_fpath = ut.compress_pdf(pdf_fpath)
>>> print(pdf_fpath)
>>> ut.quit_if_noshow()
>>> ut.startfile(pdf_fpath)
```

utool.util_latex.**compress_pdf**(*pdf_fpath*, *output_fname=None*)
uses ghostscript to write a pdf

utool.util_latex.**convert_pdf_to_image**(*pdf_fpath*, *ext='.jpg'*, *verbose=1*, *dpi=300*, *quality=90*)

utool.util_latex.**ensure_colvec**(*arr*)

utool.util_latex.**ensure_rowvec**(*arr*)

utool.util_latex.**escape_latex**(*text*)

Parameters **text** (*str*) – a plain text message

Returns the message escaped to appear correctly in LaTeX

Return type str

References

http://stackoverflow.com/questions/16259923/how-can-i-escape-characters

utool.util_latex.**find_ghostscript_exe**()

utool.util_latex.**get_latex_figure_str**(*fpath_list*, *caption_str=None*, *label_str=None*, *width_str='\\textwidth'*, *height_str=None*, *nCols=None*, *dpath=None*, *colpos_sep=' '*, *nlsep=''*, *use_sublbls=None*, *use_frame=False*)

Parameters

- **fpath_list** (*list*) –
- **dpath** (*str*) – directory relative to main tex file

Returns figure_str

Return type str

CommandLine: python -m utool.util_latex –test-get_latex_figure_str

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_latex import *  # NOQA
>>> fpath_list = ['figures/foo.png']
>>> figure_str = get_latex_figure_str(fpath_list)
>>> result = str(figure_str)
>>> print(result)
```

utool.util_latex.**get_latex_figure_str2**(*fpath_list*, *cmdname*, ***kwargs*)
hack for candidacy

utool.util_latex.**is_substr**(*find*, *strlist*)

utool.util_latex.**latex_get_stats**(*lbl*, *data*, *mode=0*)

utool.util_latex.**latex_multicolumn**(*data*, *ncol=2*, *alignstr='|c|'*)

utool.util_latex.**latex_multirow**(*data*, *nrow=2*)

utool.util_latex.**latex_newcommand**(*command_name*, *command_text*, *num_args=0*)

utool.util_latex.**latex_sanitize_command_name**(*_cmdname*)

Parameters **_cmdname** –

Returns command_name

Return type

?

CommandLine: python -m utool.util_latex –exec-latex_sanitize_command_name

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_latex import *  # NOQA
>>> _cmdname = '#foo bar.'
>>> command_name = latex_sanitize_command_name(_cmdname)
>>> result = ('command_name = %s' % (str(command_name),))
>>> print(result)
FooBar
```

utool.util_latex.**latex_scalar**(*lbl*, *data*)

utool.util_latex.**long_substr**(*strlist*)

utool.util_latex.**make_full_document**(*text*, *title=None*, *preamp_decl={}*, *preamb_extra=None*)

dummy preamble and document to wrap around latex fragment

Parameters

- **text** (*str*) –
- **title** (*str*) –

Returns str

CommandLine: python -m utool.util_latex –test-make_full_document

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_latex import *  # NOQA
>>> text = 'foo'
>>> title = 'title'
>>> preamp_decl = {}
>>> text_ = make_full_document(text, title)
>>> result = str(text_)
>>> print(result)
```

utool.util_latex.**make_score_tabular**(*row_lbls*, *col_lbls*, *values*, *title=None*, *out_of=None*, *bold_best=False*, *flip=False*, *bigger_is_better=True*, *multicol_lbls=None*, *FORCE_INT=False*, *precision=None*, *SHORTEN_ROW_LBLS=False*, *col_align='l'*, *col_sep='|'*, *multicol_sep='|'*, *centerline=True*, *astable=False*, *table_position=''*, *AUTOFIX_LATEX=True*, ***kwargs*)

makes a LaTeX tabular for displaying scores or errors

Parameters

- **row_lbls** (*list of str*) –
- **col_lbls** (*list of str*) –
- **values** (*ndarray*) –
- **title** (*str*) – (default = None)
- **out_of** (*None*) – (default = None)
- **bold_best** (*bool*) – (default = True)

- **flip** (*bool*) – (default = False)
- **table_position** (*str*) – eg '[h]'

Returns tabular_str

Return type str

CommandLine: python -m utool.util_latex –test-make_score_tabular:0 –show python -m utool.util_latex –test-make_score_tabular:1 –show python -m utool.util_latex –test-make_score_tabular:2 –show

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_latex import *  # NOQA
>>> import utool as ut
>>> row_lbls = ['config1', 'config2']
>>> col_lbls = ['score \\leq 1', 'metric2']
>>> values = np.array([[1.2, 2], [3.2, 4]])
>>> title = 'title'
>>> out_of = 10
>>> bold_best = True
>>> flip = False
>>> tabular_str = make_score_tabular(row_lbls, col_lbls, values, title, out_of,␣
↪bold_best, flip)
>>> result = tabular_str
>>> print(result)
>>> ut.quit_if_noshow()
>>> render_latex_text(tabular_str)
```

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_latex import *  # NOQA
>>> import utool as ut
>>> row_lbls = ['config1']
>>> col_lbls = ['score \\leq 1', 'metric2']
>>> values = np.array([[1.2, 2]])
>>> title = 'title'
>>> out_of = 10
>>> bold_best = True
>>> flip = False
>>> tabular_str = make_score_tabular(row_lbls, col_lbls, values, title, out_of,␣
↪bold_best, flip)
>>> result = tabular_str
>>> print(result)
>>> ut.quit_if_noshow()
>>> render_latex_text(tabular_str)
```

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_latex import *  # NOQA
>>> import utool as ut
```

(continues on next page)

(continued from previous page)

```
>>> row_lbls = ['config1', 'config2']
>>> col_lbls = ['score \\leq 1', 'metric2', 'foobar']
>>> multicol_lbls = [('spam', 1), ('eggs', 2)]
>>> values = np.array([[1.2, 2, -3], [3.2, 4, -2]])
>>> title = 'title'
>>> out_of = 10
>>> bold_best = True
>>> flip = False
>>> tabular_str = make_score_tabular(row_lbls, col_lbls, values, title, out_of,␣
↪bold_best, flip, multicol_lbls=multicol_lbls)
>>> result = tabular_str
>>> print(result)
>>> ut.quit_if_noshow()
>>> render_latex_text(tabular_str)
```

utool.util_latex.**make_stats_tabular**()
tabular for dipslaying statistics

utool.util_latex.**render_latex**(*input_text*, *dpath=None*, *fname=None*, *preamb_extra=None*, *verbose=1*, ***kwargs*)
Renders latex text into a jpeg.

Whitespace that would have appeared in the PDF is removed, so the jpeg is cropped only the the relevant part. This is ideal for figures that only take a single page.

Parameters

- **input_text** –
- **dpath** (*str*) – directory path(default = None)
- **fname** (*str*) – file name(default = None)
- **preamb_extra** (*None*) – (default = None)
- **verbose** (*int*) – verbosity flag(default = 1)

Returns jpg_fpath - file path string

Return type str

CommandLine: python -m utool.util_latex render_latex '$O(n^2)$' –fpath=~/slides/tmp.jpg

Script:

```
>>> # SCRIPT
>>> from utool.util_latex import *  # NOQA
>>> from os.path import split, expanduser
>>> import utool as ut
>>> input_text = ' '.join(ut.get_varargs()[1:])
>>> dpath, fname = split(ut.argval('--fpath', ''))
>>> dpath = expanduser(ut.argval('--dpath', dpath))
>>> fname = ut.argval('--fname', fname)
>>> kwargs = ut.dict_subset(ut.argparse_funckw(ut.convert_pdf_to_image), ['dpi
↪', 'quality'])
>>> jpg_fpath = render_latex(input_text, dpath, fname, **kwargs)
>>> if ut.argflag('--diskshow'):
>>>     ut.startfile(jpg_fpath)
```

utool.util_latex.**render_latex_text**(*input_text*, *nest_in_doc=False*, *preamb_extra=None*, *appname='utool'*, *verbose=None*)
compiles latex and shows the result

utool.util_latex.**replace_all**(*str_*, *repltups*)

utool.util_latex.**tabular_join**(*tabular_body_list*, *nCols=2*)

1.38 utool.util_list module

utool.util_list.**accumulate**(*iterator*)

Notice: use itertools.accumulate in python > 3.2

utool.util_list.**alloc_lists**(*num_alloc*)
allocates space for a `list` of lists

utool.util_list.**alloc_nones**(*num_alloc*)
allocates space for a `list` of Nones

utool.util_list.**allsame**(*list_*, *strict=True*)
checks to see if list is equal everywhere

Parameters **list** (*list*) –

Returns True if all items in the list are equal

utool.util_list.**and_lists**(**args*)

utool.util_list.**argmax**(*input_*, *multi=False*)
Returns index / key of the item with the largest value.

Parameters **input_** (*dict or list*) –

References

http://stackoverflow.com/questions/16945518/python-argmin-argmax

Ignore: list_ = np.random.rand(10000).tolist() %timeit list_.index(max(list_)) %timeit max(enumerate(list_), key=operator.itemgetter(1))[0] %timeit max(enumerate(list_), key=lambda x: x[1])[0] %timeit max(range(len(list_)), key=list_.__getitem__)

input_ = dict_ list_ = np.random.rand(100000).tolist() dict_ = {str(ut.random_uuid()): x for x in list_} %timeit list(input_.keys())[ut.argmax(list(input_.values()))] %timeit max(input_.items(), key=operator.itemgetter(1))[0]

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_list import *
>>> import utool as ut
>>> input_ = [1, 2, 3, 3, 2, 3, 2, 1]
>>> ut.argmax(input_, multi=True)
>>> input_ = {1: 4, 2: 2, 3: 3, 3: 4}
>>> ut.argmax(input_, multi=True)
```

utool.util_list.**argmin**(*input_*, *key=None*)
Returns index / key of the item with the smallest value.

Parameters **input** (*dict or list*) –

Note: a[argmin(a, key=key)] == min(a, key=key)

---

utool.util_list.**argsort**(**args*, ***kwargs*)

like np.argsort but for lists

Parameters

- ***args** – multiple lists to sort by
- ****kwargs** – reverse (bool): sort order is descending if True else acscending

CommandLine: python -m utool.util_list argsort

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> result = ut.argsort({'a': 3, 'b': 2, 'c': 100})
>>> print(result)
```

utool.util_list.**argsort2**(*indexable*, *key=None*, *reverse=False*)

Returns the indices that would sort a indexable object.

This is similar to np.argsort, but it is written in pure python and works on both lists and dictionaries.

Parameters **indexable** (*list or dict*) – indexable to sort by

Returns indices: list of indices such that sorts the indexable

Return type list

Example

```
>>> # DISABLE_DOCTEST
>>> import utool as ut
>>> # argsort works on dicts
>>> dict_ = indexable = {'a': 3, 'b': 2, 'c': 100}
>>> indices = ut.argsort2(indexable)
>>> assert list(ut.take(dict_, indices)) == sorted(dict_.values())
>>> # argsort works on lists
>>> indexable = [100, 2, 432, 10]
>>> indices = ut.argsort2(indexable)
>>> assert list(ut.take(indexable, indices)) == sorted(indexable)
>>> # argsort works on iterators
>>> indexable = reversed(range(100))
>>> indices = ut.argsort2(indexable)
>>> assert indices[0] == 99
```

utool.util_list.**aslist**(*sequence*)

Ensures that the sequence object is a Python list. Handles, numpy arrays, and python sequences (e.g. tuples, and iterables).

Parameters **sequence** (*sequence*) – a list-like object

Returns *sequence* as a Python list

Return type list

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> s1 = [1, 2, 3]
>>> s2 = (1, 2, 3)
>>> assert aslist(s1) is s1
>>> assert aslist(s2) is not s2
>>> aslist(np.array([[1, 2], [3, 4], [5, 6]]))
[[1, 2], [3, 4], [5, 6]]
>>> aslist(range(3))
[0, 1, 2]
```

utool.util_list.**broadcast_zip**(*list1*, *list2*)

Zips elementwise pairs between list1 and list2. Broadcasts the first dimension if a single list is of length 1.

Aliased as bzip

Parameters

- **list1** (*list*) –
- **list2** (*list*) –

Returns list of pairs

Return type list

SeeAlso: util_dict.dzip

Raises ValueError – if the list dimensions are not broadcastable

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> import utool as ut
>>> assert list(bzip([1, 2, 3], [4])) == [(1, 4), (2, 4), (3, 4)]
>>> assert list(bzip([1, 2, 3], [4, 4, 4])) == [(1, 4), (2, 4), (3, 4)]
>>> assert list(bzip([1], [4, 4, 4])) == [(1, 4), (1, 4), (1, 4)]
>>> ut.assert_raises(ValueError, bzip, [1, 2, 3], [])
>>> ut.assert_raises(ValueError, bzip, [], [4, 5, 6])
>>> ut.assert_raises(ValueError, bzip, [], [4])
>>> ut.assert_raises(ValueError, bzip, [1, 2], [4, 5, 6])
>>> ut.assert_raises(ValueError, bzip, [1, 2, 3], [4, 5])
```

utool.util_list.**bzip**(*list1*, *list2*)

Zips elementwise pairs between list1 and list2. Broadcasts the first dimension if a single list is of length 1.

Aliased as bzip

Parameters

- **list1** (*list*) –
- **list2** (*list*) –

Returns list of pairs

Return type list

SeeAlso: util_dict.dzip

Raises `ValueError` – if the list dimensions are not broadcastable

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> import utool as ut
>>> assert list(bzip([1, 2, 3], [4])) == [(1, 4), (2, 4), (3, 4)]
>>> assert list(bzip([1, 2, 3], [4, 4, 4])) == [(1, 4), (2, 4), (3, 4)]
>>> assert list(bzip([1], [4, 4, 4])) == [(1, 4), (1, 4), (1, 4)]
>>> ut.assert_raises(ValueError, bzip, [1, 2, 3], [])
>>> ut.assert_raises(ValueError, bzip, [], [4, 5, 6])
>>> ut.assert_raises(ValueError, bzip, [], [4])
>>> ut.assert_raises(ValueError, bzip, [1, 2], [4, 5, 6])
>>> ut.assert_raises(ValueError, bzip, [1, 2, 3], [4, 5])
```

utool.util_list.**compress**(*item_list*, *flag_list*)

like np.compress but for lists

Returns items in item list where the corresponding item in flag list is True

Parameters

- **item_list** (*list*) – list of items to mask
- **flag_list** (*list*) – list of booleans used as a mask

Returns filtered_items - masked items

Return type list

utool.util_list.**debug_consec_list**(*list_*)

Returns tuple of (missing_items, missing_indices, duplicate_items)

utool.util_list.**debug_duplicate_items**(*items*, **args*, ***kwargs*)

utool.util_list.**delete_items_by_index**(*list_*, *index_list*, *copy=False*)

Remove items from `list_` at positions specified in `index_list` The original `list_` is preserved if `copy` is True

Parameters

- **list** (*list*) –
- **index_list** (*list*) –
- **copy** (*bool*) – preserves original list if True

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> list_ = [8, 1, 8, 1, 6, 6, 3, 4, 4, 5, 6]
>>> index_list = [2, -1]
>>> result = delete_items_by_index(list_, index_list)
>>> print(result)
[8, 1, 1, 6, 6, 3, 4, 4, 5]
```

utool.util_list.**delete_list_items**(*list_*, *item_list*, *copy=False*)
Remove items in `item_list` from `list_`. The original `list_` is preserved if `copy` is True

utool.util_list.**depth**(*sequence*, *func=<built-in function max>*, *_depth=0*)
Find the nesting depth of a nested sequence

utool.util_list.**depth_profile**(*list_*, *max_depth=None*, *compress_homogenous=True*, *compress_consecutive=False*, *new_depth=False*)
Returns a nested list corresponding the shape of the nested structures lists represent depth, tuples represent shape. The values of the items do not matter. only the lengths.

Parameters

- **list** (*list*) –
- **max_depth** (*None*) –
- **compress_homogenous** (*bool*) –
- **compress_consecutive** (*bool*) – experimental

CommandLine: python -m utool.util_list –test-depth_profile

Setup:

```
>>> from utool.util_list import *  # NOQA
```

Example

```
>>> # ENABLE_DOCTEST
>>> list_ = [[[1, 1, 1, 1], [1, 1, 1, 1], [1, 1, 1, 1]], [[1, 1, 1, 1], [1, 1, 1,␣
↪1], [1, 1, 1, 1]]]
>>> result = depth_profile(list_)
>>> print(result)
(2, 3, 4)
```

Example

```
>>> # ENABLE_DOCTEST
>>> list_ = [[[[[1]]]], [3, 4, 33]], [[1], [2, 3], [4, [5, 5]]], [1, 3]]
>>> result = depth_profile(list_)
>>> print(result)
[[(1, 1, 1), 3], [1, 2, [1, 2]], 2]
```

Example

```
>>> # ENABLE_DOCTEST
>>> list_ = [[[[[1]]]], [3, 4, 33]], [[1], [2, 3], [4, [5, 5]]], [1, 3]]
>>> result = depth_profile(list_, max_depth=1)
>>> print(result)
[[(1, '1'), 3], [1, 2, [1, '2']], 2]
```

Example

```
>>> # ENABLE_DOCTEST
>>> list_ = [[[1, 2], [1, 2, 3]], None]
>>> result = depth_profile(list_, compress_homogenous=True)
>>> print(result)
[[2, 3], 1]
```

Example

```
>>> # ENABLE_DOCTEST
>>> list_ = [[3, 2], [3, 2], [3, 2], [3, 2], [3, 2], [3, 2], [9, 5, 3], [2, 2]]
>>> result = depth_profile(list_, compress_homogenous=True, compress_
↪consecutive=True)
>>> print(result)
[2] * 6 + [3, 2]
```

Example

```
>>> # ENABLE_DOCTEST
>>> list_ = [[[3, 9], 2], [[3, 9], 2], [[3, 9], 2], [[3, 9], 2]]  #, [3, 2], [3,␣
↪2]]
>>> result = depth_profile(list_, compress_homogenous=True, compress_
↪consecutive=True)
>>> print(result)
(4, [2, 1])
```

Example

```
>>> # ENABLE_DOCTEST
>>> list_ = [[[[1, 2]], [1, 2]], [[[1, 2]], [1, 2]], [[[0, 2]], [1]]]
>>> result1 = depth_profile(list_, compress_homogenous=True, compress_
↪consecutive=False)
>>> result2 = depth_profile(list_, compress_homogenous=True, compress_
↪consecutive=True)
>>> result = str(result1) + '\n' + str(result2)
>>> print(result)
[[(1, 2), 2], [(1, 2), 2], [(1, 2), 1]]
[[(1, 2), 2]] * 2 + [[(1, 2), 1]]
```

Example

```
>>> # ENABLE_DOCTEST
>>> list_ = [[{'a': [1, 2], 'b': [3, 4, 5]}, [1, 2, 3]], None]
>>> result = depth_profile(list_, compress_homogenous=True)
>>> print(result)
```

Example

```
>>> # ENABLE_DOCTEST
>>> list_ = [[[1]], [[[1, 1], [1, 1]]], [[[[1, 3], 1], [[1, 3, 3], 1, 1]]]]
>>> result = depth_profile(list_, compress_homogenous=True)
>>> print(result)
```

Example

```
>>> # ENABLE_DOCTEST
>>> list_ = []
>>> result = depth_profile(list_)
>>> print(result)
```

THIS IS AN ERROR??? SHOULD BE #[1, 1], [1, 2, 2], (1, ([1, 2]), (

Example

```
>>> # ENABLE_DOCTEST
>>> fm1 = [[0, 0], [0, 0]]
>>> fm2 = [[0, 0], [0, 0], [0, 0]]
>>> fm3 = [[0, 0], [0, 0], [0, 0], [0, 0]]
>>> list_ = [0, 0, 0]
>>> list_ = [fm1, fm2, fm3]
>>> max_depth = 0
>>> new_depth = True
>>> result = depth_profile(list_, max_depth=max_depth, new_depth=new_depth)
>>> print(result)
```

utool.util_list.**duplicates_exist**(*items*)
returns if list has duplicates

utool.util_list.**emap**(*func*, *iter_*, ***kwargs*)
Eager version of the builtin map function. This provides the same functionality as python2 map.

Note this is inefficient and should only be used when prototyping and debugging.

Extended functionality supports passing common kwargs to all functions

utool.util_list.**ensure_list_size**(*list_*, *size_*)
Allocates more space if needbe.

Ensures len(list_) == size_.

Parameters

- **list** (*list*) – list to extend

- **size** (*int*) – amount to exent by

utool.util_list.**equal**(*list1*, *list2*)
takes flags returns indexes of True values

utool.util_list.**estarmap**(*func*, *iter_*, ***kwargs*)
Eager version of it.starmap from itertools

Note this is inefficient and should only be used when prototyping and debugging.

utool.util_list.**ezip**(**args*)
Eager version of the builtin zip function. This provides the same functionality as python2 zip.

Note this is inefficient and should only be used when prototyping and debugging.

utool.util_list.**filter_Nones**(*item_list*)
Removes any nones from the list

Parameters **item_list** (*list*) –

Returns sublist which does not contain Nones

utool.util_list.**filter_items**(*item_list*, *flag_list*)
Returns items in item list where the corresponding item in flag list is True

Parameters

- **item_list** (*list*) –
- **flag_list** (*list*) –

Returns filtered_items

SeeAlso: util_iter.iter_compress

utool.util_list.**filter_startswith**(*list_*, *str_*)

utool.util_list.**filterfalse_items**(*item_list*, *flag_list*)
Returns items in item list where the corresponding item in flag list is true

Parameters

- **item_list** (*list*) – list of items
- **flag_list** (*list*) – list of truthy values

Returns items where the corresponding flag was truthy

Return type filtered_items

SeeAlso: util_iter.ifilterfalse_items

utool.util_list.**find_duplicate_items**(*items*, *k=2*)

Parameters **items** (*list*) –

Returns duplicate_map of indexes

Return type dict

CommandLine: python -m utool.util_list –test-find_duplicate_items

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> items = [0, 1, 2, 3, 3, 0, 12, 2, 9]
>>> duplicate_map = find_duplicate_items(items)
>>> result = str(duplicate_map)
>>> print(result)
```

utool.util_list.**find_list_indexes**(*list_*, *items*)

Parameters

- **list** (*list*) – list of items to be searched
- **items** (*list*) – list of items to find

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> list_ = ['a', 'b', 'c']
>>> items = ['d', 'c', 'b', 'f']
>>> index_list = find_list_indexes(list_, items)
>>> result = ('index_list = %r' % (index_list,))
>>> print(result)
index_list = [None, 2, 1, None]
```

utool.util_list.**find_nonconsec_values**(*values*, *min_=None*, *max_=None*)

Determines if a list of values is consecutive (ascending)

Parameters

- **values** (*list*) – list of values, sorted and unique
- **min** (*int*) – minimum value in range defaults min(values)
- **max** (*int*) – maximum value in range defaults max(values)

Returns missing values that would make the list consecutive

Return type missing_values

CommandLine: python -m utool.util_list –test-find_nonconsec_values

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> import numpy as np
>>> values = np.array([-2, 1,  2, 10])
>>> result = find_nonconsec_values(values)
>>> print(result)
[-1, 0, 3, 4, 5, 6, 7, 8, 9]
```

utool.util_list.**flag_None_items**(*list_*)

utool.util_list.**flag_not_None_items**(*list_*)

utool.util_list.**flag_percentile_parts**(*arr*, *front=None*, *mid=None*, *back=None*)

utool.util_list.**flag_unique_items**(*list_*)

Returns a list of flags corresponding to the first time an item is seen

Parameters **list_** (*list*) – list of items

Returns flag_list

Timing:

```
>>> import random
>>> import utool as ut
>>>
>>> def random_items(n, m):
>>>     rng = random.Random(0)
>>>     return [rng.randint(0, n) for _ in range(m)]
>>>
>>> m = 1000
>>>
>>> def method1(list\_):
>>>     seen = set()
>>>     def unseen(item):
>>>         if item in seen:
>>>             return False
>>>         seen.add(item)
>>>         return True
>>>     flag_list = [unseen(item) for item in list\_]
>>>     return flag_list
>>>
>>> def method2(list\_):
>>>     return ut.index_to_boolmask([list\_.index(x) for x in set(list\_)],␣
↪len(list\_))
>>>
>>> def method3(list\_):
>>>     return ut.index_to_boolmask(dict(zip(reversed(list\_),␣
↪reversed(range(len(list\_))))).values(), len(list\_))
>>>
>>>
>>> import ubelt as ub
>>> ub.Timerit.DEFAULT_VERBOSE = False
>>>
>>> ut.qtensure()
>>> exps = [0, .25, .5, .75, 1, 2]
>>> pnum = pt.make_pnum_nextgen(nSubplots=len(exps))
>>> current = ut.flag_unique_items
>>>
>>> for count, exp in ut.ProgIter(list(enumerate(exps, start=1))):
>>>     ydatas = ut.ddict(list)
>>>     xdata = []
>>>     for m in ut.ProgIter(list(range(0, 10000, 100)), freq=1):
>>>         xdata.append(m)
>>>         num = 10
>>>         n = int(m ** exp)
>>>         list\_ = random_items(n=n, m=m)
>>>         ydatas['method1'].append(ub.Timerit(num).call(method1, list\_))
>>>         ydatas['method2'].append(ub.Timerit(num).call(method2, list\_))
>>>         ydatas['method3'].append(ub.Timerit(num).call(method3, list\_))
>>>         ydatas['current'].append(ub.Timerit(num).call(current, list\_))
```

(continues on next page)

(continued from previous page)

```
>>>
>>>         # assert method1(list\_) == method3(list\_)
>>>         # assert method1(list\_) == current(list\_)
>>>
>>>     pt.multi_plot(
>>>         xdata, list(ydatas.values()), label_list=list(ydatas.keys()),
>>>         ylabel='time', title=str(exp), fnum=1, pnum=pnum())
```

utool.util_list.**flat_unique**(**lists_*, ***kwargs*)
returns items unique across all lists

utool.util_list.**flatten**(*list_*)

Parameters **list** (*list*) – list of lists

Returns flat list

Return type list

CommandLine: python -m utool.util_list –test-flatten

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> import utool as ut
>>> list_ = [['a', 'b'], ['c', 'd']]
>>> unflat_list2 = flatten(list_)
>>> result = ut.repr4(unflat_list2, nl=False)
>>> print(result)
['a', 'b', 'c', 'd']
```

utool.util_list.**get_dirty_items**(*item_list*, *flag_list*)
Returns each item in item_list where not flag in flag_list

Parameters

- **item_list** (*list*) –
- **flag_list** (*list*) –

Returns dirty_items

utool.util_list.**get_list_column**(*list_*, *colx*)
accepts a list of (indexables) and returns a list of indexables can also return a list of list of indexables if colx is a list

Parameters

- **list** (*list*) – list of lists
- **colx** (*int or list*) – index or key in each sublist get item

Returns list of selected items

Return type list

CommandLine: python -m utool.util_list –test-take_column

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> list_ = [['a', 'b'], ['c', 'd']]
>>> colx = 0
>>> result = take_column(list_, colx)
>>> import utool as ut
>>> result = ut.repr4(result, nl=False)
>>> print(result)
['a', 'c']
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> list_ = [['a', 'b'], ['c', 'd']]
>>> colx = [1, 0]
>>> result = take_column(list_, colx)
>>> import utool as ut
>>> result = ut.repr4(result, nl=False)
>>> print(result)
[['b', 'a'], ['d', 'c']]
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> list_ = [{'spam': 'EGGS', 'ham': 'SPAM'}, {'spam': 'JAM', 'ham': 'PRAM'}]
>>> # colx can be a key or list of keys as well
>>> colx = ['spam']
>>> result = take_column(list_, colx)
>>> import utool as ut
>>> result = ut.repr4(result, nl=False)
>>> print(result)
[['EGGS'], ['JAM']]
```

utool.util_list.**get_list_column_slice**(*list_*, *start=None*, *stop=None*, *stride=None*)

utool.util_list.**group_consecutives**(*data*, *stepsize=1*)

Return list of consecutive lists of numbers from data (number list).

References

http://stackoverflow.com/questions/7352684/how-to-find-the-groups-of-consecutive-elements-from-an-array-in-numpy

utool.util_list.**group_consecutives_numpy**(*data*, *stepsize=1*)

Parameters

- **data** –
- **stepsize** (*int*) –

Returns list of ndarrays

Return type list

References

http://stackoverflow.com/questions/7352684/how-to-find-the-groups-of-consecutive-elements-from-an-array-in-numpy

CommandLine: python -m utool.util_list –test-group_consecutives

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> # build test data
>>> data = np.array([  0,   1,   2,   3,   4, 320, 636, 637, 638, 639])
>>> stepsize = 1
>>> # execute function
>>> result = group_consecutives(data, stepsize)
>>> # verify results
>>> print(result)
[array([0, 1, 2, 3, 4]), array([320]), array([636, 637, 638, 639])]
```

Timeit: %timeit group_consecutives_numpy(data, stepsize) # 14.8 µs per loop %timeit group_consecutives(data, stepsize) # 4.47 µs per loop

utool.util_list.**iflag_unique_items**(*list_*)
Returns a list of flags corresponding to the first time an item is seen

Parameters **list** (*list*) – list of items

Returns flag_iter

utool.util_list.**index_complement**(*index_list*, *len_=None*)
Returns the other indicies in a list of length len_

utool.util_list.**index_to_boolmask**(*indices*, *maxval=None*)
Constructs a list of booleans where an item is True if its position is in *indices* otherwise it is False.

Parameters

- **indices** (*list*) – list of integer indices
- **maxval** (*int*) – length of the returned list. If not specified this is inverred from *indices*

Returns mask: list of booleans. mask[idx] is True if idx in indices

Return type list

SeeAlso: vt.index_to_boolmask numpy version

CommandLine: python -m vtool.other index_to_boolmask

Example

```
>>> # ENABLE_DOCTEST
>>> import utool as ut
>>> indices = [0, 1, 4]
>>> maxval = 5
```

(continues on next page)

(continued from previous page)

```
>>> mask = ut.index_to_boolmask(indices, maxval)
>>> assert mask == [True, True, False, False, True]
```

utool.util_list.**insert_values**(*list_*, *index*, *values*, *inplace=False*)

utool.util_list.**intersect_ordered**(*list1*, *list2*)

returns list1 elements that are also in list2. preserves order of list1

intersect_ordered

Parameters

- **list1** (*list*) –
- **list2** (*list*) –

Returns new_list

Return type list

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> list1 = ['featweight_rowid', 'feature_rowid', 'config_rowid', 'featweight_
↪forground_weight']
>>> list2 = [u'featweight_rowid']
>>> result = intersect_ordered(list1, list2)
>>> print(result)
['featweight_rowid']
```

Timeit:

```
>>> def timeit_func(func, *args):
>>>     niter = 10
>>>     times = []
>>>     for count in range(niter):
>>>         with ut.Timer(verbose=False) as t:
>>>             _ = func(*args)
>>>         times.append(t.ellapsed)
>>>     return sum(times) / niter
>>>
>>> grid = {
>>>     'size1': [1000, 5000, 10000, 50000],
>>>     'size2': [1000, 5000, 10000, 50000],
>>>     #'overlap': [0, 1],
>>> }
>>> data = []
>>> for kw in ut.all_dict_combinations(grid):
>>>     pool = np.arange(kw['size1'] * 2)
>>>     size2 = size1 = kw['size1']
>>>     size2 = kw['size2']
>>>     list1 = (np.random.rand(size1) * size1).astype(np.int32).tolist()
>>>     list1 = ut.random_sample(pool, size1).tolist()
>>>     list2 = ut.random_sample(pool, size2).tolist()
>>>     list1 = set(list1)
>>>     list2 = set(list2)
```

(continues on next page)

(continued from previous page)

```
>>>     kw['ut'] = timeit_func(ut.isect, list1, list2)
>>>     #kw['np1'] = timeit_func(np.intersect1d, list1, list2)
>>>     #kw['py1'] = timeit_func(lambda a, b: set.intersection(set(a),␣
↪set(b)), list1, list2)
>>>     kw['py2'] = timeit_func(lambda a, b: sorted(set.intersection(set(a),␣
↪set(b))), list1, list2)
>>>     data.append(kw)
>>>
>>> import pandas as pd
>>> pd.options.display.max_rows = 1000
>>> pd.options.display.width = 1000
>>> df = pd.DataFrame.from_dict(data)
>>> data_keys = list(grid.keys())
>>> other_keys = ut.setdiff(df.columns, data_keys)
>>> df = df.reindex(data_keys + other_keys, axis=1)
>>> df['abs_change'] = df['ut'] - df['py2']
>>> df['pct_change'] = df['abs_change'] / df['ut'] * 100
>>> #print(df.sort('abs_change', ascending=False))
>>>
>>> print(str(df).split('\n')[0])
>>> for row in df.values:
>>>     argmin = row[len(data_keys):len(data_keys) + len(other_keys)].
↪argmin() + len(data_keys)
>>>     print('    ' + ', '.join([
>>>     '%6d' % (r) if x < len(data_keys) else (
>>>         ut.color_text('%8.6f' % (r,), 'blue')
>>>             if x == argmin else '%8.6f' % (r,))
>>>     for x, r in enumerate(row)
>>>     ]))
>>>
>>> %timeit ut.isect(list1, list2)
>>> %timeit np.intersect1d(list1, list2, assume_unique=True)
>>> %timeit set.intersection(set(list1), set(list2))
>>>
>>> #def highlight_max(s):
>>> #    '''
>>> #    highlight the maximum in a Series yellow.
>>> #    '''
>>> #    is_max = s == s.max()
>>> #    return ['background-color: yellow' if v else '' for v in is_max]
>>> #df.style.apply(highlight_max)
```

utool.util_list.**invertible_flatten1**(*unflat_list*)

Flattens *unflat_list* but remember how to reconstruct the *unflat_list* Returns *flat_list* and the *reverse_list* with indexes into the *flat_list*

Parameters **unflat_list** (*list*) – list of nested lists that we will flatten.

Returns (flat_list, reverse_list)

Return type tuple

CommandLine: python -m utool.util_list –exec-invertible_flatten1 –show

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> import utool as ut
>>> unflat_list = [[1, 2, 3], [4, 5], [6, 6]]
>>> flat_list, reverse_list = invertible_flatten1(unflat_list)
>>> result = ('flat_list = %s\n' % (ut.repr2(flat_list),))
>>> result += ('reverse_list = %s' % (ut.repr2(reverse_list),))
>>> print(result)
flat_list = [1, 2, 3, 4, 5, 6, 6]
reverse_list = [[0, 1, 2], [3, 4], [5, 6]]
```

utool.util_list.**invertible_flatten2**(*unflat_list*)

An alternative to invertible_flatten1 which uses cumsum

Flattens `list` but remember how to reconstruct the unflat `list` Returns flat `list` and the unflat `list` with indexes into the flat `list`

Parameters **unflat_list** (*list*) –

Returns flat_list, cumlen_list

Return type tuple

SeeAlso: invertible_flatten1 unflatten1 unflatten2

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> import utool
>>> utool.util_list
>>> unflat_list = [[5], [2, 3, 12, 3, 3], [9], [13, 3], [5]]
>>> flat_list, cumlen_list = invertible_flatten2(unflat_list)
>>> unflat_list2 = unflatten2(flat_list, cumlen_list)
>>> assert unflat_list2 == unflat_list
>>> result = ((flat_list, cumlen_list))
>>> print(result)
([5, 2, 3, 12, 3, 3, 9, 13, 3, 5], [1, 6, 7, 9, 10])
```

TODO: This flatten is faster fix it to be used everywhere

Timeit: unflat_list = [[random.random() for _ in range(int(random.random() * 1000))] for __ in range(200)] unflat_arrs = list(map(np.array, unflat_list))

%timeit invertible_flatten2(unflat_list) %timeit invertible_flatten2_numpy(unflat_list) %timeit invertible_flatten2_numpy(unflat_arrs)

Timeits: import utool unflat_list = aids_list1 flat_aids1, reverse_list = utool.invertible_flatten1(unflat_list) flat_aids2, cumlen_list = utool.invertible_flatten2(unflat_list) unflat_list1 = utool.unflatten1(flat_aids1, reverse_list) unflat_list2 = utool.unflatten2(flat_aids2, cumlen_list) assert list(map(list, unflat_list1)) == unflat_list2 print(utool.get_object_size_str(unflat_list, 'unflat_list ')) print(utool.get_object_size_str(flat_aids1, 'flat_aids1 ')) print(utool.get_object_size_str(flat_aids2, 'flat_aids2 ')) print(utool.get_object_size_str(reverse_list, 'reverse_list ')) print(utool.get_object_size_str(cumlen_list, 'cumlen_list ')) print(utool.get_object_size_str(unflat_list1, 'unflat_list1 ')) print(utool.get_object_size_str(unflat_list2, 'unflat_list2 ')) print('Timings 1:) %timeit

utool.invertible_flatten1(unflat_list) %timeit utool.unflatten1(flat_aids1, reverse_list) print('Timings 2:) %timeit utool.invertible_flatten2(unflat_list) %timeit utool.unflatten2(flat_aids2, cumlen_list)

utool.util_list.**invertible_flatten2_numpy**(*unflat_arrs*, *axis=0*)

more numpy version

TODO: move to vtool

Parameters **unflat_arrs** (*list*) – list of ndarrays

Returns (flat_list, cumlen_list)

Return type tuple

CommandLine: python -m utool.util_list –test-invertible_flatten2_numpy

Ignore:

```
>>> # ENABLE_DOCTET
>>> from utool.util_list import *  # NOQA
>>> unflat_arrs = [np.array([1, 2, 1]), np.array([5, 9]), np.array([4])]
>>> (flat_list, cumlen_list) = invertible_flatten2_numpy(unflat_arrs)
>>> result = str((flat_list, cumlen_list))
>>> print(result)
(array([1, 2, 1, 5, 9, 4]), array([3, 5, 6]))
```

utool.util_list.**invertible_total_flatten**(*unflat_list*)

Parameters **unflat_list** (*list*) –

Returns (flat_list, invert_levels)

Return type tuple

CommandLine: python -m utool.util_list –exec-invertible_total_flatten –show

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> import utool as ut
>>> unflat_list = [0, [[1, 2, 3], 4, 5], 9, [2, 3], [1, [2, 3, 4]], 1, 2, 3]
>>> print('unflat_list = %r' % (unflat_list,))
>>> (flat_list, invert_levels) = invertible_total_flatten(unflat_list)
>>> print('flat_list = %r' % (flat_list,))
>>> unflat_list2 = total_unflatten(flat_list, invert_levels)
>>> print('unflat_list2 = %r' % (unflat_list2,))
>>> assert unflat_list2 == unflat_list
>>> assert ut.depth_profile(flat_list) == 16
```

utool.util_list.**is_subset**(*list1*, *list2*)

utool.util_list.**is_subset_of_any**(*set_*, *other_sets*)

returns True if set_ is a subset of any set in other_sets

Parameters

- **set_** (*set*) –
- **other_sets** (*list of sets*) –

Returns flag

Return type bool

CommandLine: python -m utool.util_list –test-is_subset_of_any

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> # build test data
>>> set_ = {1, 2}
>>> other_sets = [{1, 4}, {3, 2, 1}]
>>> # execute function
>>> result = is_subset_of_any(set_, other_sets)
>>> # verify results
>>> print(result)
True
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> # build test data
>>> set_ = {1, 2}
>>> other_sets = [{1, 4}, {3, 2}]
>>> # execute function
>>> result = is_subset_of_any(set_, other_sets)
>>> # verify results
>>> print(result)
False
```

utool.util_list.**is_superset**(*list1*, *list2*)

utool.util_list.**isdisjoint**(*list1*, *list2*)

utool.util_list.**isect**(*list1*, *list2*)

returns list1 elements that are also in list2. preserves order of list1

intersect_ordered

Parameters

- **list1** (*list*) –
- **list2** (*list*) –

Returns new_list

Return type list

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> list1 = ['featweight_rowid', 'feature_rowid', 'config_rowid', 'featweight_
↪forground_weight']
```

(continues on next page)

(continued from previous page)

```
>>> list2 = [u'featweight_rowid']
>>> result = intersect_ordered(list1, list2)
>>> print(result)
['featweight_rowid']
```

Timeit:

```
>>> def timeit_func(func, *args):
>>>     niter = 10
>>>     times = []
>>>     for count in range(niter):
>>>         with ut.Timer(verbose=False) as t:
>>>             _ = func(*args)
>>>         times.append(t.ellapsed)
>>>     return sum(times) / niter
>>>
>>> grid = {
>>>     'size1': [1000, 5000, 10000, 50000],
>>>     'size2': [1000, 5000, 10000, 50000],
>>>     #'overlap': [0, 1],
>>> }
>>> data = []
>>> for kw in ut.all_dict_combinations(grid):
>>>     pool = np.arange(kw['size1'] * 2)
>>>     size2 = size1 = kw['size1']
>>>     size2 = kw['size2']
>>>     list1 = (np.random.rand(size1) * size1).astype(np.int32).tolist()
>>>     list1 = ut.random_sample(pool, size1).tolist()
>>>     list2 = ut.random_sample(pool, size2).tolist()
>>>     list1 = set(list1)
>>>     list2 = set(list2)
>>>     kw['ut'] = timeit_func(ut.isect, list1, list2)
>>>     #kw['np1'] = timeit_func(np.intersect1d, list1, list2)
>>>     #kw['py1'] = timeit_func(lambda a, b: set.intersection(set(a),␣
↪set(b)), list1, list2)
>>>     kw['py2'] = timeit_func(lambda a, b: sorted(set.intersection(set(a),␣
↪set(b))), list1, list2)
>>>     data.append(kw)
>>>
>>> import pandas as pd
>>> pd.options.display.max_rows = 1000
>>> pd.options.display.width = 1000
>>> df = pd.DataFrame.from_dict(data)
>>> data_keys = list(grid.keys())
>>> other_keys = ut.setdiff(df.columns, data_keys)
>>> df = df.reindex(data_keys + other_keys, axis=1)
>>> df['abs_change'] = df['ut'] - df['py2']
>>> df['pct_change'] = df['abs_change'] / df['ut'] * 100
>>> #print(df.sort('abs_change', ascending=False))
>>>
>>> print(str(df).split('\n')[0])
>>> for row in df.values:
>>>     argmin = row[len(data_keys):len(data_keys) + len(other_keys)].
↪argmin() + len(data_keys)
>>>     print('    ' + ', '.join([
>>>     '%6d' % (r) if x < len(data_keys) else (
```

(continues on next page)

(continued from previous page)

```
>>>         ut.color_text('%8.6f' % (r,), 'blue')
>>>             if x == argmin else '%8.6f' % (r,))
>>>     for x, r in enumerate(row)
>>>     ]))
>>>
>>> %timeit ut.isect(list1, list2)
>>> %timeit np.intersect1d(list1, list2, assume_unique=True)
>>> %timeit set.intersection(set(list1), set(list2))
>>>
>>> #def highlight_max(s):
>>> #    '''
>>> #    highlight the maximum in a Series yellow.
>>> #    '''
>>> #    is_max = s == s.max()
>>> #    return ['background-color: yellow' if v else '' for v in is_max]
>>> #df.style.apply(highlight_max)
```

utool.util_list.**isect_indices**(*items1*, *items2*)

utool.util_list.**isetdiff_flags**(*list1*, *list2*)
move to util_iter

utool.util_list.**issorted**(*list_*, *op=<built-in function le>*)
Determines if a list is sorted

Parameters

- **list** (*list*) –
- **op** (*func*) – sorted operation (default=operator.le)

Returns True if the list is sorted

Return type bool

utool.util_list.**issubset**(*list1*, *list2*)

utool.util_list.**issuperset**(*list1*, *list2*)

utool.util_list.**isunique**(*items*)

utool.util_list.**length_hint**(*obj*, *default=0*)
Return an estimate of the number of items in obj.

This is the PEP 424 implementation. If the object supports len(), the result will be exact. Otherwise, it may over- or under-estimate by an arbitrary amount. The result will be an integer >= 0.

utool.util_list.**listT**(*list_*, *shape=None*)
Swaps rows and columns. nCols should be specified if the initial list is empty.

Parameters **list** (*list*) –

Returns

Return type list

CommandLine: python -m utool.util_list –test-list_transpose

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> list_ = [[1, 2], [3, 4]]
>>> result = list_transpose(list_)
>>> print(result)
[(1, 3), (2, 4)]
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> list_ = []
>>> result = list_transpose(list_, shape=(0, 5))
>>> print(result)
[[], [], [], [], []]
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> list_ = [[], [], [], [], []]
>>> result = list_transpose(list_)
>>> print(result)
[]
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> import utool as ut
>>> list_ = [[1, 2, 3], [3, 4]]
>>> ut.assert_raises(ValueError, list_transpose, list_)
```

utool.util_list.**list_alignment**(*list1*, *list2*, *missing=False*)

Assumes list items are unique

Parameters

- **list1** (*list*) – a list of unique items to be aligned
- **list2** (*list*) – a list of unique items in a desired ordering
- **missing** (*bool*) – True if list2 can contain items not in list1

Returns sorting that will map list1 onto list2

Return type list

CommandLine: python -m utool.util_list list_alignment

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> import utool as ut
>>> list1 = ['b', 'c', 'a']
>>> list2 = ['a', 'b', 'c']
>>> sortx = list_alignment(list1, list2)
>>> list1_aligned = take(list1, sortx)
>>> assert list1_aligned == list2
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> import utool as ut
>>> list1 = ['b', 'c', 'a']
>>> list2 = ['a', 'a2', 'b', 'c', 'd']
>>> sortx = ut.list_alignment(list1, list2, missing=True)
>>> print('sortx = %r' % (sortx,))
>>> list1_aligned = ut.none_take(list1, sortx)
>>> result = ('list1_aligned = %s' % (ut.repr2(list1_aligned),))
>>> print(result)
list1_aligned = ['a', None, 'b', 'c', None]
```

utool.util_list.**list_all_eq_to**(*list_*, *val*, *strict=True*)

checks to see if list is equal everywhere to a value

Parameters

- **list** (*list*) –
- **val** – value to check against

Returns True if all items in the list are equal to val

utool.util_list.**list_argmax**(*list_*)

utool.util_list.**list_argmaxima**(*list_*)

Parameters **list** (*list*) –

Returns argmaxima

Return type list

CommandLine: python -m utool.util_list –exec-list_argmaxima

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> list_ = np.array([1, 2, 3, 3, 3, 2, 1])
>>> argmaxima = list_argmaxima(list_)
>>> result = ('argmaxima = %s' % (str(argmaxima),))
>>> print(result)
argmaxima = [2 3 4]
```

utool.util_list.**list_argsort**(**args*, ***kwargs*)
like np.argsort but for lists

Parameters

- ***args** – multiple lists to sort by
- ****kwargs** – reverse (bool): sort order is descending if True else acscending

CommandLine: python -m utool.util_list argsort

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> result = ut.argsort({'a': 3, 'b': 2, 'c': 100})
>>> print(result)
```

utool.util_list.**list_compress**(*item_list*, *flag_list*)
like np.compress but for lists

Returns items in item list where the corresponding item in flag list is True

Parameters

- **item_list** (*list*) – list of items to mask
- **flag_list** (*list*) – list of booleans used as a mask

Returns filtered_items - masked items

Return type list

utool.util_list.**list_compresstake**(*items_list*, *flags_list*)

utool.util_list.**list_cover**(*list1*, *list2*)
returns boolean for each position in list1 if it is in list2

Parameters

- **list1** (*list*) –
- **list2** (*list*) –

Returns incover_list - true where list1 intersects list2

Return type list

CommandLine: python -m utool.util_list –test-list_cover

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> # build test data
>>> list1 = [1, 2, 3, 4, 5, 6]
>>> list2 = [2, 3, 6]
>>> # execute function
>>> incover_list = list_cover(list1, list2)
>>> # verify results
```

(continues on next page)

(continued from previous page)

```
>>> result = str(incover_list)
>>> print(result)
[False, True, True, False, False, True]
```

utool.util_list.**list_deep_types**(*list_*)
Returns all types in a deep list

utool.util_list.**list_depth**(*list_*, *func=<built-in function max>*, *_depth=0*)
Returns the deepest level of nesting within a list of lists

Parameters

- **list** – a nested listlike object
- **func** – depth aggregation strategy (defaults to max)
- **_depth** – internal var

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> list_ = [[[[[1]]], [3]], [[1], [3]], [[1], [3]]]
>>> result = (list_depth(list_, _depth=0))
>>> print(result)
```

utool.util_list.**list_getattr**(*list_*, *attrname*)

utool.util_list.**list_intersection**(*list1*, *list2*)

utool.util_list.**list_inverse_take**(*list_*, *index_list*)

Parameters

- **list_** (*list*) – list in sorted domain
- **index_list** (*list*) – index list of the unsorted domain

---

Note: Seems to be logically equivalent to ut.take(list_, ut.argsort(index_list)), but faster

---

Returns output_list_ - the input list in the unsorted domain

Return type list

CommandLine: python -m utool.util_list –test-list_inverse_take

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> import utool as ut
>>> # build test data
>>> rank_list = [3, 2, 4, 1, 9, 2]
>>> prop_list = [0, 1, 2, 3, 4, 5]
>>> index_list = ut.argsort(rank_list)
```

(continues on next page)

(continued from previous page)

```
>>> sorted_prop_list = ut.take(prop_list, index_list)
>>> # execute function
>>> list_ = sorted_prop_list
>>> output_list_  = list_inverse_take(list_, index_list)
>>> output_list2_ = ut.take(list_, ut.argsort(index_list))
>>> assert output_list_ == prop_list
>>> assert output_list2_ == prop_list
>>> # verify results
>>> result = str(output_list_)
>>> print(result)
```

Timeit: %timeit list_inverse_take(list_, index_list) %timeit ut.take(list_, ut.argsort(index_list))

utool.util_list.**list_isdisjoint**(*list1*, *list2*)

utool.util_list.**list_issubset**(*list1*, *list2*)

utool.util_list.**list_issuperset**(*list1*, *list2*)

utool.util_list.**list_replace**(*list_*, *target*, *repl*)
alias

Recursively removes target in all lists and sublists and replaces them with the repl variable

utool.util_list.**list_reshape**(*list_*, *new_shape*, *trail=False*)
reshapes leaving trailing dimnsions in front if prod(new_shape) != len(list_)

Parameters

- **list_** (*list*) –
- **new_shape** (*tuple*) –

Returns list

CommandLine: python -m utool.util_list –exec-list_reshape –show

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> import utool as ut
>>> import numpy as np
>>> list_ = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
>>> new_shape = (2, 2, 3)
>>> newlist = list_reshape(list_, new_shape)
>>> depth = ut.depth_profile(newlist)
>>> result = ('list_ = %s' % (ut.repr2(newlist, nl=1),))
>>> print('depth = %r' % (depth,))
>>> print(result)
>>> newlist2 = np.reshape(list_, depth).tolist()
>>> ut.assert_eq(newlist, newlist2)
```

utool.util_list.**list_roll**(*list_*, *n*)
Like numpy.roll for python lists

Parameters

- **list_** (*list*) –

- **n** (*int*) –

Returns

Return type list

References

http://stackoverflow.com/questions/9457832/python-list-rotation

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> list_ = [1, 2, 3, 4, 5]
>>> n = 2
>>> result = list_roll(list_, n)
>>> print(result)
[4, 5, 1, 2, 3]
```

Ignore: np.roll(list_, n)

utool.util_list.**list_set_equal**(*list1*, *list2*)

utool.util_list.**list_strip**(*list_*, *to_strip*, *left=True*, *right=True*)
list_ = [1, 2, 1, 3, 1, 1] to_strip = 1 stripped_list = ut.list_strip(list_, to_strip)

utool.util_list.**list_take**(*list_*, *index_list*)
Selects a subset of a list based on a list of indices. This is similar to np.take, but pure python.

Parameters

- **list** (*list*) – some indexable object
- **index_list** (*list, slice, int*) – some indexing object

Returns subset of the list

Return type list or scalar

CommandLine: python -m utool.util_list –test-take

SeeAlso: ut.dict_take ut.dict_subset ut.none_take ut.compress

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> list_ = [0, 1, 2, 3]
>>> index_list = [2, 0]
>>> result = take(list_, index_list)
>>> print(result)
[2, 0]
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> list_ = [0, 1, 2, 3]
>>> index = 2
>>> result = take(list_, index)
>>> print(result)
2
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> list_ = [0, 1, 2, 3]
>>> index = slice(1, None, 2)
>>> result = take(list_, index)
>>> print(result)
[1, 3]
```

utool.util_list.**list_transpose**(*list_*, *shape=None*)

Swaps rows and columns. nCols should be specified if the initial list is empty.

Parameters **list** (*list*) –

Returns

Return type list

CommandLine: python -m utool.util_list –test-list_transpose

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> list_ = [[1, 2], [3, 4]]
>>> result = list_transpose(list_)
>>> print(result)
[(1, 3), (2, 4)]
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> list_ = []
>>> result = list_transpose(list_, shape=(0, 5))
>>> print(result)
[[], [], [], [], []]
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> list_ = [[], [], [], [], []]
>>> result = list_transpose(list_)
>>> print(result)
[]
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> import utool as ut
>>> list_ = [[1, 2, 3], [3, 4]]
>>> ut.assert_raises(ValueError, list_transpose, list_)
```

utool.util_list.**list_type**(*list_*)

utool.util_list.**list_type_profile**(*sequence*, *compress_homogenous=True*, *with_dtype=True*)
similar to depth_profile but reports types

Parameters

- **sequence** –
- **compress_homogenous** (*bool*) – (default = True)

Returns level_type_str

Return type str

CommandLine: python -m utool.util_list –test-list_type_profile python3 -m utool.util_list –test-list_type_profile

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> import numpy as np
>>> sequence = [[1, 2], np.array([1, 2, 3], dtype=np.int32), (np.array([1, 2, 3],␣
↪dtype=np.int32),)]
>>> compress_homogenous = True
>>> level_type_str = list_type_profile(sequence, compress_homogenous)
>>> result = ('level_type_str = %s' % (str(level_type_str),))
>>> print(result)
level_type_str = list(list(int*2), ndarray[int32], tuple(ndarray[int32]*1))
```

utool.util_list.**list_union**(**lists*)

utool.util_list.**list_where**(*flag_list*)
takes flags returns indexes of True values

utool.util_list.**list_zipcompress**(*items_list*, *flags_list*)

SeeAlso: vt.zipcompress

utool.util_list.**list_zipflatten**(**items_lists*)

utool.util_list.**list_ziptake**(*items_list*, *indexes_list*)

SeeAlso: vt.ziptake

utool.util_list.**listclip**(*list_*, *num*, *fromback=False*)

DEPRICATE: use slices instead

Parameters

- **list** (*list*) –
- **num** (*int*) –

Returns

Return type sublist

CommandLine: python -m utool.util_list –test-listclip

Example

```
>>> # ENABLE_DOCTEST
>>> import utool as ut
>>> # build test data
>>> list_ = [1, 2, 3, 4, 5]
>>> result_list = []
>>> # execute function
>>> num = 3
>>> result_list += [ut.listclip(list_, num)]
>>> num = 9
>>> result_list += [ut.listclip(list_, num)]
>>> # verify results
>>> result = ut.repr4(result_list)
>>> print(result)
[
    [1, 2, 3],
    [1, 2, 3, 4, 5],
]
```

Example

```
>>> # ENABLE_DOCTEST
>>> import utool as ut
>>> # build test data
>>> list_ = [1, 2, 3, 4, 5]
>>> result_list = []
>>> # execute function
>>> num = 3
>>> result = ut.listclip(list_, num, fromback=True)
>>> print(result)
[3, 4, 5]
```

utool.util_list.**listfind**(*list_*, *tofind*)

get the position of item `tofind` in `list_` if it exists otherwise returns None

Parameters

- **list** –

- **tofind** –

Returns index of `tofind` in `list_`

Return type int

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> list_ = ['a', 'b', 'c']
>>> tofind = 'b'
>>> result = listfind(list_, tofind)
>>> print(result)
1
```

utool.util_list.**lmap**(*func*, *iter_*, ***kwargs*)

Eager version of the builtin map function. This provides the same functionality as python2 map.

Note this is inefficient and should only be used when prototyping and debugging.

Extended functionality supports passing common kwargs to all functions

utool.util_list.**lstarmap**(*func*, *iter_*, ***kwargs*)

Eager version of it.starmap from itertools

Note this is inefficient and should only be used when prototyping and debugging.

utool.util_list.**lzip**(**args*)

Eager version of the builtin zip function. This provides the same functionality as python2 zip.

Note this is inefficient and should only be used when prototyping and debugging.

utool.util_list.**make_index_lookup**(*list_*, *dict_factory=<class 'dict'>*)

Parameters **list** (*list*) – assumed to have unique items

Returns mapping from item to index

Return type dict

CommandLine: python -m utool.util_list –exec-make_index_lookup

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> import utool as ut
>>> list_ = [5, 3, 8, 2]
>>> idx2_item = ut.make_index_lookup(list_)
>>> result = ut.repr2(idx2_item, nl=False)
>>> assert ut.dict_take(idx2_item, list_) == list(range(len(list_)))
>>> print(result)
{2: 3, 3: 1, 5: 0, 8: 2}
```

utool.util_list.**make_sortby_func**(*item_list*, *reverse=False*)

utool.util_list.**maplen**(*iter_*)

utool.util_list.**multi_replace**(*instr*, *search_list=[]*, *repl_list=None*)
Does a string replace with a list of search and replacements

TODO: rename

utool.util_list.**none_take**(*list_*, *index_list*)
Like take but indices can be None

SeeAlso: ut.take

utool.util_list.**not_list**(*flag_list*)

utool.util_list.**or_lists**(**args*)

utool.util_list.**partial_imap_1to1**(*func*, *si_func*)
a bit messy

DEPRICATE

utool.util_list.**partial_order**(*list_*, *part*)

utool.util_list.**print_duplicate_map**(*duplicate_map*, **args*, ***kwargs*)

utool.util_list.**priority_argsort**(*list_*, *priority*)

Parameters

- **list** (*list*) –
- **priority** (*list*) – desired order of items

Returns reordered_index_list

Return type list

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> import utool as ut
>>> list_ = [2, 4, 6, 8, 10]
>>> priority = [8, 2, 6, 9]
>>> sortx = priority_argsort(list_, priority)
>>> reordered_list = priority_sort(list_, priority)
>>> assert ut.take(list_, sortx) == reordered_list
>>> result = str(sortx)
>>> print(result)
[3, 0, 2, 1, 4]
```

utool.util_list.**priority_sort**(*list_*, *priority*)

Parameters

- **list** (*list*) –
- **priority** (*list*) – desired order of items

Returns reordered_list

Return type list

CommandLine: python -m utool.util_list –test-priority_argsort

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> list_ = [2, 4, 6, 8, 10]
>>> priority = [8, 2, 6, 9]
>>> reordered_list = priority_sort(list_, priority)
>>> result = str(reordered_list)
>>> print(result)
[8, 2, 6, 4, 10]
```

utool.util_list.**rebase_labels**(*label_list*)

utool.util_list.**recursive_replace**(*list_*, *target*, *repl=-1*)
Recursively removes target in all lists and sublists and replaces them with the repl variable

utool.util_list.**replace_nones**(*list_*, *repl=-1*)
Recursively removes Nones in all lists and sublists and replaces them with the repl variable

Parameters

- **list** (*list*) –
- **repl** (*obj*) – replacement value

Returns list

CommandLine: python -m utool.util_list –test-replace_nones

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> # build test data
>>> list_ = [None, 0, 1, 2]
>>> repl = -1
>>> # execute function
>>> repl_list = replace_nones(list_, repl)
>>> # verify results
>>> result = str(repl_list)
>>> print(result)
[-1, 0, 1, 2]
```

utool.util_list.**safe_listget**(*list_*, *index*, *default='?'*)
depricate

utool.util_list.**safe_slice**(*list_*[, *start*], *stop*[, *end*][, *step*])
Slices list and truncates if out of bounds

utool.util_list.**safeapply**(*func*, **args*, ***kwargs*)

utool.util_list.**safelen**(*list_*)

utool.util_list.**sample_lists**(*items_list*, *num=1*, *seed=None*)

Parameters

- **items_list** (*list*) –
- **num** (*int*) – (default = 1)

- **seed** (*None*) – (default = None)

Returns samples_list

Return type list

CommandLine: python -m utool.util_list –exec-sample_lists

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> items_list = [[], [1, 2, 3], [4], [5, 6], [7, 8, 9, 10]]
>>> num = 2
>>> seed = 0
>>> samples_list = sample_lists(items_list, num, seed)
>>> result = ('samples_list = %s' % (str(samples_list),))
>>> print(result)
samples_list = [[], [3, 2], [4], [5, 6], [10, 9]]
```

utool.util_list.**sample_zip**(*items_list*, *num_samples*, *allow_overflow=False*, *per_bin=1*)

Helper for sampling

Given a list of lists, samples one item for each list and bins them into num_samples bins. If all sublists are of equal size this is equivilent to a zip, but otherwise consecutive bins will have monotonically less elemements

Doctest doesn't work with assertionerror #util_list.sample_zip(items_list, 2) #... #AssertionError: Overflow occured

Parameters

- **items_list** (*list*) –
- **num_samples** –
- **allow_overflow** (*bool*) –
- **per_bin** (*int*) –

Returns (samples_list, overflow_samples)

Return type tuple

Example

```
>>> # DISABLE_DOCTEST
>>> from utool import util_list
>>> items_list = [[1, 2, 3, 4, 0], [5, 6, 7], [], [8, 9], [10]]
>>> util_list.sample_zip(items_list, 5)
...
[[1, 5, 8, 10], [2, 6, 9], [3, 7], [4], [0]]
>>> util_list.sample_zip(items_list, 2, allow_overflow=True)
...
([[1, 5, 8, 10], [2, 6, 9]], [3, 7, 4])
>>> util_list.sample_zip(items_list, 4, allow_overflow=True, per_bin=2)
...
([[1, 5, 8, 10, 2, 6, 9], [3, 7, 4], [], []], [0])
```

utool.util_list.**scalar_input_map**(*func*, *input_*)
Map like function

Parameters

- **func** – function to apply
- **input** – either an iterable or scalar value

Returns If input_ is iterable this function behaves like map otherwise applies func to input_

utool.util_list.**search_list**(*text_list*, *pattern*, *flags=0*)

CommandLine: python -m utool.util_list –test-search_list

Example

```
>>> # ENABLE_DOCTEST
>>> import utool as ut
>>> text_list = ['ham', 'jam', 'eggs', 'spam']
>>> pattern = '.am'
>>> flags = 0
>>> (valid_index_list, valid_match_list) = ut.search_list(text_list, pattern,␣
↪flags)
>>> result = str(valid_index_list)
>>> print(result)
[0, 1, 3]
```

utool.util_list.**setdiff**(*list1*, *list2*)
returns list1 elements that are not in list2. preserves order of list1

Parameters

- **list1** (*list*) –
- **list2** (*list*) –

Returns new_list

Return type list

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> import utool as ut
>>> list1 = ['featweight_rowid', 'feature_rowid', 'config_rowid', 'featweight_
↪forground_weight']
>>> list2 = [u'featweight_rowid']
>>> new_list = setdiff_ordered(list1, list2)
>>> result = ut.repr4(new_list, nl=False)
>>> print(result)
['feature_rowid', 'config_rowid', 'featweight_forground_weight']
```

utool.util_list.**setdiff_flags**(*list1*, *list2*)

utool.util_list.**setdiff_ordered**(*list1*, *list2*)
returns list1 elements that are not in list2. preserves order of list1

Parameters

- **list1** (*list*) –
- **list2** (*list*) –

Returns new_list

Return type list

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> import utool as ut
>>> list1 = ['featweight_rowid', 'feature_rowid', 'config_rowid', 'featweight_
↪forground_weight']
>>> list2 = [u'featweight_rowid']
>>> new_list = setdiff_ordered(list1, list2)
>>> result = ut.repr4(new_list, nl=False)
>>> print(result)
['feature_rowid', 'config_rowid', 'featweight_forground_weight']
```

utool.util_list.**setintersect**(*list1*, *list2*)
returns list1 elements that are in list2. preserves order of list1

setintersect_ordered

Parameters

- **list1** (*list*) –
- **list2** (*list*) –

Returns new_list

Return type list

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> list1 = [1, 2, 3, 5, 8, 13, 21]
>>> list2 = [6, 4, 2, 21, 8]
>>> new_list = setintersect_ordered(list1, list2)
>>> result = new_list
>>> print(result)
[2, 8, 21]
```

utool.util_list.**setintersect_ordered**(*list1*, *list2*)
returns list1 elements that are in list2. preserves order of list1

setintersect_ordered

Parameters

- **list1** (*list*) –
- **list2** (*list*) –

Returns new_list

Return type list

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> list1 = [1, 2, 3, 5, 8, 13, 21]
>>> list2 = [6, 4, 2, 21, 8]
>>> new_list = setintersect_ordered(list1, list2)
>>> result = new_list
>>> print(result)
[2, 8, 21]
```

utool.util_list.**snapped_slice**(*size*, *frac*, *n*)

Creates a slice spanning *n* items in a list of length *size* at position *frac*.

Parameters

- **size** (*int*) – length of the list
- **frac** (*float*) – position in the range [0, 1]
- **n** (*int*) – number of items in the slice

Returns slice object that best fits the criteria

Return type slice

SeeAlso: take_percentile_parts

Example:

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> import utool as ut
>>> print(snapped_slice(0, 0, 10))
>>> print(snapped_slice(1, 0, 10))
>>> print(snapped_slice(100, 0, 10))
>>> print(snapped_slice(9, 0, 10))
>>> print(snapped_slice(100, 1, 10))
pass
```

utool.util_list.**sortedby**(*item_list*, *key_list*, *reverse=False*)

sorts `item_list` using key_list

Parameters

- **list** (*list*) – list to sort
- **key_list** (*list*) – list to sort by
- **reverse** (*bool*) – sort order is descending (largest first) if reverse is True else acscending (smallest first)

Returns `list_` sorted by the values of another `list`. defaults to ascending order

Return type list

SeeAlso: sortedby2

Example

```
>>> # ENABLE_DOCTEST
>>> import utool
>>> list_    = [1, 2, 3, 4, 5]
>>> key_list = [2, 5, 3, 1, 5]
>>> result = utool.sortedby(list_, key_list, reverse=True)
>>> print(result)
[5, 2, 3, 1, 4]
```

utool.util_list.**sortedby2**(*item_list*, **args*, ***kwargs*)

sorts item_list using key_list

Parameters

- **item_list** (*list*) – list to sort
- ***args** – multiple lists to sort by
- ****kwargs** – reverse (bool): sort order is descending if True else acscending

Returns list_ sorted by the values of another list. defaults to ascending order

Return type list

CommandLine: python -m utool.util_list –exec-sortedby2 –show

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> import utool as ut
>>> item_list = [1, 2, 3, 4, 5]
>>> key_list1 = [1, 1, 2, 3, 4]
>>> key_list2 = [2, 1, 4, 1, 1]
>>> args = (key_list1, key_list2)
>>> kwargs = dict(reverse=False)
>>> result = ut.sortedby2(item_list, *args, **kwargs)
>>> print(result)
[2, 1, 3, 4, 5]
```

Example

```
>>> # ENABLE_DOCTEST
>>> # Python 3 Compatibility Test
>>> import utool as ut
>>> item_list = [1, 2, 3, 4, 5]
>>> key_list1 = ['a', 'a', 2, 3, 4]
>>> key_list2 = ['b', 'a', 4, 1, 1]
>>> args = (key_list1, key_list2)
>>> kwargs = dict(reverse=False)
>>> result = ut.sortedby2(item_list, *args, **kwargs)
>>> print(result)
[3, 4, 5, 2, 1]
```

utool.util_list.**strided_sample**(*items*, *num*, *offset=0*)

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> # build test data
>>> items = [1, 2, 3, 4, 5]
>>> num = 3
>>> offset = 0
>>> # execute function
>>> sample_items = strided_sample(items, num, offset)
>>> # verify results
>>> result = str(sample_items)
>>> print(result)
```

utool.util_list.**take**(*list_*, *index_list*)

Selects a subset of a list based on a list of indices. This is similar to np.take, but pure python.

Parameters

- **list** (*list*) – some indexable object
- **index_list** (*list, slice, int*) – some indexing object

Returns subset of the list

Return type list or scalar

CommandLine: python -m utool.util_list –test-take

SeeAlso: ut.dict_take ut.dict_subset ut.none_take ut.compress

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> list_ = [0, 1, 2, 3]
>>> index_list = [2, 0]
>>> result = take(list_, index_list)
>>> print(result)
[2, 0]
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> list_ = [0, 1, 2, 3]
>>> index = 2
>>> result = take(list_, index)
>>> print(result)
2
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> list_ = [0, 1, 2, 3]
>>> index = slice(1, None, 2)
>>> result = take(list_, index)
>>> print(result)
[1, 3]
```

utool.util_list.**take_around_percentile**(*arr*, *frac*, *n*)

utool.util_list.**take_column**(*list_*, *colx*)

accepts a list of (indexables) and returns a list of indexables can also return a list of list of indexables if colx is a list

Parameters

- **list** (*list*) – list of lists
- **colx** (*int or list*) – index or key in each sublist get item

Returns list of selected items

Return type list

CommandLine: python -m utool.util_list –test-take_column

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> list_ = [['a', 'b'], ['c', 'd']]
>>> colx = 0
>>> result = take_column(list_, colx)
>>> import utool as ut
>>> result = ut.repr4(result, nl=False)
>>> print(result)
['a', 'c']
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> list_ = [['a', 'b'], ['c', 'd']]
>>> colx = [1, 0]
>>> result = take_column(list_, colx)
>>> import utool as ut
>>> result = ut.repr4(result, nl=False)
>>> print(result)
[['b', 'a'], ['d', 'c']]
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> list_ = [{'spam': 'EGGS', 'ham': 'SPAM'}, {'spam': 'JAM', 'ham': 'PRAM'}]
>>> # colx can be a key or list of keys as well
>>> colx = ['spam']
>>> result = take_column(list_, colx)
>>> import utool as ut
>>> result = ut.repr4(result, nl=False)
>>> print(result)
[['EGGS'], ['JAM']]
```

utool.util_list.**take_complement**(*list_*, *index_list*)
Returns items in list_ not indexed by index_list

utool.util_list.**take_percentile**(*arr*, *percent*)
take the top *percent* items in a list rounding up

utool.util_list.**take_percentile_parts**(*arr*, *front=None*, *mid=None*, *back=None*)
Take parts from front, back, or middle of a list

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> import utool as ut
>>> arr = list(range(20))
>>> front = 3
>>> mid = 3
>>> back = 3
>>> result = take_percentile_parts(arr, front, mid, back)
>>> print(result)
[0, 1, 2, 9, 10, 11, 17, 18, 19]
```

utool.util_list.**total_flatten**(*unflat_list*)
unflat_list = [1, 2, [3, 4], [5, [9]]] :param unflat_list: :type unflat_list: list

Returns flat_list

Return type list

CommandLine: python -m utool.util_list –exec-total_flatten –show

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> import utool as ut
>>> unflat_list = [[[1, 2, 3], 4, 5], 9, [2, 3], [1, [2, 3, 4]], 1, 2, 3]
>>> flat_list = total_flatten(unflat_list)
>>> result = ('flat_list = %s' % (ut.repr2(flat_list),))
>>> print(result)
```

utool.util_list.**total_unflatten**(*flat_list*, *invert_levels*)

utool.util_list.**type_profile**(*sequence*, *compress_homogenous=True*, *with_dtype=True*)
similar to depth_profile but reports types

Parameters

- **sequence** –
- **compress_homogenous** (*bool*) – (default = True)

Returns level_type_str

Return type str

CommandLine: python -m utool.util_list –test-list_type_profile python3 -m utool.util_list –test-list_type_profile

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> import numpy as np
>>> sequence = [[1, 2], np.array([1, 2, 3], dtype=np.int32), (np.array([1, 2, 3],␣
↪dtype=np.int32),)]
>>> compress_homogenous = True
>>> level_type_str = list_type_profile(sequence, compress_homogenous)
>>> result = ('level_type_str = %s' % (str(level_type_str),))
>>> print(result)
level_type_str = list(list(int*2), ndarray[int32], tuple(ndarray[int32]*1))
```

utool.util_list.**type_profile2**(*sequence*, *TypedSequence=None*)

similar to depth_profile but reports types

Parameters

- **sequence** –
- **compress_homogenous** (*bool*) – (default = True)

Returns level_type_str

Return type str

CommandLine: python -m utool.util_list –exec-type_profile2

Example

```
>>> # DISABLE_DOCTEST
>>> sequence = []
>>> from utool.util_list import *  # NOQA
>>> self = typeprof = type_profile2(sequence, type_sequence_factory())
>>> result = ('level_type_str = %s' % (str(level_type_str),))
>>> print(result)
```

utool.util_list.**type_sequence_factory**()

utool.util_list.**unflat_map**(*func*, *unflat_items*, *vectorized=False*, ***kwargs*)

Uses an wbia lookup function with a non-flat rowid list. In essence this is equivilent to [list(map(func, _items)) for _items in unflat_items]. The utility of this function is that it only calls method once. This is more efficient for calls that can take a list of inputs

Parameters

- **func** (*func*) – function
- **unflat_items** (*list*) – list of rowid lists

Returns unflat_vals

Return type list of values

CommandLine: python -m utool.util_list –test-unflat_map

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> vectorized = False
>>> kwargs = {}
>>> def func(x):
>>>     return x + 1
>>> unflat_items = [[], [1, 2, 3], [4, 5], [6, 7, 8, 9], [], []]
>>> unflat_vals = unflat_map(func, unflat_items)
>>> result = str(unflat_vals)
>>> print(result)
[[], [2, 3, 4], [5, 6], [7, 8, 9, 10], [], []]
```

utool.util_list.**unflat_take**(*items_list*, *unflat_index_list*)

Returns nested subset of items_list

Parameters

- **items_list** (*list*) –
- **unflat_index_list** (*list*) – nested list of indices

CommandLine: python -m utool.util_list –exec-unflat_take

SeeAlso: ut.take

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> items_list = [1, 2, 3, 4, 5]
>>> unflat_index_list = [[0, 1], [2, 3], [0, 4]]
>>> result = unflat_take(items_list, unflat_index_list)
>>> print(result)
[[1, 2], [3, 4], [1, 5]]
```

utool.util_list.**unflat_unique_rowid_map**(*func*, *unflat_rowids*, ***kwargs*)

performs only one call to the underlying func with unique rowids the func must be some lookup function

TODO: move this to a better place.

CommandLine: python -m utool.util_list –test-unflat_unique_rowid_map:0 python -m utool.util_list –test-unflat_unique_rowid_map:1

Ignore:

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> import utool as ut
>>> kwargs = {}
>>> unflat_rowids = [[1, 2, 3], [2, 5], [1], []]
>>> num_calls0 = [0]
>>> num_input0 = [0]
>>> def func0(rowids, num_calls0=num_calls0, num_input0=num_input0):
...     num_calls0[0] += 1
...     num_input0[0] += len(rowids)
...     return [rowid + 10 for rowid in rowids]
>>> func = func0
>>> unflat_vals = unflat_unique_rowid_map(func, unflat_rowids, **kwargs)
>>> result = [arr.tolist() for arr in unflat_vals]
>>> print(result)
>>> ut.assert_eq(num_calls0[0], 1)
>>> ut.assert_eq(num_input0[0], 4)
[[11, 12, 13], [12, 15], [11], []]
```

Ignore:

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> import utool as ut
>>> import numpy as np
>>> kwargs = {}
>>> unflat_rowids = [[1, 2, 3], [2, 5], [1], []]
>>> num_calls1 = [0]
>>> num_input1 = [0]
>>> def func1(rowids, num_calls1=num_calls1, num_input1=num_input1, np=np):
...     num_calls1[0] += 1
...     num_input1[0] += len(rowids)
...     return [np.array([rowid + 10, rowid, 3]) for rowid in rowids]
>>> func = func1
>>> unflat_vals = unflat_unique_rowid_map(func, unflat_rowids, **kwargs)
>>> result = [arr.tolist() for arr in unflat_vals]
>>> print(result)
>>> ut.assert_eq(num_calls1[0], 1)
>>> ut.assert_eq(num_input1[0], 4)
[[[11, 1, 3], [12, 2, 3], [13, 3, 3]], [[12, 2, 3], [15, 5, 3]], [[11, 1, 3]],
↪ []]
```

utool.util_list.**unflat_vecmap**(*func*, *unflat_items*, *vectorized=False*, ***kwargs*)
unflat map for vectorized functions

utool.util_list.**unflatten1**(*flat_list*, *reverse_list*)
Rebuilds unflat list from invertible_flatten1

Parameters

- **flat_list** (*list*) – the flattened list
- **reverse_list** (*list*) – the list which undoes flattenting

Returns original nested list

Return type unflat_list2

SeeAlso: invertible_flatten1 invertible_flatten2 unflatten2

utool.util_list.**unflatten2**(*flat_list*, *cumlen_list*)
Rebuilds unflat list from invertible_flatten1

Parameters

- **flat_list** (*list*) – the flattened list
- **cumlen_list** (*list*) – the list which undoes flattenting

Returns original nested list

Return type unflat_list2

SeeAlso: invertible_flatten1 invertible_flatten2 unflatten2

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> import utool
>>> utool.util_list
>>> flat_list = [5, 2, 3, 12, 3, 3, 9, 13, 3, 5]
>>> cumlen_list = [ 1,  6,  7,  9, 10]
>>> unflat_list2 = unflatten2(flat_list, cumlen_list)
>>> result = (unflat_list2)
>>> print(result)
[[5], [2, 3, 12, 3, 3], [9], [13, 3], [5]]
```

utool.util_list.**union**(**lists*, ***kwargs*)

Ignore:

```
>>> %timeit len(reduce(set.union, map(set, x)))
>>> %timeit len(ut.union(*x))
>>> %timeit len(ut.unique(ut.flatten(ut.lmap(np.unique, x))))
>>> %timeit len(ut.unique(ut.flatten(x)))
>>> %timeit len(ut.union(*x))
>>> %timeit len(ut.list\_union(*x))
>>> %timeit len(set.union(*[set(list\_) for list\_ in lists]))
>>> %timeit len(set.union(*(set(list\_) for list\_ in lists)))
```

utool.util_list.**union_ordered**(**lists*)

utool.util_list.**unique**(*list_*, *ordered=True*)
Returns unique items in `list_`. Generally, unordered (*should be*) faster.

utool.util_list.**unique_flags**(*list_*)
Returns a list of flags corresponding to the first time an item is seen

Parameters **list_** (*list*) – list of items

Returns flag_list

Timing:

```
>>> import random
>>> import utool as ut
>>>
>>> def random_items(n, m):
```

(continues on next page)

(continued from previous page)

```
>>>     rng = random.Random(0)
>>>     return [rng.randint(0, n) for _ in range(m)]
>>>
>>> m = 1000
>>>
>>> def method1(list\_):
>>>     seen = set()
>>>     def unseen(item):
>>>         if item in seen:
>>>             return False
>>>         seen.add(item)
>>>         return True
>>>     flag_list = [unseen(item) for item in list\_]
>>>     return flag_list
>>>
>>> def method2(list\_):
>>>     return ut.index_to_boolmask([list\_.index(x) for x in set(list\_)],␣
↪len(list\_))
>>>
>>> def method3(list\_):
>>>     return ut.index_to_boolmask(dict(zip(reversed(list\_),␣
↪reversed(range(len(list\_))))).values(), len(list\_))
>>>
>>>
>>> import ubelt as ub
>>> ub.Timerit.DEFAULT_VERBOSE = False
>>>
>>> ut.qtensure()
>>> exps = [0, .25, .5, .75, 1, 2]
>>> pnum = pt.make_pnum_nextgen(nSubplots=len(exps))
>>> current = ut.flag_unique_items
>>>
>>> for count, exp in ut.ProgIter(list(enumerate(exps, start=1))):
>>>     ydatas = ut.ddict(list)
>>>     xdata = []
>>>     for m in ut.ProgIter(list(range(0, 10000, 100)), freq=1):
>>>         xdata.append(m)
>>>         num = 10
>>>         n = int(m ** exp)
>>>         list\_ = random_items(n=n, m=m)
>>>         ydatas['method1'].append(ub.Timerit(num).call(method1, list\_))
>>>         ydatas['method2'].append(ub.Timerit(num).call(method2, list\_))
>>>         ydatas['method3'].append(ub.Timerit(num).call(method3, list\_))
>>>         ydatas['current'].append(ub.Timerit(num).call(current, list\_))
>>>
>>>         # assert method1(list\_) == method3(list\_)
>>>         # assert method1(list\_) == current(list\_)
>>>
>>>     pt.multi_plot(
>>>         xdata, list(ydatas.values()), label_list=list(ydatas.keys()),
>>>         ylabel='time', title=str(exp), fnum=1, pnum=pnum())
```

utool.util_list.**unique_indices**(*list_*)

utool.util_list.**unique_inverse**(*item_list*)

Like np.unique(item_list, return_inverse=True)

utool.util_list.**unique_ordered**(*list_*)
Returns unique items in `list_` in the order they were seen.

Parameters **list** (*list*) –

Returns unique_list - unique list which maintains order

Return type list

CommandLine: python -m utool.util_list –exec-unique_ordered

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> list_ = [4, 6, 6, 0, 6, 1, 0, 2, 2, 1]
>>> unique_list = unique_ordered(list_)
>>> result = ('unique_list = %s' % (str(unique_list),))
>>> print(result)
unique_list = [4, 6, 0, 1, 2]
```

utool.util_list.**unique_unordered**(*list_*)
wrapper around list(set(list_))

utool.util_list.**unpack_iterables**(*list_*)

utool.util_list.**where**(*flag_list*)
takes flags returns indexes of True values

utool.util_list.**where_not_None**(*item_list*)
returns list of indexes of non None values

SeeAlso: flag_None_items

utool.util_list.**xor_lists**(**args*)

Returns

Return type list

CommandLine: python -m utool.util_list –test-xor_lists

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_list import *  # NOQA
>>> args = ([True, False, False, True], [True, True, False, False])
>>> result = xor_lists(*args)
>>> print(result)
[False, True, False, True]
```

utool.util_list.**zipcompress**(*items_list*, *flags_list*)

SeeAlso: vt.zipcompress

utool.util_list.**zipflat**(**args*)

utool.util_list.**ziptake**(*items_list*, *indexes_list*)

SeeAlso: vt.ziptake

1.39 utool.util_logging module

If logging is on, utool will overwrite the print function with a logging function

This is a special module which will not get injected into (should it be internal?)

References

maybe we can do something like this Queue to try fixing error when # when using injected print statments with Qt signals and slots http://stackoverflow.com/questions/21071448/redirecting-stdout-and-stderr-to-a-pyqt4-qtextedit-from-a-secondary-thread

class utool.util_logging.**CustomStreamHandler**(*stream=None*)
Bases: logging.Handler

Modified from logging.py

emit(*record*)
Emit a record.

If a formatter is specified, it is used to format the record. The record is then written to the stream with a trailing newline. If exception information is present, it is formatted using traceback.print_exception and appended to the stream. If the stream has an 'encoding' attribute, it is used to determine how to do the output to the stream.

flush()
Flushes the stream.

utool.util_logging.**add_logging_handler**(*handler*, *format_='file'*)
mostly for util_logging internals

utool.util_logging.**debug_logging_iostreams**()

utool.util_logging.**ensure_logging**()

utool.util_logging.**get_current_log_fpath**()

utool.util_logging.**get_current_log_text**()

utool.util_logging.**get_log_fpath**(*num='next'*, *appname=None*, *log_dir=None*)

Returns path to log file

Return type log_fpath (str)

utool.util_logging.**get_logging_dir**(*appname='default'*)
The default log dir is in the system resource directory But the utool global cache allows for the user to override where the logs for a specific app should be stored.

Returns real path to logging directory

Return type log_dir_realpath (str)

utool.util_logging.**get_shelves_dir**(*appname='default'*)
The default shelf dir is in the system resource directory But the utool global cache allows for the user to override where the shelf for a specific app should be stored.

Returns real path to shelves directory

Return type log_dir_realpath (str)

utool.util_logging.**get_utool_logger**()

utool.util_logging.**is_logging**()

utool.util_logging.**start_logging**(*log_fpath=None*, *mode='a'*, *appname='default'*, *log_dir=None*)

Overwrites utool print functions to use a logger

CommandLine: python -m utool.util_logging –test-start_logging python -m utool.util_logging –test-start_logging:0 python -m utool.util_logging –test-start_logging:1 python -m utool.util_logging –test-start_logging:2

Example

```
>>> # DISABLE_DOCTEST
>>> import sys
>>> sys.argv.append('--verb-logging')
>>> import utool as ut
>>> ut.start_logging()
>>> ut.util_logging._utool_print()('hello world')
>>> ut.util_logging._utool_write()('writing1')
>>> ut.util_logging._utool_write()('writing2\n')
>>> ut.util_logging._utool_write()('writing3')
>>> ut.util_logging._utool_flush()()
>>> handler = ut.util_logging.__UTOOL_ROOT_LOGGER__.handlers[0]
>>> current_log_fpath = handler.stream.name
>>> current_log_text = ut.read_from(current_log_fpath)
>>> print('current_log_text =\n%s' % (current_log_text,))
>>> assert current_log_text.find('hello world') > 0, 'cant hello world'
>>> # assert current_log_text.find('writing1writing2') > 0, 'cant find␣
↪writing1writing2'
>>> assert current_log_text.find('writing3') > 0, 'cant find writing3'
```

Example

```
>>> # DISABLE_DOCTEST
>>> # Ensure that progress is logged
>>> import sys
>>> sys.argv.append('--verb-logging')
>>> import utool as ut
>>> ut.start_logging()
>>> [x for x in  ut.ProgressIter(range(0, 1000), freq=4)]
>>> handler = ut.util_logging.__UTOOL_ROOT_LOGGER__.handlers[0]
>>> current_log_fpath = handler.stream.name
>>> current_log_text = ut.read_from(current_log_fpath)
>>> assert current_log_text.find('rate') > 0, 'progress was not logged'
>>> print(current_log_text)
```

Example

```
>>> # DISABLE_DOCTEST
>>> import sys
>>> sys.argv.append('--verb-logging')
>>> import utool as ut
>>> ut.start_logging()
>>> ut.util_logging._utool_print()(u'\u0303')
>>> ut.util_logging._utool_flush()()
```

utool.util_logging.**stop_logging**()
Restores utool print functions to python defaults

utool.util_logging.**testlogprog**()
Test to ensure that all progress lines are outputed to the file logger while only a few progress lines are outputed to stdout. (if backspace is specified)

CommandLine: python -m utool.util_logging testlogprog –show –verb-logging python -m utool.util_logging testlogprog –show

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_logging import *  # NOQA
>>> import utool as ut
>>> result = testlogprog()
>>> print(result)
```

1.40 utool.util_num module

utool.util_num.**float_to_decimal**(*f*)
Convert a floating point number to a Decimal with no loss of information

References

http://docs.python.org/library/decimal.html#decimal-faq

utool.util_num.**get_sys_maxfloat**()

utool.util_num.**get_sys_maxint**()

utool.util_num.**get_sys_minint**()

utool.util_num.**int_comma_str**(*num*)

utool.util_num.**num2_sigfig**(*num*)

utool.util_num.**num_fmt**(*num*, *max_digits=None*)
Weird function. Not very well written. Very special case-y

Parameters

- **num** (*int or float*) –
- **max_digits** (*int*) –

Returns

Return type str

CommandLine: python -m utool.util_num –test-num_fmt

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_num import *  # NOQA
>>> # build test data
>>> num_list = [0, 0.0, 1.2, 1003232, 41431232., .0000000343, -.443243]
>>> max_digits = None
>>> # execute function
>>> result = [num_fmt(num, max_digits) for num in num_list]
>>> # verify results
>>> print(result)
['0', '0.0', '1.2', '1,003,232', '41431232.0', '0.0', '-0.443']
```

utool.util_num.**order_of_magnitude_ceil**(*num*)

utool.util_num.**sigfig_str**(*number*, *sigfig*)

References

http://stackoverflow.com/questions/2663612/nicely-repr-float-in-python

1.41 utool.util_numpy module

utool.util_numpy.**deterministic_sample**(*list_*, *nSample*, *seed=0*, *rng=None*, *strict=False*)
Grabs data randomly, but in a repeatable way

utool.util_numpy.**deterministic_shuffle**(*list_*, *seed=0*, *rng=None*)

Parameters

- **list** (*list*) –
- **seed** (*int*) –

Returns list

CommandLine: python -m utool.util_numpy –test-deterministic_shuffle

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_numpy import *  # NOQA
>>> list_ = [1, 2, 3, 4, 5, 6]
>>> seed = 1
>>> list_ = deterministic_shuffle(list_, seed)
>>> result = str(list_)
>>> print(result)
[3, 2, 5, 1, 4, 6]
```

utool.util_numpy.**ensure_rng**(*rng*, *impl='numpy'*)
Returns a random number generator

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_numpy import *  # NOQA
>>> import utool as ut
>>> import numpy as np
>>> num = 4
>>> print('--- Python as PYTHON ---')
>>> py_rng = random.Random(0)
>>> pp_nums = [py_rng.random() for _ in range(num)]
>>> print(pp_nums)
>>> print('--- Numpy as PYTHON ---')
>>> np_rng = ut.ensure_rng(random.Random(0), impl='numpy')
>>> np_nums = [np_rng.rand() for _ in range(num)]
>>> print(np_nums)
>>> print('--- Numpy as NUMPY---')
>>> np_rng = np.random.RandomState(seed=0)
>>> nn_nums = [np_rng.rand() for _ in range(num)]
>>> print(nn_nums)
>>> print('--- Python as NUMPY---')
>>> py_rng = ut.ensure_rng(np.random.RandomState(seed=0), impl='python')
>>> pn_nums = [py_rng.random() for _ in range(num)]
>>> print(pn_nums)
>>> assert np_nums == pp_nums
>>> assert pn_nums == nn_nums
```

utool.util_numpy.**inbounds**(*arr*, *low*, *high*)

utool.util_numpy.**index_of**(*item*, *array*)
index of [item] in [array]

utool.util_numpy.**intersect2d**(*A*, *B*)
intersect rows of 2d numpy arrays

DEPRICATE: use intersect2d in vtool instead

Parameters

- **A** (*ndarray[ndim=2]*) –
- **B** (*ndarray[ndim=2]*) –

Returns (C, Ax, Bx)

Return type tuple

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_numpy import *  # NOQA
>>> import utool as ut
>>> A = np.array([[1, 2, 3], [1, 1, 1]])
>>> B = np.array([[1, 2, 3], [1, 2, 14]])
>>> (C, Ax, Bx) = ut.intersect2d(A, B)
>>> result = str((C, Ax, Bx))
>>> print(result)
(array([[1, 2, 3]]), array([0]), array([0]))
```

utool.util_numpy.**make_incrementer**()

utool.util_numpy.**npfind**(*arr*)

utool.util_numpy.**quantum_random**()
returns a 32 bit unsigned integer quantum random number

utool.util_numpy.**random_indexes**(*max_index*, *subset_size=None*, *seed=None*, *rng=None*)
random unrepeated indicies

Parameters

- **max_index** –
- **subset_size** (*None*) – (default = None)
- **seed** (*None*) – (default = None)
- **rng** (*RandomState*) – random number generator(default = None)

Returns subst

Return type

?

CommandLine: python -m utool.util_numpy –exec-random_indexes

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_numpy import *  # NOQA
>>> max_index = 10
>>> subset_size = None
>>> seed = None
>>> rng = np.random.RandomState(0)
>>> subst = random_indexes(max_index, subset_size, seed, rng)
>>> result = ('subst = %s' % (str(subst),))
>>> print(result)
```

utool.util_numpy.**random_sample**(*list_*, *nSample*, *strict=False*, *rng=None*, *seed=None*)
Grabs data randomly

Parameters

- **list** (*list*) –
- **nSample** –
- **strict** (*bool*) – (default = False)
- **rng** (*module*) – random number generator(default = numpy.random)
- **seed** (*None*) – (default = None)

Returns sample_list

Return type list

CommandLine: python -m utool.util_numpy –exec-random_sample

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_numpy import *  # NOQA
>>> list_ = np.arange(10)
>>> nSample = 4
>>> strict = False
>>> rng = np.random.RandomState(0)
>>> seed = None
>>> sample_list = random_sample(list_, nSample, strict, rng, seed)
>>> result = ('sample_list = %s' % (str(sample_list),))
>>> print(result)
```

utool.util_numpy.**sample_domain**(*min_*, *max_*, *nSamp*, *mode='linear'*)

Example

```
>>> # ENABLE_DOCTEST
>>> import utool
>>> min_ = 10
>>> max_ = 1000
>>> nSamp  = 7
>>> result = utool.sample_domain(min_, max_, nSamp)
>>> print(result)
[10, 151, 293, 434, 576, 717, 859]
```

utool.util_numpy.**shuffle**(*list_*, *seed=0*, *rng=None*)

Shuffles a list inplace and then returns it for convinience

Parameters

- **list** (*list or ndarray*) – list to shuffl
- **rng** (*RandomState or int*) – seed or random number gen

Returns this is the input, but returned for convinience

Return type list

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_numpy import *  # NOQA
>>> list1 = [1, 2, 3, 4, 5, 6]
>>> list2 = shuffle(list(list1), rng=1)
>>> assert list1 != list2
>>> result = str(list2)
>>> print(result)
[3, 2, 5, 1, 4, 6]
```

utool.util_numpy.**spaced_indexes**(*len_*, *n*, *trunc=False*)

Returns n evenly spaced indexes. Returns as many as possible if trunc is true

utool.util_numpy.**spaced_items**(*list_*, *n*, ***kwargs*)

Returns n evenly spaced items

utool.util_numpy.**tiled_range**(*range_*, *cols*)

1.42 utool.util_parallel module

Module to executes the same function with different arguments in parallel.

class utool.util_parallel.**KillableProcess**(*group=None*, *target=None*, *name=None*, *args=()*, *kwargs={}*, ***, *daemon=None*)

Bases: multiprocessing.context.Process

Simple subclass of multiprocessing.Process Gives an additional method to kill all children as well as itself. calls this function on delete.

DEPRICATE, do not kill processes. It is not a good idea. It can cause deadlocks.

terminate2()

class utool.util_parallel.**KillableThread**(*group=None*, *target=None*, *name=None*, *args=()*, *kwargs=None*, ***, *daemon=None*)

Bases: threading.Thread

DEPRICATE, do not kill threads. It is not a good idea. It can cause deadlocks.

References

http://code.activestate.com/recipes/496960-thread2-killable-threads/ http://tomerfiliba.com/recipes/Thread2/

raise_exc(*excobj*)

terminate()

utool.util_parallel.**bgfunc**(*path*)

utool.util_parallel.**buffered_generator**(*source_gen*, *buffer_size=2*, *use_multiprocessing=False*)

Generator that runs a slow source generator in a separate process.

My generate function still seems faster on test cases. However, this function is more flexible in its compatability.

Parameters

- **source_gen** (*iterable*) – slow generator
- **buffer_size** (*int*) – the maximal number of items to pre-generate (length of the buffer) (default = 2)
- **use_multiprocessing** (*bool*) – if False uses GIL-hindered threading instead of multiprocessing (defualt = False).

Note: use_multiprocessing = True seems to freeze if passed in a generator built by six.moves.map.

References

Taken from Sander Dieleman's data augmentation pipeline https://github.com/benanne/kaggle-ndsb/blob/11a66cdbddee16c69514b9530a727df0ac6e136f/buffering.py

CommandLine: python -m utool.util_parallel –test-buffered_generator:0 python -m utool.util_parallel –test-buffered_generator:1

Ignore:

```
>>> #functime = timeit.timeit(
>>> # 'ut.is_prime(' + str(prime) + ')', setup='import utool as ut',
>>> # number=500) / 1000.0
```

Example

```
>>> # DISABLE_DOCTEST
>>> # UNSTABLE_DOCTEST
>>> from utool.util_parallel import *  # NOQA
>>> import utool as ut
>>> num = 2 ** 14
>>> func = ut.is_prime
>>> data = [38873] * num
>>> data = list(range(num))
>>> with ut.Timer('serial') as t1:
...     result1 = list(map(func, data))
>>> with ut.Timer('ut.generate2') as t3:
...     result3 = list(ut.generate2(func, zip(data), chunksize=2, quiet=1,␣
↪verbose=0))
>>> with ut.Timer('ut.buffered_generator') as t2:
...     result2 = list(ut.buffered_generator(map(func, data)))
>>> assert len(result1) == num and len(result2) == num and len(result3) == num
>>> assert result3 == result2, 'inconsistent results'
>>> assert result1 == result2, 'inconsistent results'
```

Example

```
>>> # DISABLE_DOCTEST
>>> # VERYSLLOOWWW_DOCTEST
>>> from utool.util_parallel import _test_buffered_generator
>>> _test_buffered_generator2()
```

utool.util_parallel.**generate2**(*func*, *args_gen*, *kw_gen=None*, *ntasks=None*, *ordered=True*, *force_serial=False*, *use_pool=False*, *chunksize=None*, *nprocs=None*, *progkw={}*, *nTasks=None*, *verbose=None*, *futures_threaded=True*, *timeout=3600*)

Interfaces to either multiprocessing or futures. Esentially maps `args_gen` onto `func` using pool.imap. However, args_gen must be a tuple of args that will be unpacked and send to the function. Thus, the function can take multiple args. Also specifing keyword args is supported.

Useful for embarrassingly parallel loops. Currently does not work with opencv3

CommandLine: python -m utool.util_parallel generate2

Parameters

- **func** (*function*) – live python function
- **args_gen** –
- **kw_gen** (*None*) – (default = None)
- **ntasks** (*None*) – (default = None)
- **ordered** (*bool*) – (default = True)
- **force_serial** (*bool*) – (default = False)

- **verbose** (*bool*) – verbosity flag(default = None)

CommandLine: python -m utool.util_parallel generate2

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_parallel import *  # NOQA
>>> from utool.util_parallel import _kw_wrap_worker  # NOQA
>>> import utool as ut
>>> args_gen = list(zip(range(10000)))
>>> kw_gen = [{}] * len(args_gen)
>>> func = ut.is_prime
>>> _ = list(generate2(func, args_gen))
>>> _ = list(generate2(func, args_gen, ordered=False))
>>> _ = list(generate2(func, args_gen, force_serial=True))
>>> _ = list(generate2(func, args_gen, use_pool=True))
>>> _ = list(generate2(func, args_gen, futures_threaded=True))
>>> _ = list(generate2(func, args_gen, ordered=False, verbose=False))
```

Example

```
>>> # ENABLE_DOCTEST
>>> import utool as ut
>>> #num = 8700  # parallel is slower for smaller numbers
>>> num = 500  # parallel has an initial (~.1 second startup overhead)
>>> print('TESTING SERIAL')
>>> func = ut.is_prime
>>> args_list = list(range(0, num))
>>> flag_generator0 = ut.generate2(ut.is_prime, zip(range(0, num)), force_
↪serial=True)
>>> flag_list0 = list(flag_generator0)
>>> print('TESTING PARALLEL (PROCESS)')
>>> flag_generator1 = ut.generate2(ut.is_prime, zip(range(0, num)))
>>> flag_list1 = list(flag_generator1)
>>> print('TESTING PARALLEL (THREAD)')
>>> flag_generator2 = ut.generate2(ut.is_prime, zip(range(0, num)), futures_
↪threaded=True)
>>> flag_list2 = list(flag_generator2)
>>> print('ASSERTING')
>>> assert len(flag_list1) == num
>>> assert len(flag_list2) == num
>>> assert flag_list0 == flag_list1
>>> assert flag_list0 == flag_list2
```

Example

```
>>> # ENABLE_DOCTEST
>>> # Trying to recreate the freeze seen in IBEIS
>>> import utool as ut
>>> print('TESTING SERIAL')
>>> flag_generator0 = ut.generate2(ut.is_prime, zip(range(0, 1)))
>>> flag_list0 = list(flag_generator0)
```

(continues on next page)

(continued from previous page)

```
>>> flag_generator1 = ut.generate2(ut.fibonacci_recursive, zip(range(0, 1)))
>>> flag_list1 = list(flag_generator1)
>>> print('TESTING PARALLEL')
>>> flag_generator2 = ut.generate2(ut.is_prime, zip(range(0, 12)))
>>> flag_list2 = list(flag_generator2)
>>> flag_generator3 = ut.generate2(ut.fibonacci_recursive, zip(range(0, 12)))
>>> flag_list3 = list(flag_generator3)
>>> print('flag_list0 = %r' % (flag_list0,))
>>> print('flag_list1 = %r' % (flag_list1,))
>>> print('flag_list2 = %r' % (flag_list1,))
>>> print('flag_list3 = %r' % (flag_list1,))
```

Example

```
>>> # DISABLE_DOCTEST
>>> # UNSTABLE_DOCTEST
>>> # Trying to recreate the freeze seen in IBEIS
>>> try:
>>>     import vtool as vt
>>> except ImportError:
>>>     import vtool as vt
>>> import utool as ut
>>> from wbia.algo.preproc.preproc_chip import gen_chip
>>> #from wbia.algo.preproc.preproc_feat import gen_feat_worker
>>> key_list = ['grace.jpg', 'easy1.png', 'ada2.jpg', 'easy3.png',
>>>             'hard3.png', 'zebra.png', 'patsy.jpg', 'ada.jpg',
>>>             'carl.jpg', 'lena.png', 'easy2.png']
>>> img_fpath_list = [ut.grab_test_imgpath(key) for key in key_list]
>>> arg_list1 = [(ut.augpath(img_fpath, '_testgen'), img_fpath, (0, 0, 100, 100),␣
↪0.0, (545, 372), []) for img_fpath in img_fpath_list[0:1]]
>>> arg_list2 = [(ut.augpath(img_fpath, '_testgen'), img_fpath, (0, 0, 100, 100),␣
↪0.0, (545, 372), []) for img_fpath in img_fpath_list]
>>> #arg_list3 = [(count, fpath, {}) for count, fpath in enumerate(ut.get_list_
↪column(arg_list1, 0))]
>>> #arg_list4 = [(count, fpath, {}) for count, fpath in enumerate(ut.get_list_
↪column(arg_list2, 0))]
>>> ut.remove_file_list(ut.get_list_column(arg_list2, 0))
>>> chips1 = [x for x in ut.generate2(gen_chip, arg_list1)]
>>> chips2 = [y for y in ut.generate2(gen_chip, arg_list2, force_serial=True)]
>>> #feats3 = [z for z in ut.generate2(gen_feat_worker, arg_list3)]
>>> #feats4 = [w for w in ut.generate2(gen_feat_worker, arg_list4)]
```

Example

```
>>> # DISABLE_DOCTEST
>>> # FAILING_DOCTEST
>>> # Trying to recreate the freeze seen in IBEIS
>>> # Extremely weird case: freezes only if dsize > (313, 313) AND __testwarp was␣
↪called beforehand.
>>> # otherwise the parallel loop works fine. Could be an opencv 3.0.0-dev issue.
>>> try:
>>>     import vtool as vt
>>> except ImportError:
```

(continues on next page)

(continued from previous page)

```
>>>     import vtool as vt
>>> import utool as ut
>>> from wbia.algo.preproc.preproc_chip import gen_chip
>>> import cv2
>>> from utool.util_parallel import __testwarp
>>> key_list = ['grace.jpg', 'easy1.png', 'ada2.jpg', 'easy3.png',
>>>             'hard3.png', 'zebra.png', 'patsy.jpg', 'ada.jpg',
>>>             'carl.jpg', 'lena.png', 'easy2.png']
>>> img_fpath_list = [ut.grab_test_imgpath(key) for key in key_list]
>>> arg_list1 = [(vt.imread(fpath),) for fpath in img_fpath_list[0:1]]
>>> arg_list2 = [(vt.imread(fpath),) for fpath in img_fpath_list]
>>> #new1 = [x for x in ut.generate2(__testwarp, arg_list1)]
>>> new1 =  __testwarp(arg_list1[0])
>>> new2 = [y for y in ut.generate2(__testwarp, arg_list2, force_serial=False)]
>>> #print('new2 = %r' % (new2,))
```

#Example: # >>> # Freakin weird. When IBEIS Runs generate it doesn't close the processes # >>> # UNSTABLE_DOCTEST # >>> # python -m utool.util_parallel –test-generate:4 # >>> # Trying to see if we can recreate the problem where there are # >>> # defunct python processes # >>> import utool as ut # >>> #num = 8700 # parallel is slower for smaller numbers # >>> num = 70000 # parallel has an initial (~.1 second startup overhead) # >>> print('TESTING PARALLEL') # >>> flag_generator1 = list(ut.generate2(ut.is_prime, range(0, num))) # >>> flag_generator1 = list(ut.generate2(ut.is_prime, range(0, num))) # >>> import time # >>> time.sleep(10)

utool.util_parallel.**get_default_numprocs**()

utool.util_parallel.**get_sys_thread_limit**()

utool.util_parallel.**in_main_process**()
Returns if you are executing in a multiprocessing subprocess Usefull to disable init print messages on windows

utool.util_parallel.**init_worker**()

utool.util_parallel.**set_num_procs**(*num_procs*)

utool.util_parallel.**spawn_background_daemon_thread**(*func*, **args*, ***kwargs*)

utool.util_parallel.**spawn_background_process**(*func*, **args*, ***kwargs*)
Run a function in the background (like rebuilding some costly data structure)

References

http://stackoverflow.com/questions/2046603/is-it-possible-to-run-function-in-a-subprocess-without-threading-or-writing-a-se http://stackoverflow.com/questions/1196074/starting-a-background-process-in-python http://stackoverflow.com/questions/15063963/python-is-thread-still-running

Parameters **func** (*function*) –

CommandLine: python -m utool.util_parallel –test-spawn_background_process

Example

```
>>> # DISABLE_DOCTEST
>>> # SLOW_DOCTEST
>>> from utool.util_parallel import *  # NOQA
>>> import utool as ut
>>> import time
```

(continues on next page)

(continued from previous page)

```
>>> from os.path import join
>>> # build test data
>>> fname = 'test_bgfunc_output.txt'
>>> dpath = ut.get_app_resource_dir('utool')
>>> ut.ensuredir(dpath)
>>> fpath = join(dpath, fname)
>>> # ensure file is not around
>>> sleep_time = 1
>>> ut.delete(fpath)
>>> assert not ut.checkpath(fpath, verbose=True)
>>> def backgrond_func(fpath, sleep_time):
...     import utool as ut
...     import time
...     print('[BG] Background Process has started')
...     time.sleep(sleep_time)
...     print('[BG] Background Process is writing')
...     ut.write_to(fpath, 'background process')
...     print('[BG] Background Process has finished')
...     #raise AssertionError('test exception')
>>> # execute function
>>> func = backgrond_func
>>> args = (fpath, sleep_time)
>>> kwargs = {}
>>> print('[FG] Spawning process')
>>> threadid = ut.spawn_background_process(func, *args, **kwargs)
>>> assert threadid.is_alive() is True, 'thread should be active'
>>> print('[FG] Spawned process. threadid=%r' % (threadid,))
>>> # background process should not have finished yet
>>> assert not ut.checkpath(fpath, verbose=True)
>>> print('[FG] Waiting to check')
>>> time.sleep(sleep_time + .1)
>>> print('[FG] Finished waiting')
>>> # Now the file should be there
>>> assert ut.checkpath(fpath, verbose=True)
>>> assert threadid.is_alive() is False, 'process should have died'
```

utool.util_parallel.**spawn_background_thread**(*func*, **args*, ***kwargs*)

1.43 utool.util_path module

python -c "import utool, doctest; print(doctest.testmod(utool.util_path))"

This module becomes nav

class utool.util_path.**ChdirContext**(*dpath=None*, *stay=False*, *verbose=None*)

Bases: object

References http://www.astropython.org/snippet/2009/10/chdir-context-manager

utool.util_path.**ancestor_paths**(*start=None*, *limit={}*)

All paths above you

utool.util_path.**assert_exists**(*path_*, *msg=''*, ***kwargs*)

Asserts that a patha exists

utool.util_path.**assertpath**(*path_*, *msg=''*, ***kwargs*)

Asserts that a patha exists

utool.util_path.**augpath**(*path*, *augsuf=''*, *augext=''*, *augpref=''*, *augdir=None*, *newext=None*, *newfname=None*, *ensure=False*, *prefix=None*, *suffix=None*)

augments end of path before the extension.

augpath

Parameters

- **path** (*str*) –
- **augsuf** (*str*) – augment filename before extension

Returns newpath

Return type str

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_path import *  # NOQA
>>> path = 'somefile.txt'
>>> augsuf = '_aug'
>>> newpath = augpath(path, augsuf)
>>> result = str(newpath)
>>> print(result)
somefile_aug.txt
```

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_path import *  # NOQA
>>> path = 'somefile.txt'
>>> augsuf = '_aug2'
>>> newext = '.bak'
>>> augdir = 'backup'
>>> newpath = augpath(path, augsuf, newext=newext, augdir=augdir)
>>> result = str(newpath)
>>> print(result)
backup/somefile_aug2.bak
```

utool.util_path.**basename_noext**(*path*)

utool.util_path.**checkpath**(*path_*, *verbose=False*, *n=None*, *info=False*)

verbose wrapper around os.path.exists

Returns true if path_ exists on the filesystem show only the top *n* directories

Parameters

- **path** (*str*) – path string
- **verbose** (*bool*) – verbosity flag(default = False)
- **n** (*int*) – (default = None)
- **info** (*bool*) – (default = False)

CommandLine: python -m utool.util_path –test-checkpath

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_path import *  # NOQA
>>> import utool as ut
>>> path_ = ut.__file__
>>> verbose = True
>>> n = None
>>> info = False
>>> result = checkpath(path_, verbose, n, info)
>>> print(result)
True
```

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_path import *  # NOQA
>>> import utool as ut
>>> path_ = ut.__file__ + 'foobar'
>>> verbose = True
>>> result = checkpath(path_, verbose, n=None, info=True)
>>> print(result)
False
```

utool.util_path.**copy**(*src*, *dst*, *overwrite=True*, *deeplink=True*, *verbose=True*, *dryrun=False*)

utool.util_path.**copy_all**(*src_dir*, *dest_dir*, *glob_str_list*, *recursive=False*)

utool.util_path.**copy_files_to**(*src_fpath_list*, *dst_dpath=None*, *dst_fpath_list=None*, *overwrite=False*, *verbose=True*, *veryverbose=False*)

parallel copier

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_path import *
>>> import utool as ut
>>> overwrite = False
>>> veryverbose = False
>>> verbose = True
>>> src_fpath_list = [ut.grab_test_imgpath(key)
>>>                   for key in ut.get_valid_test_imgkeys()]
>>> dst_dpath = ut.get_app_resource_dir('utool', 'filecopy_tests')
>>> copy_files_to(src_fpath_list, dst_dpath, overwrite=overwrite,
>>>               verbose=verbose)
```

utool.util_path.**copy_list**(*src_list*, *dst_list*, *lbl='Copying'*, *ioerr_ok=False*, *sherro_ok=False*, *oserror_ok=False*)

Copies all data and stat info

utool.util_path.**copy_single**(*src*, *dst*, *overwrite=True*, *verbose=True*, *deeplink=True*, *dryrun=False*)

Parameters

- **src** (*str*) – file or directory to copy

- **dst** (*str*) – directory or new file to copy to

Copies src file or folder to dst.

If src is a folder this copy is recursive.

utool.util_path.**delete**(*path*, *dryrun=False*, *recursive=True*, *verbose=None*, *print_exists=True*, *ignore_errors=True*)

Removes a file, directory, or symlink

utool.util_path.**dirsplit**(*path*)

Parameters **path** (*str*) –

Returns components of the path

Return type list

CommandLine: python -m utool.util_path –exec-dirsplit

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_path import *  # NOQA
>>> paths = []
>>> paths.append('E:/window file/foo')
>>> paths.append('/normal/foo')
>>> paths.append('~/relative/path')
>>> results = [dirsplit(path) for path in paths]
>>> import re
>>> results2 = [re.split('\\/', path) for path in paths]
>>> print(results2)
>>> result = ut.repr2(results)
>>> print(result)
```

utool.util_path.**ensure_crossplat_path**(*path*, *winroot='C:'*)

Parameters **path** (*str*) –

Returns crossplat_path

Return type str

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_path import *  # NOQA
>>> path = r'C:\somedir'
>>> cplat_path = ensure_crossplat_path(path)
>>> result = cplat_path
>>> print(result)
C:/somedir
```

utool.util_path.**ensure_ext**(*fpath*, *ext*, *replace=False*)

Parameters

- **fpath** (*str*) – file name or path
- **ext** (*str or list*) – valid extensions

- **replace** (*bool*) – if true all other extensions are removed. this removes all nonstarting characters including and after the first period. Otherwise only the trailing extension is kept

Returns fpath - file name or path

Return type str

CommandLine: python -m utool.util_path –exec-ensure_ext –show

Example

```
>>> # DISABLE_DOCTEST
>>> import utool as ut
>>> print(ut.ensure_ext('foo', '.bar'))
foo.bar
>>> print(ut.ensure_ext('foo.bar', '.bar'))
foo.bar
>>> print(ut.ensure_ext('foo.bar', '.baz'))
foo.bar.baz
>>> print(ut.ensure_ext('foo.bar', '.baz', True))
foo.baz
>>> print(ut.ensure_ext('foo.bar.baz', '.biz', True))
foo.biz
>>> print(ut.ensure_ext('..foo.bar.baz', '.biz', True))
..foo.biz
```

utool.util_path.**ensure_mingw_drive**(*win32_path*)
replaces windows drives with mingw style drives

Parameters **win32_path** (*str*) –

CommandLine: python -m utool.util_path –test-ensure_mingw_drive

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_path import *  # NOQA
>>> win32_path = r'C:/Program Files/Foobar'
>>> result = ensure_mingw_drive(win32_path)
>>> print(result)
/c/Program Files/Foobar
```

utool.util_path.**ensure_native_path**(*path*, *winroot='C:'*)

utool.util_path.**ensure_unixslash**(*path*)

utool.util_path.**ensuredir**(*path_*, *verbose=None*, *info=False*, *mode=1023*)
Ensures that directory will exist. creates new dir with sticky bits by default

Parameters

- **path** (*str*) – dpath to ensure. Can also be a tuple to send to join
- **info** (*bool*) – if True prints extra information
- **mode** (*int*) – octal mode of directory (default 0o1777)

Returns path - the ensured directory

Return type str

utool.util_path.**ensurefile**(*fpath*, *times=None*, *verbose=True*)
Creates file if it doesnt exist

Parameters

- **fpath** (*str*) – file path
- **times** (*None*) –
- **verbose** (*bool*) –

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_path import *  # NOQA
>>> fpath = '?'
>>> times = None
>>> verbose = True
>>> result = touch(fpath, times, verbose)
>>> print(result)
```

References

http://stackoverflow.com/questions/1158076/implement-touch-using-python

utool.util_path.**ensurepath**(*path_*, *verbose=None*)
DEPRICATE - alias - use ensuredir instead

utool.util_path.**existing_commonprefix**(*paths*)

utool.util_path.**existing_subpath**(*root_path*, *valid_subpaths*, *tiebreaker='first'*, *verbose=False*)
Returns join(root_path, subpath) where subpath in valid_subpath ane exists(subpath)

utool.util_path.**expand_win32_shortname**(*path1*)

utool.util_path.**file_bytes**(*fpath*)

Parameters **fpath** (*str*) – file path string

Returns size of file in bytes

Return type int

utool.util_path.**file_megabytes**(*fpath*)

Parameters **fpath** (*str*) – file path string

Returns size of file in megabytes

Return type float

utool.util_path.**find_lib_fpath**(*libname*, *root_dir*, *recurse_down=True*, *verbose=False*, *debug=False*)
Search for the library

utool.util_path.**fnames_to_fpaths**(*fname_list*, *path*)

utool.util_path.**fpath_has_ext**(*fname*, *exts*, *case_sensitive=False*)
returns true if the filename has any of the given extensions

utool.util_path.**fpath_has_imgext**(*fname*)
returns true if a filename matches an image pattern

utool.util_path.**fpaths_to_fnames**(*fpath_list*)

Parameters **fpath_list** (*list of strs*) – list of file-paths

Returns list of file-names

Return type fname_list (list of strs)

utool.util_path.**get_modname_from_modpath**(*module_fpath*)
returns importable name from file path

get_modname_from_modpath

Parameters **module_fpath** (*str*) – module filepath

Returns modname

Return type str

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_path import *  # NOQA
>>> import utool as ut
>>> module_fpath = ut.util_path.__file__
>>> modname = ut.get_modname_from_modpath(module_fpath)
>>> result = modname
>>> print(result)
utool.util_path
```

utool.util_path.**get_modpath**(*modname*, *prefer_pkg=False*, *prefer_main=False*)
Returns path to module

Parameters **modname** (*str or module*) – module name or actual module

Returns module_dir

Return type str

CommandLine: python -m utool.util_path –test-get_modpath

Setup:

```
>>> from utool.util_path import *  # NOQA
>>> import utool as ut
>>> utool_dir = dirname(dirname(ut.__file__))
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_path import *  # NOQA
>>> import utool as ut
>>> utool_dir = dirname(dirname(ut.__file__))
>>> modname = 'utool.util_path'
>>> module_dir = get_modpath(modname)
>>> result = ut.truepath_relative(module_dir, utool_dir)
```

(continues on next page)

(continued from previous page)

```
>>> result = ut.ensure_unixslash(result)
>>> print(result)
utool/util_path.py
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_path import *  # NOQA
>>> import utool as ut
>>> utool_dir = dirname(dirname(ut.__file__))
>>> modname = 'utool._internal'
>>> module_dir = get_modpath(modname, prefer_pkg=True)
>>> result = ut.ensure_unixslash(module_dir)
>>> print(result)
>>> assert result.endswith('utool/_internal')
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_path import *  # NOQA
>>> import utool as ut
>>> utool_dir = dirname(dirname(ut.__file__))
>>> modname = 'utool'
>>> module_dir = get_modpath(modname)
>>> result = ut.truepath_relative(module_dir, utool_dir)
>>> result = ut.ensure_unixslash(result)
>>> print(result)
utool/__init__.py
```

utool.util_path.**get_module_dir**(*module*, **args*)

utool.util_path.**get_module_subdir_list**(*module_fpath*)

Parameters **module_fpath** (*str*) –

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_path import *  # NOQA
>>> import utool as ut
>>> module_fpath = ut.util_path.__file__
>>> modsubdir_list = get_module_subdir_list(module_fpath)
>>> result = modsubdir_list
>>> print(result)
['utool', 'util_path']
```

utool.util_path.**get_nonconflicting_path**(*path_*, *dpath=None*, *offset=0*, *suffix=None*, *force_fmt=False*)

Searches for and finds a path garuenteed to not exist.

Parameters

- **path_** (*str*) – path string. If may include a "%" formatstr.

- **dpath** (*str*) – directory path(default = None)
- **offset** (*int*) – (default = 0)
- **suffix** (*None*) – (default = None)

Returns path string

Return type str

CommandLine: python -m utool.util_path non_existing_path

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_path import *  # NOQA
>>> import utool as ut
>>> base = ut.ensure_app_resource_dir('utool', 'tmp')
>>> ut.touch(base + '/tmp.txt')
>>> ut.touch(base + '/tmp0.txt')
>>> ut.delete(base + '/tmp1.txt')
>>> path_ = base + '/tmp.txt'
>>> newpath = ut.non_existing_path(path_)
>>> assert basename(newpath) == 'tmp1.txt'
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_path import *  # NOQA
>>> import utool as ut
>>> base = ut.ensure_app_resource_dir('utool', 'tmp')
>>> ut.ensurepath(base + '/dir_old')
>>> ut.ensurepath(base + '/dir_old0')
>>> ut.ensurepath(base + '/dir_old1')
>>> ut.delete(base + '/dir_old2')
>>> path_ = base + '/dir'
>>> suffix = '_old'
>>> newpath = ut.non_existing_path(path_, suffix=suffix)
>>> ut.assert_eq(basename(newpath), 'dir_old2')
```

utool.util_path.**get_path_type**(*path_*)
returns if a path is a file, directory, link, or mount

utool.util_path.**get_relative_modpath**(*module_fpath*)
Returns path to module relative to the package root

Parameters **module_fpath** (*str*) – module filepath

Returns modname

Return type str

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_path import *  # NOQA
>>> import utool as ut
>>> module_fpath = ut.util_path.__file__
>>> rel_modpath = ut.get_relative_modpath(module_fpath)
>>> rel_modpath = rel_modpath.replace('.pyc', '.py')  # allow pyc or py
>>> result = ensure_crossplat_path(rel_modpath)
>>> print(result)
utool/util_path.py
```

utool.util_path.**get_standard_exclude_dnames**()

utool.util_path.**get_standard_include_patterns**()

utool.util_path.**get_win32_short_path_name**(*long_name*)

Gets the short path name of a given long path.

References

http://stackoverflow.com/a/23598461/200291 http://stackoverflow.com/questions/23598289/get-win-short-fname-python

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_path import *  # NOQA
>>> import utool as ut  # NOQA
>>> # build test data
>>> #long_name = unicode(normpath(ut.get_resource_dir()))
>>> long_name = unicode(r'C:/Program Files (x86)')
>>> #long_name = unicode(r'C:/Python27')
#unicode(normpath(ut.get_resource_dir()))
>>> # execute function
>>> result = get_win32_short_path_name(long_name)
>>> # verify results
>>> print(result)
C:/PROGRA~2
```

utool.util_path.**glob**(*dpath*, *pattern=None*, *recursive=False*, *with_files=True*, *with_dirs=True*, *maxdepth=None*, *exclude_dirs=[]*, *fullpath=True*, ***kwargs*)

Globs directory for pattern

DEPRICATED: use pathlib.glob instead

Parameters

- **dpath** (*str*) – directory path or pattern
- **pattern** (*str or list*) – pattern or list of patterns (use only if pattern is not in dpath)
- **recursive** (*bool*) – (default = False)
- **with_files** (*bool*) – (default = True)
- **with_dirs** (*bool*) – (default = True)
- **maxdepth** (*None*) – (default = None)
- **exclude_dirs** (*list*) – (default = [])

Returns path_list

Return type list

SeeAlso: iglob

CommandLine: python -m utool.util_path –test-glob python -m utool.util_path –exec-glob:1

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_path import *  # NOQA
>>> from os.path import dirname
>>> import utool as ut
>>> dpath = dirname(ut.__file__)
>>> pattern = '__*.py'
>>> recursive = True
>>> with_files = True
>>> with_dirs = True
>>> maxdepth = None
>>> fullpath = False
>>> exclude_dirs = ['_internal', join(dpath, 'experimental')]
>>> print('exclude_dirs = ' + ut.repr2(exclude_dirs))
>>> path_list = glob(dpath, pattern, recursive, with_files, with_dirs,
>>>                  maxdepth, exclude_dirs, fullpath)
>>> path_list = sorted(path_list)
>>> result = ('path_list = %s' % (ut.repr3(path_list),))
>>> result = result.replace(r'\\', '/')
>>> print(result)
exclude_dirs = ['_internal', '...utool/utool/experimental']
path_list = [
    '__init__.py',
    '__main__.py',
    '_graveyard/__prefwidg_dep.py',
    'tests/__init__.py',
]
```

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_path import *  # NOQA
>>> import utool as ut
>>> dpath = dirname(ut.__file__) + '/__*.py'
>>> path_list = glob(dpath)
>>> result = ('path_list = %s' % (str(path_list),))
>>> print(result)
```

utool.util_path.**glob_python_modules**(*dirname*, ***kwargs*)

utool.util_path.**grep**(*regex_list*, *recursive=True*, *dpath_list=None*, *include_patterns=None*, *exclude_dirs=[]*, *greater_exclude_dirs=None*, *inverse=False*, *exclude_patterns=[]*, *verbose=False*, *fpath_list=None*, *reflags=0*, *cache=None*)

greps for patterns Python implementation of grep. NOT FINISHED

Parameters

- **regex_list** (*str or list*) – one or more patterns to find

- **recursive** (*bool*) –
- **dpath_list** (*list*) – directories to search (defaults to cwd)
- **include_patterns** (*list*) – defaults to standard file extensions

Returns (found_fpaths, found_lines_list, found_lxs_list)

Return type (list, list, list)

CommandLine: python -m utool.util_path –test-grep utprof.py -m utool.util_path –exec-grep utprof.py utool/util_path.py –exec-grep

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_path import *  # NOQA
>>> import utool as ut
>>> #dpath_list = [ut.truepath('~/code/utool/utool')]
>>> dpath_list = [ut.truepath(dirname(ut.__file__))]
>>> include_patterns = ['*.py']
>>> exclude_dirs = []
>>> regex_list = ['grepfile']
>>> verbose = True
>>> recursive = True
>>> result = ut.grep(regex_list, recursive, dpath_list, include_patterns,
>>>                  exclude_dirs)
>>> (found_fpath_list, found_lines_list, found_lxs_list) = result
>>> assert 'util_path.py' in list(map(basename, found_fpath_list))
```

utool.util_path.**grepfile**(*fpath*, *regexpr_list*, *reflags=0*, *cache=None*)

grepfile - greps a specific file

Parameters

- **fpath** (*str*) –
- **regexpr_list** (*list or str*) – pattern or list of patterns

Returns list of lines and list of line numbers

Return type tuple (list, list)

CommandLine: python -m utool.util_path –exec-grepfile

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_path import *  # NOQA
>>> import utool as ut
>>> fpath = ut.get_modpath(ut.util_path)
>>> regexpr_list = ['foundthisline', '__future__']
>>> cache = None
>>> reflags = 0
>>> found_lines, found_lxs = ut.grepfile(fpath, regexpr_list)
>>> result = ut.repr3({'found_lines': found_lines, 'found_lxs': found_lxs})
>>> print(result)
>>> assert 7 in found_lxs
```

(continues on next page)

(continued from previous page)

```
>>> others = ut.take_complement(found_lxs, [found_lxs.index(7)])
>>> assert others[0] == others[1]
```

utool.util_path.**greplines**(*lines*, *regexpr_list*, *reflags=0*)

grepfile - greps a specific file

TODO: move to util_str, rework to be core of grepfile

utool.util_path.**iglob**(*dpath*, *pattern=None*, *recursive=False*, *with_files=True*, *with_dirs=True*, *maxdepth=None*, *exclude_dirs=[]*, *fullpath=True*, ***kwargs*)

Iteratively globs directory for pattern

FIXME: This function has a speed issue

Parameters

- **dpath** (*str*) – directory path
- **pattern** (*str*) –
- **recursive** (*bool*) – (default = False)
- **with_files** (*bool*) – (default = True)
- **with_dirs** (*bool*) – (default = True)
- **maxdepth** (*None*) – (default = None)
- **exclude_dirs** (*list*) – (default = [])

Yields path

References

http://stackoverflow.com/questions/19859840/excluding-dirs-in-os-walk

utool.util_path.**is_module_dir**(*path*)

Parameters **path** (*str*) –

Returns True if path contains an __init__ file

Return type flag

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_path import *  # NOQA
>>> path = truepath('~/code/utool/utool')
>>> flag = is_module_dir(path)
>>> result = (flag)
>>> print(result)
```

utool.util_path.**is_private_module**(*path*)

utool.util_path.**is_python_module**(*path*)

utool.util_path.**list_images**(*img_dpath_*, *ignore_list=[]*, *recursive=False*, *fullpath=False*, *full=None*, *sort=True*)

Returns a list of images in a directory. By default returns relative paths.

TODO: rename to ls_images TODO: Change all instances of fullpath to full

Parameters

- **img_dpath** (*str*) –
- **ignore_list** (*list*) – (default = [])
- **recursive** (*bool*) – (default = False)
- **fullpath** (*bool*) – (default = False)
- **full** (*None*) – (default = None)
- **sort** (*bool*) – (default = True)

Returns gname_list

Return type list

CommandLine: python -m utool.util_path –exec-list_images

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_path import *  # NOQA
>>> img_dpath_ = '?'
>>> ignore_list = []
>>> recursive = False
>>> fullpath = False
>>> full = None
>>> sort = True
>>> gname_list = list_images(img_dpath_, ignore_list, recursive,
>>>                          fullpath, full, sort)
>>> result = ('gname_list = %s' % (str(gname_list),))
>>> print(result)
```

utool.util_path.**longest_existing_path**(*_path*)

Returns the longest root of _path that exists

Parameters **_path** (*str*) – path string

Returns _path - path string

Return type str

CommandLine: python -m utool.util_path –exec-longest_existing_path

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_path import *  # NOQA
>>> import utool as ut
>>> target = dirname(ut.__file__)
>>> _path = join(target, 'nonexist/foobar')
>>> existing_path = longest_existing_path(_path)
>>> result = ('existing_path = %s' % (str(existing_path),))
>>> print(result)
>>> assert existing_path == target
```

utool.util_path.**ls**(*path*, *pattern='*'*)
like unix ls - lists all files and dirs in path

utool.util_path.**ls_dirs**(*path*, *pattern='*'*)

utool.util_path.**ls_images**(*img_dpath_*, *ignore_list=[]*, *recursive=False*, *fullpath=False*, *full=None*, *sort=True*)
Returns a list of images in a directory. By default returns relative paths.

TODO: rename to ls_images TODO: Change all instances of fullpath to full

Parameters

- **img_dpath** (*str*) –
- **ignore_list** (*list*) – (default = [])
- **recursive** (*bool*) – (default = False)
- **fullpath** (*bool*) – (default = False)
- **full** (*None*) – (default = None)
- **sort** (*bool*) – (default = True)

Returns gname_list

Return type list

CommandLine: python -m utool.util_path –exec-list_images

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_path import *  # NOQA
>>> img_dpath_ = '?'
>>> ignore_list = []
>>> recursive = False
>>> fullpath = False
>>> full = None
>>> sort = True
>>> gname_list = list_images(img_dpath_, ignore_list, recursive,
>>>                          fullpath, full, sort)
>>> result = ('gname_list = %s' % (str(gname_list),))
>>> print(result)
```

utool.util_path.**ls_moduledirs**(*path*, *private=True*, *full=True*)
lists all dirs which are python modules in path

utool.util_path.**ls_modulefiles**(*path*, *private=True*, *full=True*, *noext=False*)

utool.util_path.**make_grep_resultstr**(*grep_result*, *extended_regex_list*, *reflags*, *colored=True*)

utool.util_path.**matching_fpaths**(*dpath_list*, *include_patterns*, *exclude_dirs=[]*, *greater_exclude_dirs=[]*, *exclude_patterns=[]*, *recursive=True*)
walks dpath lists returning all directories that match the requested pattern.

Parameters

- **dpath_list** (*list*) –
- **include_patterns** (*str*) –

- **exclude_dirs** (*None*) –
- **recursive** (*bool*) –

References

TODO: fix names and behavior of exclude_dirs and greater_exclude_dirs http://stackoverflow.com/questions/19859840/excluding-directories-in-os-walk

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_path import *  # NOQA
>>> import utool as ut
>>> dpath_list = [dirname(dirname(ut.__file__))]
>>> include_patterns = get_standard_include_patterns()
>>> exclude_dirs = ['_page']
>>> greater_exclude_dirs = get_standard_exclude_dnames()
>>> recursive = True
>>> fpath_gen = matching_fpaths(dpath_list, include_patterns, exclude_dirs,
>>>                             greater_exclude_dirs, recursive)
>>> result = list(fpath_gen)
>>> print('\n'.join(result))
```

utool.util_path.**move**(*src*, *dst*, *verbose=True*)

utool.util_path.**move_list**(*src_list*, *dst_list*, *lbl='Moving'*, *verbose=True*)

utool.util_path.**newcd**(*path*)
DEPRICATE

utool.util_path.**non_existing_path**(*path_*, *dpath=None*, *offset=0*, *suffix=None*, *force_fmt=False*)
Searches for and finds a path garuenteed to not exist.

Parameters

- **path_** (*str*) – path string. If may include a "%" formatstr.
- **dpath** (*str*) – directory path(default = None)
- **offset** (*int*) – (default = 0)
- **suffix** (*None*) – (default = None)

Returns path string

Return type str

CommandLine: python -m utool.util_path non_existing_path

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_path import *  # NOQA
>>> import utool as ut
>>> base = ut.ensure_app_resource_dir('utool', 'tmp')
```

(continues on next page)

(continued from previous page)

```
>>> ut.touch(base + '/tmp.txt')
>>> ut.touch(base + '/tmp0.txt')
>>> ut.delete(base + '/tmp1.txt')
>>> path_ = base + '/tmp.txt'
>>> newpath = ut.non_existing_path(path_)
>>> assert basename(newpath) == 'tmp1.txt'
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_path import *  # NOQA
>>> import utool as ut
>>> base = ut.ensure_app_resource_dir('utool', 'tmp')
>>> ut.ensurepath(base + '/dir_old')
>>> ut.ensurepath(base + '/dir_old0')
>>> ut.ensurepath(base + '/dir_old1')
>>> ut.delete(base + '/dir_old2')
>>> path_ = base + '/dir'
>>> suffix = '_old'
>>> newpath = ut.non_existing_path(path_, suffix=suffix)
>>> ut.assert_eq(basename(newpath), 'dir_old2')
```

utool.util_path.**num_images_in_dir**(*path*)
returns the number of images in a directory

utool.util_path.**path_ndir_split**(*path_*, *n*, *force_unix=True*, *winroot='C:'*, *trailing=True*)
Shows only a little bit of the path. Up to the n bottom-level directories

TODO: rename to path_tail? ndir_split?

Returns (str) the trailing n paths of path.

CommandLine: python3 -m utool.util_path –test-path_ndir_split python3 -m utool –tf path_ndir_split python -m utool –tf path_ndir_split

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_path import *  # NOQA
>>> import utool as ut
>>> paths = [r'/usr/bin/local/foo/bar',
...          r'C:/',
...          #r'lonerel',
...          #r'reldir/other',
...          r'/ham',
...          r'./eggs',
...          r'/spam/eggs',
...          r'C:\Program Files (x86)/foobar/bin']
>>> N = 2
>>> iter_ = ut.iprod(paths, range(1, N + 1))
>>> force_unix = True
>>> tuplist = [(n, path_ndir_split(path_, n)) for path_, n in iter_]
>>> chunklist = list(ut.ichunks(tuplist, N))
>>> list_ = [['n=%r: %s' % (x, ut.reprfunc(y)) for x, y in chunk]
```

(continues on next page)

(continued from previous page)

```
>>>          for chunk in chunklist]
>>> line_list = [', '.join(strs) for strs in list_]
>>> result = '\n'.join(line_list)
>>> print(result)
n=1: '.../bar', n=2: '.../foo/bar'
n=1: 'C:/', n=2: 'C:/'
n=1: '.../ham', n=2: '/ham'
n=1: '.../eggs', n=2: './eggs'
n=1: '.../eggs', n=2: '.../spam/eggs'
n=1: '.../bin', n=2: '.../foobar/bin'
```

utool.util_path.**pathsplit_full**(*path*)
splits all directories in path into a list

utool.util_path.**platform_path**(*path*)
Returns platform specific path for pyinstaller usage

Parameters **path** (*str*) –

Returns path2

Return type str

CommandLine: python -m utool.util_path –test-platform_path

Ignore:

```
>>> # ENABLE_DOCTEST
>>> # FIXME: find examples of the wird paths this fixes (mostly on win32 i␣
↪think)
>>> from utool.util_path import *  # NOQA
>>> import utool as ut
>>> path = 'some/odd/../weird/path'
>>> path2 = platform_path(path)
>>> result = str(path2)
>>> if ut.WIN32:
...     ut.assert_eq(path2, r'some\weird\path')
... else:
...     ut.assert_eq(path2, r'some/weird/path')
```

Ignore:

```
>>> # ENABLE_DOCTEST
>>> from utool.util_path import *  # NOQA
>>> import utool as ut    # NOQA
>>> if ut.WIN32:
...     path = 'C:/PROGRA~2'
...     path2 = platform_path(path)
...     assert path2 == u'..\\..\\..\\..\\Program Files (x86)'
```

utool.util_path.**relpath_unix**(*path*, *otherpath*)

utool.util_path.**remove_broken_links**(*dpath*, *verbose=True*)
Removes all broken links in a directory

Parameters **dpath** (*str*) – directory path

Returns num removed

Return type int

References

http://stackoverflow.com/questions/20794/find-broken-symlinks-with-python

CommandLine: python -m utool remove_broken_links:0

Example

```
>>> # DISABLE_DOCTEST
>>> # SCRIPT
>>> from utool.util_path import *  # NOQA
>>> remove_broken_links('.')
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_path import *  # NOQA
>>> import utool as ut
>>> dpath = ut.ensure_app_resource_dir('utool', 'path_tests')
>>> ut.delete(dpath)
>>> test_dpath = ut.ensuredir(join(dpath, 'testdpath'))
>>> test_fpath = ut.ensurefile(join(dpath, 'testfpath.txt'))
>>> flink1 = ut.symlink(test_fpath, join(dpath, 'flink1'))
>>> dlink1 = ut.symlink(test_fpath, join(dpath, 'dlink1'))
>>> assert len(ut.ls(dpath)) == 4
>>> ut.delete(test_fpath)
>>> assert len(ut.ls(dpath)) == 3
>>> remove_broken_links(dpath)
>>> ut.delete(test_dpath)
>>> remove_broken_links(dpath)
>>> assert len(ut.ls(dpath)) == 0
```

utool.util_path.**remove_dirs**(*dpath*, *verbose=None*, *ignore_errors=True*, *dryrun=False*, *quiet=False*)

Recursively removes a single directory (need to change function name)

DEPRICATE

Parameters

- **dpath** (*str*) – directory path
- **dryrun** (*bool*) – (default = False)
- **ignore_errors** (*bool*) – (default = True)
- **quiet** (*bool*) – (default = False)

Returns False

Return type bool

CommandLine: python -m utool.util_path –test-remove_dirs

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_path import *  # NOQA
>>> import utool as ut
>>> dpath = ut.ensure_app_resource_dir('utool', 'testremovedir')
>>> assert exists(dpath), 'nothing to remove'
>>> flag = remove_dirs(dpath, verbose=True)
>>> print('flag = %r' % (flag,))
>>> assert not exists(dpath), 'did not remove dpath'
>>> assert flag is True
```

utool.util_path.**remove_existing_fpaths**(*fpath_list*, *verbose=False*, *quiet=False*, *strict=False*, *print_caller=False*, *lbl='files'*)

checks existance before removing. then tries to remove exisint paths

utool.util_path.**remove_file**(*fpath*, *verbose=None*, *ignore_errors=True*, *dryrun=False*, *quiet=False*)

Removes a file

utool.util_path.**remove_file_list**(*fpaths*, *verbose=False*, *quiet=False*, *strict=False*, *print_caller=False*, *lbl='files'*)

Removes multiple file paths

utool.util_path.**remove_files_in_dir**(*dpath*, *fname_pattern_list='*'*, *recursive=False*, *verbose=False*, *dryrun=False*, *ignore_errors=False*)

Removes files matching a pattern from a directory

utool.util_path.**remove_fpaths**(*fpaths*, *verbose=False*, *quiet=False*, *strict=False*, *print_caller=False*, *lbl='files'*)

Removes multiple file paths

utool.util_path.**sanitize_filename**(*fname*)

utool.util_path.**search_candidate_paths**(*candidate_path_list*, *candidate_name_list=None*, *priority_paths=None*, *required_subpaths=[]*, *verbose=None*)

searches for existing paths that meed a requirement

Parameters

- **candidate_path_list** (*list*) – list of paths to check. If candidate_name_list is specified this is the dpath list instead
- **candidate_name_list** (*list*) – specifies several names to check (default = None)
- **priority_paths** (*None*) – specifies paths to check first. Ignore candidate_name_list (default = None)
- **required_subpaths** (*list*) – specified required directory structure (default = [])
- **verbose** (*bool*) – verbosity flag(default = True)

Returns return_path

Return type str

CommandLine: python -m utool.util_path –test-search_candidate_paths

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_path import *  # NOQA
>>> candidate_path_list = [ut.truepath('~/RPI/code/utool'),
>>>                        ut.truepath('~/code/utool')]
>>> candidate_name_list = None
>>> required_subpaths = []
>>> verbose = True
>>> priority_paths = None
>>> return_path = search_candidate_paths(candidate_path_list,
>>>                                      candidate_name_list,
>>>                                      priority_paths, required_subpaths,
>>>                                      verbose)
>>> result = ('return_path = %s' % (str(return_path),))
>>> print(result)
```

utool.util_path.**search_in_dirs**(*fname*, *search_dpaths=[]*, *shortcircuit=True*, *return_tried=False*, *strict=False*)

Parameters

- **fname** (*str*) – file name
- **search_dpaths** (*list*) –
- **shortcircuit** (*bool*) –
- **return_tried** (*bool*) – return tried paths
- **strict** (*bool*) – (default = False)

Returns None

Return type fpath

Example

```
>>> # DISABLE_DOCTEST
>>> import utool as ut
>>> fname = 'Inno Setup 5\\ISCC.exe'
>>> search_dpaths = ut.get_install_dirs()
>>> shortcircuit = True
>>> fpath = ut.search_in_dirs(fname, search_dpaths, shortcircuit)
>>> print(fpath)
```

utool.util_path.**sed**(*regexpr*, *repl*, *force=False*, *recursive=False*, *dpath_list=None*, *fpath_list=None*, *verbose=None*, *include_patterns=None*, *exclude_patterns=[]*)

Python implementation of sed. NOT FINISHED

searches and replaces text in files

Parameters

- **regexpr** (*str*) – regx patterns to find
- **repl** (*str*) – text to replace
- **force** (*bool*) –
- **recursive** (*bool*) –

- **dpath_list** (*list*) – directories to search (defaults to cwd)

utool.util_path.**sedfile**(*fpath*, *regexpr*, *repl*, *force=False*, *verbose=True*, *veryverbose=False*)

Executes sed on a specific file

Parameters

- **fpath** (*str*) – file path string
- **regexpr** (*str*) –
- **repl** (*str*) –
- **force** (*bool*) – (default = False)
- **verbose** (*bool*) – verbosity flag(default = True)
- **veryverbose** (*bool*) – (default = False)

Returns changed_lines

Return type list

CommandLine: python -m utool.util_path –exec-sedfile –show

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_path import *  # NOQA
>>> import utool as ut
>>> fpath = ut.get_modpath(ut.util_path)
>>> regexpr = 'sedfile'
>>> repl = 'saidfile'
>>> force = False
>>> verbose = True
>>> veryverbose = False
>>> changed_lines = sedfile(fpath, regexpr, repl, force, verbose, veryverbose)
>>> result = ('changed_lines = %s' % (ut.repr3(changed_lines),))
>>> print(result)
```

utool.util_path.**symlink**(*real_path*, *link_path*, *overwrite=False*, *on_error='raise'*, *verbose=2*)

Attempt to create a symbolic link.

Todo: Can this be fixed on windows?

Parameters

- **path** (*str*) – path to real file or directory
- **link_path** (*str*) – path to desired location for symlink
- **overwrite** (*bool*) – overwrite existing symlinks (default = False)
- **on_error** (*str*) – strategy for dealing with errors. raise or ignore
- **verbose** (*int*) – verbosity level (default=2)

Returns link path

Return type str

CommandLine: python -m utool.util_path symlink

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_path import *  # NOQA
>>> import utool as ut
>>> dpath = ut.get_app_resource_dir('utool')
>>> real_path = join(dpath, 'real_file.txt')
>>> link_path = join(dpath, 'link_file.txt')
>>> ut.emap(ut.delete, [real_path, link_path], verbose=0)
>>> ut.writeto(real_path, 'foo')
>>> result = symlink(real_path, link_path)
>>> assert ut.readfrom(result) == 'foo'
>>> ut.emap(ut.delete, [real_path, link_path], verbose=0)
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_path import *  # NOQA
>>> import utool as ut
>>> real_dpath = ut.get_app_resource_dir('utool', 'real_dpath')
>>> link_dpath = ut.augpath(real_dpath, newfname='link_dpath')
>>> real_path = join(real_dpath, 'afile.txt')
>>> link_path = join(link_dpath, 'afile.txt')
>>> ut.emap(ut.delete, [real_path, link_path], verbose=0)
>>> ut.ensuredir(real_dpath)
>>> ut.writeto(real_path, 'foo')
>>> result = symlink(real_dpath, link_dpath)
>>> assert ut.readfrom(link_path) == 'foo'
>>> ut.delete(link_dpath, verbose=0)
>>> assert ut.checkpath(real_path)
>>> ut.delete(real_dpath, verbose=0)
>>> assert not ut.checkpath(real_path)
```

utool.util_path.**tail**(*fpath*, *n=2*, *trailing=True*)
Alias for path_ndir_split

utool.util_path.**testgrep**()
utprof.py -m utool.util_path –exec-testgrep

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_path import *  # NOQA
>>> import utool as ut
>>> #dpath_list = [ut.truepath('~/code/utool/utool')]
>>> dpath_list = [ut.truepath(dirname(ut.__file__))]
>>> include_patterns = ['*.py']
>>> exclude_dirs = []
>>> regex_list = ['grepfile']
>>> verbose = True
>>> recursive = True
```

(continues on next page)

(continued from previous page)

```
>>> result = ut.grep(regex_list, recursive, dpath_list, include_patterns,
>>>                  exclude_dirs)
>>> (found_fpath_list, found_lines_list, found_lxs_list) = result
>>> assert 'util_path.py' in list(map(basename, found_fpath_list))
```

utool.util_path.**touch**(*fpath*, *times=None*, *verbose=True*)

Creates file if it doesnt exist

Parameters

- **fpath** (*str*) – file path
- **times** (*None*) –
- **verbose** (*bool*) –

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_path import *  # NOQA
>>> fpath = '?'
>>> times = None
>>> verbose = True
>>> result = touch(fpath, times, verbose)
>>> print(result)
```

References

http://stackoverflow.com/questions/1158076/implement-touch-using-python

utool.util_path.**truepath_relative**(*path*, *otherpath=None*)

Normalizes and returns absolute path with so specs

Parameters

- **path** (*str*) – path to file or directory
- **otherpath** (*None*) – (default = None)

Returns str

CommandLine: python -m utool.util_path –exec-truepath_relative –show

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_path import *  # NOQA
>>> import utool as ut
>>> path = 'C:/foobar/foobiz'
>>> otherpath = 'C:/foobar'
>>> path_ = truepath_relative(path, otherpath)
>>> result = ('path = %s' % (ut.repr2(path_),))
>>> print(result)
path = 'foobiz'
```

utool.util_path.**unexpanduser**(*path*)
Replaces home directory with '~'

utool.util_path.**win_shortcut**(*source*, *link_name*)
Attempt to create windows shortcut TODO: TEST / FIXME

References

http://stackoverflow.com/questions/1447575/symlinks-on-windows

1.44 utool.util_print module

class utool.util_print.**Indenter**(*lbl=' '*, *enabled=True*)
Bases: object

Monkey patches modules injected with print to change the way print behaves.

Works with utool.inject to allow for prefixing of all within-context print statements in a semi-dynamic manner. There seem to be some bugs but it works pretty well.

CommandLine: python -m utool.util_print –exec-Indenter

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_print import *  # NOQA
>>> import utool as ut
>>> ut.util_print._test_indent_print()
```

start()

stop()

utool.util_print.**colorprint**(*text*, *color=None*)
provides some color to terminal output

Parameters

- **text** (*str*) –
- **color** (*str*) –

Ignore: assert color in ['', 'yellow', 'blink', 'lightgray', 'underline', 'darkyellow', 'blue', 'darkblue', 'faint', 'fuchsia', 'black', 'white', 'red', 'brown', 'turquoise', 'bold', 'darkred', 'darkgreen', 'reset', 'standout', 'darkteal', 'darkgray', 'overline', 'purple', 'green', 'teal', 'fuscia']

CommandLine: python -c "import pygments.console; print(list(pygments.console.codes.keys()))" python -m utool.util_print –exec-colorprint python -m utool.util_print –exec-colorprint:1

import pygments print(ut.repr4(list(pygments.formatters.get_all_formatters()))) print(list(pygments.styles.get_all_styles()))

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_print import *  # NOQA
>>> import pygments.console
>>> msg_list = list(pygments.console.codes.keys())
>>> color_list = list(pygments.console.codes.keys())
>>> [colorprint(text, color) for text, color in zip(msg_list, color_list)]
```

Example

```
>>> # DISABLE_DOCTEST (Windows test)
>>> from utool.util_print import *  # NOQA
>>> import pygments.console
>>> print('line1')
>>> colorprint('line2', 'red')
>>> colorprint('line3', 'blue')
>>> colorprint('line4', 'fuchsia')
>>> colorprint('line5', 'reset')
>>> colorprint('line5', 'fuchsia')
>>> print('line6')
```

utool.util_print.**cprint**(*text*, *color=None*)

provides some color to terminal output

Parameters

- **text** (*str*) –
- **color** (*str*) –

Ignore: assert color in [‘’, ‘yellow’, ‘blink’, ‘lightgray’, ‘underline’, ‘darkyellow’, ‘blue’, ‘darkblue’, ‘faint’, ‘fuchsia’, ‘black’, ‘white’, ‘red’, ‘brown’, ‘turquoise’, ‘bold’, ‘darkred’, ‘darkgreen’, ‘reset’, ‘standout’, ‘darkteal’, ‘darkgray’, ‘overline’, ‘purple’, ‘green’, ‘teal’, ‘fuscia’]

CommandLine: python -c “import pygments.console; print(list(pygments.console.codes.keys()))” python -m utool.util_print –exec-colorprint python -m utool.util_print –exec-colorprint:1

import pygments print(ut.repr4(list(pygments.formatters.get_all_formatters()))) print(list(pygments.styles.get_all_styles()))

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_print import *  # NOQA
>>> import pygments.console
>>> msg_list = list(pygments.console.codes.keys())
>>> color_list = list(pygments.console.codes.keys())
>>> [colorprint(text, color) for text, color in zip(msg_list, color_list)]
```

Example

```
>>> # DISABLE_DOCTEST (Windows test)
>>> from utool.util_print import *  # NOQA
>>> import pygments.console
>>> print('line1')
>>> colorprint('line2', 'red')
>>> colorprint('line3', 'blue')
>>> colorprint('line4', 'fuchsia')
>>> colorprint('line5', 'reset')
>>> colorprint('line5', 'fuchsia')
>>> print('line6')
```

utool.util_print.**dictprint**(*dict_*, *dict_name=None*, ***kwargs*)

utool.util_print.**horiz_print**(**args*)

utool.util_print.**printNOTQUIET**(*msg*)

utool.util_print.**printVERBOSE**(*msg*, *verbarg*)

utool.util_print.**printWARN**(*msg*)

utool.util_print.**print_code**(*text*, *lexer_name='python'*)

Parameters **text** (*str*) –

CommandLine: python -m utool.util_print –test-print_python_code

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_print import *  # NOQA
>>> import utool as ut
>>> # build test data
>>> text = ut.read_from(ut.__file__.replace('.pyc', '.py'))
>>> # execute function
>>> print_python_code(text)
```

utool.util_print.**print_dict**(*dict_*, *dict_name=None*, ***kwargs*)

utool.util_print.**print_difftext**(*text*, *other=None*)

Parameters **text** (*str*) –

CommandLine: #python -m utool.util_print –test-print_difftext #autopep8 ingest_data.py –diff | python -m utool.util_print –test-print_difftext

utool.util_print.**print_filesize**(*fpath*)

utool.util_print.**print_list**(*list_*, ***kwargs*)

utool.util_print.**print_locals**(**args*, ***kwargs*)

Prints local variables in function.

If no arguments all locals are printed.

Variables can be specified directly (variable values passed in) as varargs or indirectly (variable names passed in) in kwargs by using keys and a list of strings.

utool.util_print.**print_python_code**(*text*)

SeeAlso: print_code

utool.util_print.**printdict**(*dict_*, *dict_name=None*, ***kwargs*)

utool.util_print.**printif**(*func*, *condition=False*)
execute printfunc only if condition=QUIET

utool.util_print.**printshape**(*arr_name*, *locals_*)

1.45 utool.util_profile module

Cleaning script for the output of utool profiling

Removes profiled output of code that never ran

class utool.util_profile.**Profiler**
Bases: object

get_output()

utool.util_profile.**clean_line_profile_text**(*text*)
Sorts the output from line profile by execution time Removes entries which were not run

utool.util_profile.**clean_lprof_file**(*input_fname*, *output_fname=None*)
Reads a .lprof file and cleans it

utool.util_profile.**dump_profile_text**()

utool.util_profile.**fix_rawprofile_blocks**(*profile_block_list*)

utool.util_profile.**get_block_id**(*block*)

utool.util_profile.**get_block_totaltime**(*block*)

utool.util_profile.**get_profile_text**(*profile*)

utool.util_profile.**get_summary**(*profile_block_list*, *maxlines=20*)

References

https://github.com/rkern/line_profiler

utool.util_profile.**make_profiler**()

utool.util_profile.**parse_rawprofile_blocks**(*text*)
Split the file into blocks along delimters and and put delimeters back in the list

utool.util_profile.**parse_timemap_from_blocks**(*profile_block_list*)
Build a map from times to line_profile blocks

1.46 utool.util_progress module

progress handler.

Old progress funcs needto be depricated ProgressIter and ProgChunks are pretty much the only useful things here.

utool.util_progress.**ProgChunks**(*list_*, *chunksize*, *nInput=None*, ***kwargs*)
Yeilds an iterator in chunks and computes progress Progress version of ut.ichunks

Parameters

- **list** (*list*) –

- **chunksize** –
- **nInput** (*None*) – (default = None)

Kwargs: length, freq

Returns ProgressIter

CommandLine: python -m utool.util_progress ProgChunks –show

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_progress import *  # NOQA
>>> import utool as ut
>>> list_ = range(100)
>>> chunksize = 10
>>> nInput = None
>>> progiter_ = ProgChunks(list_, chunksize, nInput)
>>> iter_ = iter(progiter_)
>>> chunk = six.next(iter_)
>>> assert len(chunk) == 10
>>> rest = ut.flatten(list(progiter_))
>>> assert len(rest) == 90
```

class utool.util_progress.**ProgIter**(*iterable*, *lbl='Prog'*, *adjust=True*, *freq=1*, *bs=True*, ***kwargs*)

Bases: *utool.util_progress.ProgressIter*

Thin wrapper with better arg positions

utool.util_progress.**ProgPartial**(**args*, ***kwargs*)

class utool.util_progress.**ProgressIter**(*iterable=None*, **args*, ***kwargs*)

Bases: object

Wraps a for loop with progress reporting

lbl='Progress: ', length=0, flushfreq=4, startafter=-1, start=True, repl=False, approx=False, disable=False, writefreq=1, with_time=False, backspace=True, pad_stdout=False, wfreq=None, ffreq=None, freq=None, total=None, num=None, with_totaltime=None

References: https://github.com/verigak/progress/blob/master/progress/__init__.py

Parameters

- **()** (*iterable*) – iterable normally passed to for loop
- **lbl** (*str*) – progress label
- **length** (*int*) –
- **flushfreq** (*int*) –
- **startafter** (*int*) –
- **start** (*bool*) –
- **repl** (*bool*) –
- **approx** (*bool*) –

- **enabled** (*bool*) –
- **writefreq** (*int*) –
- **with_totaltime** (*bool*) –
- **backspace** (*bool*) –
- **pad_stdout** (*bool*) –
- **autoadjust** (*bool*) – no adjusting frequency if True (default False)
- **wfreq** (*None*) – alias for write_freq
- **ffreq** (*None*) – alias for flush_freq
- **total** (*None*) – alias for length
- **num** (*None*) – alias for length

Timeit:

```
>>> import utool as ut
>>> setup = ut.codeblock(
>>> '''
>>> import utool as ut
>>> from six.moves import range, zip
>>> import time
>>> def time_append(size):
>>>     start_time    = time.time()
>>>     last_time     = start_time
>>>     list2 = []
>>>     for x in range(size):
>>>         now_time    = time.time()
>>>         between = now_time - last_time
>>>         last_time   = now_time
>>>         list2.append(between)
>>>
>>> def time_assign(size):
>>>     start_time    = time.time()
>>>     last_time     = start_time
>>>     list1 = ut.alloc_nones(size)
>>>     for x in range(size):
>>>         now_time    = time.time()
>>>         between = now_time - last_time
>>>         last_time   = now_time
>>>         list1[x] = between
>>>
>>> def time_baseline(size):
>>>     start_time    = time.time()
>>>     last_time     = start_time
>>>     for x in range(size):
>>>         now_time    = time.time()
>>>         between = now_time - last_time
>>>         last_time   = now_time
>>>
>>> def time_null(size):
>>>     for x in range(size):
>>>         pass
>>> ''')
>>>
```

(continues on next page)

(continued from previous page)

```
>>> input_sizes = [2 ** count for count in range(7, 12)]
>>> stmt_list = ['time_assign', 'time_append', 'time_baseline', 'time_null']
>>> input_sizes=[100, 1000, 10000]
>>> ut.timeit_grid(stmt_list, setup, input_sizes=input_sizes, show=True)
```

CommandLine: python -m utool.util_progress –test-ProgressIter python -m utool.util_progress –test-ProgressIter:0 python -m utool.util_progress –test-ProgressIter:1 python -m utool.util_progress –test-ProgressIter:2 python -m utool.util_progress –test-ProgressIter:3

Example

```
>>> # ENABLE_DOCTEST
>>> import utool as ut
>>> from six.moves import range
>>> num = 1000
>>> num2 = 10001
>>> results1 = [x for x in ut.ProgressIter(range(num), wfreq=10, adjust=True)]
>>> results4 = [x for x in ut.ProgressIter(range(num), wfreq=1, adjust=True)]
>>> results2 = [x for x in range(num)]
>>> results3 = [x for x in ut.progiter((y + 1 for y in range(num2)),
>>>                                    ntotal=num2, wfreq=1000,
>>>                                    backspace=True, adjust=True)]
>>> assert results1 == results2
```

Example

```
>>> # DISABLE_DOCTEST
>>> # SLOW_DOCTEST
>>> import utool as ut
>>> from six.moves import range
>>> num2 = 10001
>>> progiter = ut.ProgressIter(range(num2), lbl='testing primes',
>>>                            report_unit='seconds', freq=1,
>>>                            time_thresh=.1, adjust=True)
>>> [ut.get_nth_prime_bruteforce(29) for x in progiter]
```

Example

```
>>> # DISABLE_DOCTEST
>>> # SLOW_DOCTEST
>>> import utool as ut
>>> from six.moves import range
>>> num2 = 100001
>>> progiter = ut.ProgressIter(range(num2), lbl='testing primes',
>>>                            report_unit='seconds', freq=1,
>>>                            time_thresh=3, adjust=True, bs=True)
>>> [ut.get_nth_prime_bruteforce(29) for x in progiter]
```

Example

```
>>> # DISABLE_DOCTEST
>>> # SLOW_DOCTEST
>>> import utool as ut
>>> from six.moves import range
>>> import time
>>> crazy_time_list = [.001, .01, .0001] * 1000
>>> crazy_time_iter = (time.sleep(x) for x in crazy_time_list)
>>> progiter = ut.ProgressIter(crazy_time_iter, lbl='crazy times',␣
↪length=len(crazy_time_list), freq=10)
>>> list(progiter)
```

static build_msg_fmtstr2(*lbl*, *length*, *invert_rate*, *backspace*)

Parameters

- **lbl** (*str*) –
- **invert_rate** (*bool*) –
- **backspace** (*bool*) –

Returns msg_fmtstr_time

Return type str

CommandLine: python -m utool.util_progress –exec-ProgressIter.build_msg_fmtstr2

Setup:

```
>>> from utool.util_progress import *  # NOQA
>>> lbl = 'foo'
>>> invert_rate = True
>>> backspace = False
>>> length = None
```

Example

```
>>> # DISABLE_DOCTEST
>>> msg_fmtstr_time = ProgressIter.build_msg_fmtstr2(lbl, length, invert_rate,
↪ backspace)
>>> result = ('%s' % (ut.repr2(msg_fmtstr_time),))
>>> print(result)
```

static build_msg_fmtstr_head_cols(*length*, *lbl*)

display_message()

ensure_newline()

use before any custom printing when using the progress iter to ensure your print statement starts on a new line instead of at the end of a progress line

iter_rate()

pun not intended

TODO: record iteration times for analysis # TODO Incorporate this better # FIXME; pad_stdout into subfunctions

import dis dis.dis(ut.ProgressIter.iter_rate)

set_extra(*extra*)
specify a custom info appended to the end of the next message TODO: come up with a better name and rename

utool.util_progress.**get_num_chunks**(*length*, *chunksize*)
Returns the number of chunks that a list will be split into given a chunksize.

Parameters

- **length** (*int*) –
- **chunksize** (*int*) –

Returns n_chunks

Return type int

CommandLine: python -m utool.util_progress –exec-get_num_chunks:0

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_progress import *  # NOQA
>>> length = 2000
>>> chunksize = 256
>>> n_chunks = get_num_chunks(length, chunksize)
>>> result = ('n_chunks = %s' % (six.text_type(n_chunks),))
>>> print(result)
n_chunks = 8
```

utool.util_progress.**log_progress**(*lbl='Progress: '*, *length=0*, *flushfreq=4*, *startafter=-1*, *start=True*, *repl=False*, *approx=False*, *disable=False*, *writefreq=1*, *with_time=False*, *backspace=True*, *pad_stdout=False*, *wfreq=None*, *ffreq=None*, *freq=None*, *total=None*, *num=None*, *with_totaltime=None*)
DEPRICATE FIXME: depricate for ProgressIter. still used in util_dev

utool.util_progress.**progiter**
alias of *utool.util_progress.ProgressIter*

utool.util_progress.**progress_str**(*max_val*, *lbl='Progress: '*, *repl=False*, *approx=False*, *backspace=True*)
makes format string that prints progress: %Xd/MAX_VAL with backspaces

NOTE: r can be used instead of backspaces. This function is not very relevant because of that.

utool.util_progress.**test_progress**()

CommandLine: python -m utool.util_progress –test-test_progress

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_progress import *  # NOQA
>>> test_progress()
```

1.47 utool.util_project module

Ignore: ~/local/init/REPOS1.py

class utool.util_project.**GrepResult**(*found_fpath_list*, *found_lines_list*, *found_lxs_list*, *extended_regex_list*, *reflags*)

Bases: *utool.util_dev.NiceRepr*

hack_remove_pystuff()

inplace_filter_results(*filter_pat*)

make_resultstr(*colored=True*)

pattern_filterflags(*filter_pat*)

remove_results(*indicies*)

class utool.util_project.**SetupRepo**

Bases: object

Maybe make a new interface to SetupRepo?

Example

```
>>> # DISABLE_DOCTEST
>>> # SCRIPT
>>> from utool.util_project import *  # NOQA
>>> import utool as ut
>>> self = SetupRepo()
>>> print(self)
```

all()

ensure_text(*fname*, *text*, ***kwargs*)

main()

python -m utool SetupRepo.main –modname=sklearn –repo=scikit-learn –codedir=~/code -w python -m utool SetupRepo.main –repo=ubelt –codedir=~/code –modname=ubelt -w

Example

```
>>> # DISABLE_DOCTEST
>>> # SCRIPT
>>> from utool.util_project import *  # NOQA
>>> SetupRepo().main()
```

class utool.util_project.**UserProfile**(*name=None*)

Bases: *utool.util_dev.NiceRepr*

glob(**args*, ***kwargs*)

Ignore:

```
>>> # Ensure that .gitignore has certain lines
>>> git_ignore_lines = [
>>>     'timeings.txt'
>>> ]
```

(continues on next page)

(continued from previous page)

```
>>> fpath_list = profile.glob('.gitignore', recursive=False)
>>> for fpath in fpath_list:
>>>     lines = ut.readfrom(fpath, verbose=False).split('\n')
>>>     lines = [line.strip() for line in lines]
>>>     missing = ut.setdiff(git_ignore_lines, lines)
>>>     if missing:
>>>         print('fpath = %r' % (fpath,))
>>>         ut.writeto(fpath, '\n'.join(lines + missing))
```

grep(**args*, ***kwargs*)

utool.util_project.**ensure_text**(*fname*, *text*, *repo_dpath='.'*, *force=None*, *locals_={}*, *chmod=None*)

Parameters

- **fname** (*str*) – file name
- **text** (*str*) –
- **repo_dpath** (*str*) – directory path string(default = '.')
- **force** (*bool*) – (default = False)
- **locals** (*dict*) – (default = {})

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_project import *  # NOQA
>>> import utool as ut
>>> result = setup_repo()
>>> print(result)
```

utool.util_project.**ensure_user_profile**(*user_profile=None*)

Parameters **user_profile** (UserProfile) – (default = None)

Returns user_profile

Return type *UserProfile*

CommandLine: python -m utool.util_project –exec-ensure_user_profile –show

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_project import *  # NOQA
>>> import utool as ut
>>> user_profile = None
>>> user_profile = ensure_user_profile(user_profile)
>>> result = ('user_profile = %s' % (ut.repr2(user_profile),))
>>> print(ut.repr3(user_profile.project_dpaths))
>>> print(result)
```

utool.util_project.**glob_projects**(*pat*, *user_profile=None*, *recursive=True*)

def testenv(modname, funcname): ut.import_modname(modname) exec(ut.execstr_funckw(table.get_rowid), globals())

Ignore:

```
>>> import utool as ut
>>> ut.testenv('utool.util_project', 'glob_projects', globals())
>>> from utool.util_project import *  # NOQA
```

utool.util_project.**grep_projects**(*tofind_list*, *user_profile=None*, *verbose=True*, *new=False*, ***kwargs*)

Greps the projects defined in the current UserProfile

Parameters

- **tofind_list** (*list*) –
- **user_profile** (*None*) – (default = None)

Kwargs: user_profile

CommandLine: python -m utool –tf grep_projects grep_projects

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_project import *  # NOQA
>>> import utool as ut
>>> import sys
>>> tofind_list = ut.get_argval('--find', type_=list,
>>>                             default=[sys.argv[-1]])
>>> grep_projects(tofind_list)
```

utool.util_project.**sed_projects**(*regexpr*, *repl*, *force=False*, *recursive=True*, *user_profile=None*, ***kwargs*)

Parameters

- **regexpr** –
- **repl** –
- **force** (*bool*) – (default = False)
- **recursive** (*bool*) – (default = True)
- **user_profile** (*None*) – (default = None)

CommandLine: python -m utool.util_project –exec-sed_projects

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_project import *  # NOQA
>>> regexpr = ut.get_argval('--find', type_=str, default=sys.argv[-1])
>>> repl = ut.get_argval('--repl', type_=str, default=sys.argv[-2])
>>> force = False
>>> recursive = True
>>> user_profile = None
```

(continues on next page)

(continued from previous page)

```
>>> result = sed_projects(regexpr, repl, force, recursive, user_profile)
>>> print(result)
```

Ignore: regexpr = 'annotation match_scores' repl = 'draw_annot_scoresep'

utool.util_project.**setup_repo**()
Creates default structure for a new repo

CommandLine: python -m utool setup_repo –repo=dtool –codedir=~/code

python -m utool setup_repo –repo=dtool –codedir=~/code python -m utool setup_repo –repo=wbia-flukematch-module –codedir=~/code –modname=wbia_flukematch python -m utool setup_repo –repo=mtgmonte –codedir=~/code –modname=mtgmonte python -m utool setup_repo –repo=pydarknet –codedir=~/code –modname=pydarknet python -m utool setup_repo –repo=sandbox_utools –codedir=~/code –modname=sandbox_utools

python -m utool setup_repo –repo=ubelt –codedir=~/code –modname=ubelt -w

python -m utool setup_repo

Python: ipython import utool as ut ut.rrrr(0); ut.setup_repo()

Example

```
>>> # DISABLE_DOCTEST
>>> # SCRIPT
>>> from utool.util_project import *  # NOQA
>>> import utool as ut
>>> result = setup_repo()
>>> print(result)
```

utool.util_project.**wbia_user_profile**()

1.48 utool.util_regex module

in vim nongreedy .* is .{-} in python nongreedy .* is .*?

utool.util_regex.**backref_field**(*key*)

utool.util_regex.**bref_field**(*key*)

utool.util_regex.**convert_text_to_varname**(*text*)

Parameters **text** (*str*) – text that might not be a valid variablename

Returns varname

Return type str

CommandLine: python -m utool.util_regex –test-convert_text_to_varname

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_regex import *  # NOQA
>>> text = '0) View Application-Files Directory. '
>>> varname = convert_text_to_varname(text)
>>> result = ('varname = %s' % (str(varname),))
>>> print(result)
_0_View_ApplicationFiles_Directory_
```

utool.util_regex.**extend_regex**(*regexpr*)

Extends the syntax of regular expressions by replacing convineince syntax with re friendly syntax. Nameely things that I use in vim like <>

utool.util_regex.**extend_regex2**(*regexpr*, *reflags=0*)
also preprocesses flags

utool.util_regex.**extend_regex3**(*regex_list*, *reflags=0*)

utool.util_regex.**get_match_text**(*match*)

utool.util_regex.**modify_quoted_strs**(*text*, *modify_func=None*)
doesnt work with escaped quotes or multilines single quotes only. no nesting.

Parameters

- **text** –
- **modify_func** (*None*) –

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_regex import *  # NOQA
>>> text = "'just' 'a' sentance with 'strings' in it "
>>> text2 = "'this' 'text' wont work 'because \'of \"the\"\' \"nesting\"'"
>>> text3 = " ''' god \"help\" you ''' if you use 'triple quotes'  "
>>> def modify_func(quoted_str):
...     return quoted_str.upper()
>>> result = modify_quoted_strs(text, modify_func)
>>> print(result)
'JUST' 'A' sentance with 'STRINGS'
```

utool.util_regex.**named_field**(*key*, *regex*, *vim=False*)
Creates a named regex group that can be referend via a backref. If key is None the backref is referenced by number.

References

https://docs.python.org/2/library/re.html#regular-expression-syntax

utool.util_regex.**named_field_regex**(*keypat_tups*)

Parameters

- **keypat_tups** (*list*) – tuples of (name, pattern) or a string for an unnamed
- **pattern** –

Returns regex

Return type str

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_regex import *  # NOQA
>>> keypat_tups = [
...     ('name',  r'G\d+'),  # species and 2 numbers
...     ('under', r'_'),     # 2 more numbers
...     ('id',    r'\d+'),   # 2 more numbers
...     ( None,   r'\.'),
...     ('ext',   r'\w+'),
... ]
>>> regex = named_field_regex(keypat_tups)
>>> result = (regex)
>>> print(result)
(?P<name>G\d+)(?P<under>_)(?P<id>\d+)(\.)(?P<ext>\w+)
```

utool.util_regex.**named_field_repl**(*field_list*)

Parameters **field_list** (*list*) – list of either a tuples to denote a keyword, or a strings for relacement t3ext

Returns repl for regex

Return type str

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_regex import *  # NOQA
>>> field_list = [('key',), 'unspecial string']
>>> repl = named_field_repl(field_list)
>>> result = repl
>>> print(result)
\g<key>unspecial string
```

utool.util_regex.**negative_lookahead**(*regex*, *vim=False*)

utool.util_regex.**negative_lookbehind**(*regex*, *vim=False*)

Parameters **regex** –

Returns

Return type

?

CommandLine: python -m utool.util_regex –exec-negative_lookbehind

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_regex import *  # NOQA
>>> regex = re.escape('\'')
```

(continues on next page)

(continued from previous page)

```
>>> pattern = negative_lookbehind(regex) + 'foo'
>>> match1 = re.search(pattern, '\'foo\'')
>>> match2 = re.search(pattern, '"foo"')
>>> match3 = re.search(pattern, '\' foo\'')
>>> match4 = re.search(pattern, '" foo"')
```

utool.util_regex.**nongreedy_kleene_star**(*vim=False*)

utool.util_regex.**padded_parse**(*pattern*, *text*)

utool.util_regex.**parse_docblock**(*func_code*)

#TODO: Finish me

References

http://pyparsing.wikispaces.com/share/view/1264103 http://code.activestate.com/recipes/576704-python-code-minifier/

Example

```
>>> # DISABLE_DOCTEST
>>> import utool as ut
>>> import inspect
>>> func_code = inspect.getsource(ut.modify_quoted_strs)
```

utool.util_regex.**parse_python_syntax**(*text*)

step1: split lines

step2: parse enclosure pairity for each line to find unended lines

for each unending line, is there a valid merge line? (a line that could snytatically finish this runnon statement? If no then error. Else try to join the two lines.

step3: perform context_sensitive_edit

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_regex import *   # NOQA
>>> import utool
>>> from os.path import normpath
>>> text = utool.read_from(utool.util_regex.__file__)
```

utool.util_regex.**positive_lookahead**(*regex*, *vim=False*)

utool.util_regex.**positive_lookbehind**(*regex*, *vim=False*)

utool.util_regex.**regex_get_match**(*regex*, *text*, *fromstart=False*)

utool.util_regex.**regex_matches**(*regex*, *text*, *fromstart=True*)

utool.util_regex.**regex_or**(*list_*)

utool.util_regex.**regex_parse**(*regex*, *text*, *fromstart=True*)

Parameters

- **regex** (*str*) –
- **text** (*str*) –
- **fromstart** (*bool*) –

Returns

Return type dict or None

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_regex import *  # NOQA
>>> regex = r'(?P<string>\'[^\']*\')'
>>> text = " 'just' 'a' sentance with 'strings' in it "
>>> fromstart = False
>>> result = regex_parse(regex, text, fromstart)['string']
>>> print(result)
```

utool.util_regex.**regex_replace**(*regex*, *repl*, *text*)

thin wrapper around re.sub regex_replace

MULTILINE and DOTALL are on by default in all util_regex functions

Parameters

- **regex** (*str*) – pattern to find
- **repl** (*str*) – replace pattern with this
- **text** (*str*) – text to modify

Returns modified text

Return type str

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_regex import *  # NOQA
>>> regex = r'\(.*\):'
>>> repl = '(*args)'
>>> text = '''def foo(param1,
...                   param2,
...                   param3):'''
>>> result = regex_replace(regex, repl, text)
>>> print(result)
def foo(*args)
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_regex import *  # NOQA
>>> import utool as ut
>>> regex = ut.named_field_regex([('keyword', 'def'), ' ', ('funcname', '.*'), r
↪'\(.*\):'])
```

(continues on next page)

(continued from previous page)

```
>>> repl = ut.named_field_repl([('funcname',), ('keyword',)])
>>> text = '''def foo(param1,
...                   param2,
...                   param3):'''
>>> result = regex_replace(regex, repl, text)
>>> print(result)
foodef
```

utool.util_regex.**regex_replace_lines**(*lines*, *regexpat*, *replpat*)

utool.util_regex.**regex_search**(*regex*, *text*)

utool.util_regex.**regex_split**(*regex*, *text*)

utool.util_regex.**regex_word**(*w*)

utool.util_regex.**whole_word**(*regex*)

1.49 utool.util_resources module

utool.util_resources.**available_memory**()
Returns total system wide available memory in bytes

utool.util_resources.**current_memory_usage**()
Returns this programs current memory usage in bytes

utool.util_resources.**get_matching_process_ids**(*cmd_pattern*, *user_pattern*)

CommandLine: export PID=30196 export PID=$(python -c "import utool as ut; print(ut.get_matching_process_ids('jonc', 'python2.7'))") export PID=$(python -c "import utool as ut; print(ut.get_matching_process_ids('jonc', 'matlab'))") sudo -H echo $PID ps -o pid,comm,nice -p $PID renice 10 -p $PID sudo renice -4 -p $PID

user_pattern = 'jonc' cmd_pattern = 'main.py' user_pattern = None cmd_pattern = 'matlab' get_matching_process_ids(cmd_pattern, user_pattern)

utool.util_resources.**get_memstats_str**()

utool.util_resources.**get_python_datastructure_sizes**()

References

http://stackoverflow.com/questions/1331471/in-memory-size-of-python-stucture

CommandLine: python -m utool.util_resources –test-get_python_datastructure_sizes

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_resources import *  # NOQA
>>> import utool as ut  # NOQA
>>> type_sizes = get_python_datastructure_sizes()
>>> result = ut.repr4(type_sizes, sorted_=True)
>>> print(result)
```

utool.util_resources.**get_resource_limits**()

utool.util_resources.**get_resource_usage_str**()

utool.util_resources.**memstats**()

utool.util_resources.**num_cpus**()

utool.util_resources.**num_unused_cpus**(*thresh=10*)
Returns the number of cpus with utilization less than *thresh* percent

utool.util_resources.**peak_memory**()
Returns the resident set size (the portion of a process's memory that is held in RAM.)

utool.util_resources.**print_resource_usage**()

utool.util_resources.**time_in_systemmode**()

utool.util_resources.**time_in_usermode**()

utool.util_resources.**time_str2**(*seconds*)

utool.util_resources.**total_memory**()
Returns total system wide memory in bytes

utool.util_resources.**used_memory**()
Returns total system wide used memory in bytes

1.50 utool.util_set module

class utool.util_set.**OrderedSet**(*iterable=None*)
Bases: collections.abc.MutableSet

Set the remembers the order elements were added Big-O running times for all methods are the same as for regular sets. The internal self._map dictionary maps keys to links in a doubly linked list. The circular doubly linked list starts and ends with a sentinel element. The sentinel element never gets deleted (this simplifies the algorithm). The prev/next links are weakref proxies (to prevent circular references). Individual links are kept alive by the hard reference in self._map. Those hard references disappear when a key is deleted from an OrderedSet.

References

http://code.activestate.com/recipes/576696/ http://code.activestate.com/recipes/576694/ http://stackoverflow.com/questions/1653970/does-python-have-an-ordered-set

add(*key*)
Store new key in a new link at the end of the linked list

append(*key*)
Alias for add

discard(*key*)
Remove an element. Do not raise an exception if absent.

index(*item*)
Find the index of *item* in the OrderedSet

Example

```
>>> # ENABLE_DOCTEST
>>> import utool as ut
>>> self = ut.oset([1, 2, 3])
>>> assert self.index(1) == 0
>>> assert self.index(2) == 1
>>> assert self.index(3) == 2
>>> ut.assert_raises(ValueError, self.index, 4)
```

pop(*last=True*)
Return the popped value. Raise KeyError if empty.

classmethod union(**sets*)

```
>>> from utool.util_set import *  # NOQA
```

update(*other*)
union update

utool.util_set.**oset**
alias of *utool.util_set.OrderedSet*

1.51 utool.util_setup module

utool.util_setup.**NOOP**()

class utool.util_setup.**SETUP_PATTERNS**
Bases: object

chmod_test = ['test_*.py']

clutter_cyth = ['_*_cyth.o', '_*_cyth_bench.py', 'run_cyth_benchmarks.sh', '_*_cyth.c'

clutter_pybuild = ['*.pyc', '*.pyo']

class utool.util_setup.**SetupManager**
Bases: object

Helps with writing setup.py

get_cmdclass()

register_command(*name*)

utool.util_setup.**assert_in_setup_repo**(*setup_fpath*, *name=''*)
pass in __file__ from setup.py

utool.util_setup.**autogen_sphinx_apidoc**()
autogen_sphinx_docs.py

Ignore: C:Python27Scriptsautogen_sphinx_docs.py autogen_sphinx_docs.py

pip uninstall sphinx pip install sphinx pip install sphinxcontrib-napoleon pip install sphinx –upgrade pip install sphinxcontrib-napoleon –upgrade

cd C:Python27Scripts ls C:Python27Scripts

python -c "import sphinx; print(sphinx.__version__)"

CommandLine: python -m utool.util_setup –exec-autogen_sphinx_apidoc

Example

```
>>> # DISABLE_DOCTEST
>>> # SCRIPT
>>> from utool.util_setup import *   # NOQA
>>> autogen_sphinx_apidoc()
```

utool.util_setup.**build_pyo**(*project_dirs*)

utool.util_setup.**clean**(*setup_dir*, *clutter_patterns*, *clutter_dirs*)

utool.util_setup.**find_ext_modules**(*disable_warnings=True*)

utool.util_setup.**find_packages**(*recursive=True*, *maxdepth=None*)
Finds all directories with an __init__.py file in them

utool.util_setup.**get_cmdclass**()
DEPRICATE

utool.util_setup.**get_numpy_include_dir**()

utool.util_setup.**parse_author**()
TODO: this function should parse setup.py or a module for the author variable

utool.util_setup.**parse_package_for_version**(*name*)
Searches for a variable named __version__ in name's __init__.py file and returns the value. This function parses the source text. It does not load the module.

utool.util_setup.**parse_readme**(*readmefile='README.md'*)

utool.util_setup.**presetup**(*setup_fpath*, *kwargs*)

utool.util_setup.**presetup_commands**(*setup_fpath*, *kwargs*)

utool.util_setup.**read_license**(*license_file*)

utool.util_setup.**setup_chmod**(*setup_fpath*, *setup_dir*, *chmod_patterns*)
Gives files matching pattern the same chmod flags as setup.py

utool.util_setup.**setuptools_setup**(*setup_fpath=None*, *module=None*, ***kwargs*)

1.52 utool.util_six module

1.53 utool.util_sqlite module

class utool.util_sqlite.**SQLColumnRichInfo**(*column_id*, *name*, *type_*, *notnull*, *dflt_value*, *pk*)
Bases: tuple

column_id
Alias for field number 0

dflt_value
Alias for field number 4

name
Alias for field number 1

notnull
Alias for field number 3

pk
Alias for field number 5

type_
Alias for field number 2

utool.util_sqlite.**get_nonprimary_columninfo**(*cur*, *tablename*)

utool.util_sqlite.**get_primary_columninfo**(*cur*, *tablename*)

utool.util_sqlite.**get_table_column**(*cur*, *tablename*, *colname*)
Conveinience:

utool.util_sqlite.**get_table_columninfo_list**(*cur*, *tablename*)
Returns a list of tuples with the following format: [0] column_id : id of the column [1] name : the name of the column [2] type_ : the type of the column (TEXT, INT, etc...) [3] notnull : 0 or 1 if the column can contains null values [4] dflt_value : the default value [5] pk : 0 or 1 if the column partecipate to the primary key

Parameters **tablename** (*str*) – table name

Returns list of tuples

Return type column_list

References

http://stackoverflow.com/questions/17717829/how-to-get-column-names-from-a-table-in-sqlite-via-pragma-net-c

CommandLine: python -m utool.util_sqlite –test-get_table_columninfo_list

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_sqlite import *  # NOQA
```

utool.util_sqlite.**get_table_columnname_list**(*cur*, *tablename*)

utool.util_sqlite.**get_table_columns**(*cur*, *tablename*, *exclude_columns=[]*)

utool.util_sqlite.**get_table_csv**(*cur*, *tablename*, *exclude_columns=[]*)
Conveinience: Converts a tablename to csv format

Parameters

- **tablename** (*str*) –
- **exclude_columns** (*list*) –

Returns csv_table

Return type str

CommandLine: python -m wbia.control.SQLDatabaseControl –test-get_table_csv

Example

```
>>> # DISABLE_DOCTEST
>>> from wbia.control.SQLDatabaseControl import *  # NOQA
>>> # build test data
>>> import wbia
>>> ibs = wbia.opendb('testdb1')
>>> db = ibs.db
>>> tablename = wbia.const.NAME_TABLE
>>> exclude_columns = []
>>> # execute function
>>> csv_table = db.get_table_csv(tablename, exclude_columns)
>>> # verify results
>>> result = str(csv_table)
>>> print(result)
```

utool.util_sqlite.**get_table_num_rows**(*cur*, *tablename*)

utool.util_sqlite.**get_table_rows**(*cur*, *tablename*, *colnames*, *where=None*, *params=None*, *unpack=True*)

utool.util_sqlite.**get_tablenames**(*cur*)

Conveinience:

utool.util_sqlite.**print_database_structure**(*cur*)

1.54 utool.util_str module

Module that handles string formating and manipulation of various data

utool.util_str.**align**(*text*, *character='='*, *replchar=None*, *pos=0*)

Left justifies text on the left side of character

align

Parameters

- **text** (*str*) – text to align
- **character** (*str*) – character to align at
- **replchar** (*str*) – replacement character (default=None)

Returns new_text

Return type str

CommandLine: python -m utool.util_str –test-align:0

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_str import *  # NOQA
>>> character = '='
>>> text = 'a = b=\none = two\nthree = fish\n'
>>> print(text)
>>> result = (align(text, '='))
```

(continues on next page)

(continued from previous page)

```
>>> print(result)
a     = b=
one   = two
three = fish
```

utool.util_str.**align_lines**(*line_list*, *character='='*, *replchar=None*, *pos=0*)

Left justifies text on the left side of character

align_lines

Todo: clean up and move to ubelt?

Parameters

- **line_list** (*list of strs*) –
- **character** (*str*) –
- **pos** (*int or list or None*) – does one alignment for all chars beyond this column position. If pos is None, then all chars are aligned.

Returns new_lines

Return type list

CommandLine: python -m utool.util_str –test-align_lines:0 python -m utool.util_str –test-align_lines:1 python -m utool.util_str –test-align_lines:2 python -m utool.util_str –test-align_lines:3

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_str import *  # NOQA
>>> line_list = 'a = b\none = two\nthree = fish'.split('\n')
>>> character = '='
>>> new_lines = align_lines(line_list, character)
>>> result = ('\n'.join(new_lines))
>>> print(result)
a     = b
one   = two
three = fish
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_str import *  # NOQA
>>> line_list = 'foofish:\n    a = b\n    one    = two\n    three    = fish'.
↪split('\n')
>>> character = '='
>>> new_lines = align_lines(line_list, character)
>>> result = ('\n'.join(new_lines))
>>> print(result)
foofish:
    a        = b
```

(continues on next page)

(continued from previous page)

```
    one      = two
    three    = fish
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_str import *  # NOQA
>>> import utool as ut
>>> character = ':'
>>> text = ut.codeblock('''
    {'max': '1970/01/01 02:30:13',
     'mean': '1970/01/01 01:10:15',
     'min': '1970/01/01 00:01:41',
     'range': '2:28:32',
     'std': '1:13:57',}''').split('\n')
>>> new_lines = align_lines(text, ':', ' :')
>>> result = '\n'.join(new_lines)
>>> print(result)
{'max'   : '1970/01/01 02:30:13',
 'mean'  : '1970/01/01 01:10:15',
 'min'   : '1970/01/01 00:01:41',
 'range' : '2:28:32',
 'std'   : '1:13:57',}
```

Example

```
>>> # ENABLE_DOCEST
>>> from utool.util_str import *  # NOQA
>>> line_list = 'foofish:\n a = b = c\n one = two = three\nthree=4= fish'.split(
↪'\n')
>>> character = '='
>>> # align the second occurence of a character
>>> new_lines = align_lines(line_list, character, pos=None)
>>> print(('\n'.join(line_list)))
>>> result = ('\n'.join(new_lines))
>>> print(result)
foofish:
 a   = b   = c
 one = two = three
three=4    = fish
```

Ignore: # use this as test case begin{tabular}{lrrll} toprule {} & Names & Annots & Annots size & Training Edges \ midrule training & 390 & 1164 & 2.98pm2.83 & 9360 \ testing & 363 & 1119 & 3.08pm2.82 & - \ bottomrule end{tabular}

utool.util_str.**autoformat_pep8**(*sourcecode*, ***kwargs*)

Parameters **code** (*str*) –

CommandLine: python -m utool.util_str –exec-autoformat_pep8

Kwargs: ‘aggressive’: 0, ‘diff’: False, ‘exclude’: [], ‘experimental’: False, ‘files’: [u”], ‘global_config’: ~/.config/pep8, ‘ignore’: set([u’E24’]), ‘ignore_local_config’: False, ‘in_place’: False, ‘indent_size’: 4,

'jobs': 1, 'line_range': None, 'list_fixes': False, 'max_line_length': 79, 'pep8_passes': -1, 'recursive': False, 'select': , 'verbose': 0,

Ignore: autopep8 –recursive –in-place –ignore E126,E127,E201,E202,E203,E221,E222,E241,E265,E271,E272,E301,E501,W60
.

utool.util_str.**autopep8_format**(*sourcecode*, ***kwargs*)

Parameters **code** (*str*) –

CommandLine: python -m utool.util_str –exec-autoformat_pep8

Kwargs: 'aggressive': 0, 'diff': False, 'exclude': [], 'experimental': False, 'files': [u''], 'global_config': ~/.config/pep8, 'ignore': set([u'E24']), 'ignore_local_config': False, 'in_place': False, 'indent_size': 4, 'jobs': 1, 'line_range': None, 'list_fixes': False, 'max_line_length': 79, 'pep8_passes': -1, 'recursive': False, 'select': , 'verbose': 0,

Ignore: autopep8 –recursive –in-place –ignore E126,E127,E201,E202,E203,E221,E222,E241,E265,E271,E272,E301,E501,W60
.

utool.util_str.**bbox_str**(*bbox*, *pad=4*, *sep=', '*)
makes a string from an integer bounding box

utool.util_str.**bubbletext**(*text*, *font='cybermedium'*)
Uses pyfiglet to create bubble text.

Parameters **font** (*str*) – default=cybermedium, other fonts include: cybersmall and cyberlarge.

References

http://www.figlet.org/

Example

```
>>> # ENABLE_DOCTEST
>>> import utool as ut
>>> bubble_text = ut.bubbletext('TESTING BUBBLE TEXT', font='cybermedium')
>>> print(bubble_text)
```

utool.util_str.**byte_str**(*nBytes*, *unit='bytes'*, *precision=2*)
representing the number of bytes with the chosen unit

Returns str

utool.util_str.**byte_str2**(*nBytes*, *precision=2*)
Automatically chooses relevant unit (KB, MB, or GB) for displaying some number of bytes.

Parameters **nBytes** (*int*) –

Returns

Return type str

CommandLine: python -m utool.util_str –exec-byte_str2

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_str import *  # NOQA
>>> import utool as ut
>>> nBytes_list = [1, 100, 1024,  1048576, 1073741824, 1099511627776]
>>> result = ut.list_str(list(map(byte_str2, nBytes_list)), nl=False)
>>> print(result)
['0.00 KB', '0.10 KB', '1.00 KB', '1.00 MB', '1.00 GB', '1.00 TB']
```

utool.util_str.**chr_range**(**args*, ***kw*)
Like range but returns characters

Parameters

- **start** (*None*) – (default = None)
- **stop** (*None*) – (default = None)
- **step** (*None*) – (default = None)

Kwargs: base (str): charater to start with (default='a')

Returns list of characters

Return type list

CommandLine: python -m utool.util_str –exec-chr_range

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_str import *  # NOQA
>>> import utool as ut
>>> args = (5,)
>>> result = ut.repr2(chr_range(2, base='a'))
>>> print(chr_range(0, 5))
>>> print(chr_range(0, 50))
>>> print(chr_range(0, 5, 2))
>>> print(result)
['a', 'b']
```

utool.util_str.**closet_words**(*query*, *options*, *num=1*, *subset=False*)

utool.util_str.**codeblock**(*block_str*)
Convinience function for defining code strings. Esspecially useful for templated code.

utool.util_str.**color_diff_text**(*text*)

utool.util_str.**color_text**(*text*, *color*)

SeeAlso: highlight_text lexer_shortnames = sorted(ut.flatten(ut.take_column(pygments.lexers.LEXERS.values(), 2)))

utool.util_str.**conj_phrase**(*list_*, *cond='or'*)
Joins a list of words using English conjunction rules

Parameters

- **list_** (*list*) – of strings

- **cond** (*str*) – a conjunction (or, and, but)

Returns the joined cconjunction phrase

Return type str

References

http://en.wikipedia.org/wiki/Conjunction_(grammar)

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_str import *  # NOQA
>>> list_ = ['a', 'b', 'c']
>>> result = conj_phrase(list_, 'or')
>>> print(result)
a, b, or c
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_str import *  # NOQA
>>> list_ = ['a', 'b']
>>> result = conj_phrase(list_, 'and')
>>> print(result)
a and b
```

utool.util_str.**countdown_flag**(*count_or_bool*)

utool.util_str.**dict_itemstr_list**(*dict_*, ***dictkw*)

Returns a list of human-readable dictionary items

Return type list

Parameters **explicit** – if True uses dict(key=val,...) format instead of {key:val,...}

utool.util_str.**dict_str**(*dict_*, ***dictkw*)

Makes a pretty printable / human-readable string representation of a dictionary. In most cases this string could be evaled.

Parameters

- **dict_** (*dict*) – a dictionary
- ****dictkw** – stritems, strkeys, strvals, nl, newlines, truncate, nobr, nobraces, align, trailing_sep, explicit, itemsep, truncatekw, sorted_, indent_, key_order, precision, with_comma, key_order_metric, maxlen, recursive, use_numpy, with_dtype, force_dtype, packed
- **sorted_** (*None*) – returns str sorted by a metric (default = None)
- **nl** (*int*) – prefered alias for newline. can be a coundown variable (default = None)
- **key_order** (*None*) – overrides default ordering (default = None)
- **key_order_metric** (*str*) – special sorting of items. Accepted values: None, 'strlen', 'val'

- **precision** (*int*) – (default = 8)
- **explicit** (*int*) – can be a countdown variable. if True, uses dict(a=b) syntax instead of {'a': b}
- **nobr** (*bool*) – removes outer braces (default = False)

Ignore: python -m utool.util_inspect recursive_parse_kwargs:2 –mod utool.util_str –func dict_str –verbinspect

CommandLine: python -m utool.util_str –test-dict_str:1 python -m utool.util_str –test-dict_str –truncate=False –no-checkwant python -m utool.util_str –test-dict_str –truncate=1 –no-checkwant python -m utool.util_str –test-dict_str –truncate=2 –no-checkwant

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_str import dict_str, dict_itemstr_list
>>> import utool as ut
>>> dict_ = {'foo': {'spam': 'barbarbarbarbar' * 3, 'eggs': 'jam'},
>>>          'baz': 'barbarbarbarbar' * 3}
>>> truncate = ut.get_argval('--truncate', type_=None, default=1)
>>> result = dict_str(dict_, strvals=True, truncate=truncate,
>>>                   truncatekw={'maxlen': 20})
>>> print(result)
{
    'baz': barbarbarbarbarbarbarbarbarbarbarbarbarbar,
    'foo': {
        'eggs': jam,
        's ~~~TRUNCATED~~~ r,
    },
}
```

Example

```
>>> # ENABLE_DOCTEST
>>> import utool as ut
>>> import numpy as np
>>> a, b, c = 'a', 'b', 'c'
>>> dict_ = {
>>>     'float': 2.3333,
>>>     'slice': slice(1, 2, None),
>>>     'arr': np.eye(3),
>>>     'list1': [1, 2, 3, 2.3333, a, b, c],
>>>     'dict1': {2.3333: 2.3333, a: b, c: [a, b]},
>>>     't': {c: {c: {c: {c : c}}}},
>>>     'set1': {c, a, b},
>>>     'set2': ut.oset([c, a, b]),
>>>     'list2': [
>>>         {a: {c, a, b}, 1: slice(1, 2, 3)},
>>>         [1, 2, {c, a, 2.333}, {a: [b], b: {c}, c: 2.333}]
>>>     ],
>>> }
>>> dictkw = dict(stritems=True, itemsep='', precision=2, nl=1,
>>>               nobr=True, explicit=True)
>>> result = ut.dict_str(dict_, **dictkw)
```

(continues on next page)

(continued from previous page)

```
>>> print(result)
>>> dictkw = dict(stritems=0, precision=2, nl=True, nobr=False,
>>>                   explicit=0)
>>> result = ut.dict_str(dict_, **dictkw)
>>> print(result)
```

utool.util_str.**difftext**(*text1*, *text2*, *num_context_lines=0*, *ignore_whitespace=False*)

Uses difflib to return a difference string between two similar texts

Parameters

- **text1** (*str*) –
- **text2** (*str*) –

Returns formatted difference text message

Return type str

SeeAlso: ut.color_diff_text

References

http://www.java2s.com/Code/Python/Utility/IntelligentdiffbetweentextfilesTimPeters.htm

CommandLine: python -m utool.util_str –test-get_textdiff:1 python -m utool.util_str –test-get_textdiff:0

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_str import *  # NOQA
>>> # build test data
>>> text1 = 'one\ntwo\nthree'
>>> text2 = 'one\ntwo\nfive'
>>> # execute function
>>> result = get_textdiff(text1, text2)
>>> # verify results
>>> print(result)
- three
+ five
```

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_str import *  # NOQA
>>> # build test data
>>> text1 = 'one\ntwo\nthree\n3.1\n3.14\n3.1415\npi\n3.4\n3.5\n4'
>>> text2 = 'one\ntwo\nfive\n3.1\n3.14\n3.1415\npi\n3.4\n4'
>>> # execute function
>>> num_context_lines = 1
>>> result = get_textdiff(text1, text2, num_context_lines)
>>> # verify results
>>> print(result)
```

utool.util_str.**doctest_code_line**(*line_str*, *varname=None*, *verbose=True*)

utool.util_str.**doctest_repr**(*var*, *varname=None*, *precision=2*, *verbose=True*)

utool.util_str.**ensure_ascii**(*str_*)

utool.util_str.**ensure_unicode_strlist**(*str_list*)

utool.util_str.**file_megabytes_str**(*fpath*)

utool.util_str.**filesize_str**(*fpath*)

utool.util_str.**filtered_infostr**(*flags*, *lbl*, *reason=None*)

utool.util_str.**find_block_end**(*row*, *line_list*, *sentinal*, *direction=1*)
Searches up and down until it finds the endpoints of a block Rectify with find_paragraph_end in pyvim_funcs

utool.util_str.**flatten_textlines**(*text*)

utool.util_str.**format_multi_paragraphs**(*text*, *debug=False*, ***kwargs*)
FIXME: funky things happen when multiple newlines in the middle of paragraphs

CommandLine: python ~/local/vim/rc/pyvim_funcs.py –test-format_multiple_paragraph_sentences

CommandLine: python -m utool.util_str –exec-format_multiple_paragraph_sentences –show

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_str import *  # NOQA
>>> import os, sys
>>> #sys.path.append(os.path.expanduser('~/local/vim/rc'))
>>> text = testdata_text(2)
>>> formated_text = format_multiple_paragraph_sentences(text, debug=True)
>>> print('+--- Text ---')
>>> print(text)
>>> print('+--- Formated Text ---')
>>> print(formated_text)
>>> print('L_____')
```

utool.util_str.**format_multiple_paragraph_sentences**(*text*, *debug=False*, ***kwargs*)
FIXME: funky things happen when multiple newlines in the middle of paragraphs

CommandLine: python ~/local/vim/rc/pyvim_funcs.py –test-format_multiple_paragraph_sentences

CommandLine: python -m utool.util_str –exec-format_multiple_paragraph_sentences –show

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_str import *  # NOQA
>>> import os, sys
>>> #sys.path.append(os.path.expanduser('~/local/vim/rc'))
>>> text = testdata_text(2)
>>> formated_text = format_multiple_paragraph_sentences(text, debug=True)
>>> print('+--- Text ---')
>>> print(text)
>>> print('+--- Formated Text ---')
>>> print(formated_text)
>>> print('L_____')
```

utool.util_str.**format_single_paragraph_sentences**(*text*, *debug=False*, *mypre-fix=True*, *sentence_break=True*, *max_width=73*, *sepcolon=True*)

helps me separatate sentences grouped in paragraphs that I have a difficult time reading due to dyslexia

Parameters **text** (*str*) –

Returns wrapped_text

Return type str

CommandLine: python -m utool.util_str –exec-format_single_paragraph_sentences –show python -m utool.util_str –exec-format_single_paragraph_sentences –show –nobreak

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_str import *  # NOQA
>>> import utool as ut
>>> text = '     lorium ipsum doloar dolar dolar dolar erata man foobar is this␣
↪there yet almost man not quit ate 80 chars yet hold out almost there? dolar␣
↪erat. sau.ltum. fds.fd... . . fd oob fd. list: (1) abcd, (2) foobar (4)␣
↪123456789 123456789 123456789 123456789 123 123 123 123 123456789 123 123 123␣
↪123 123456789 123456789 123456789 123456789 123456789 123 123 123 123 123␣
↪123456789 123456789 123456789 123456789 123456789 123456789 (3) spam.'
>>> #text = 'list: (1) abcd, (2) foobar (3) spam.'
>>> #text = 'foo. when: (1) there is a new individual,'
>>> #text = 'when: (1) there is a new individual,'
>>> #text = '? ? . lorium. ipsum? dolar erat. saultum. fds.fd...  fd oob fd. ? '
↪# causes breakdown
>>> print('text = %r' % (text,))
>>> sentence_break = not ut.get_argflag('--nobreak')
>>> wrapped_text = format_single_paragraph_sentences(text, debug=True, sentence_
↪break=sentence_break)
>>> result = ('wrapped_text =\n%s' % (str(wrapped_text),))
>>> print(result)
```

utool.util_str.**format_text_as_docstr**(*text*)

CommandLine: python ~/local/vim/rc/pyvim_funcs.py –test-format_text_as_docstr

Example

```
>>> # DISABLE_DOCTEST
>>> from pyvim_funcs import *  # NOQA
>>> text = testdata_text()
>>> formated_text = format_text_as_docstr(text)
>>> result = ('formated_text = \n%s' % (str(formated_text),))
>>> print(result)
```

utool.util_str.**func_callsig**(*func*, *with_name=True*)

String of function call signature

Parameters **func** (*function*) – live python function

Returns callsig

Return type str

CommandLine: python -m utool.util_str –exec-func_callsig

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_str import *  # NOQA
>>> func = func_str
>>> callsig = func_callsig(func)
>>> result = str(callsig)
>>> print(result)
func_str(func, args, kwargs, type_aliases, packed, packkw, truncate)
```

utool.util_str.**func_defsig**(*func*, *with_name=True*)

String of function definition signature

Parameters **func** (*function*) – live python function

Returns defsig

Return type str

CommandLine: python -m utool.util_str –exec-func_defsig

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_str import *  # NOQA
>>> func = func_str
>>> defsig = func_defsig(func)
>>> result = str(defsig)
>>> print(result)
func_str(func, args=[], kwargs={}, type_aliases=[], packed=False, packkw=None,␣
↪truncate=False)
```

utool.util_str.**func_str**(*func*, *args=[]*, *kwargs={}*, *type_aliases=[]*, *packed=False*, *packkw=None*, *truncate=False*)

string representation of function definition

Returns a representation of func with args, kwargs, and type_aliases

Return type str

Parameters

- **func** (*function*) –
- **args** (*list*) – argument values (default = [])
- **kwargs** (*dict*) – kwargs values (default = {})
- **type_aliases** (*list*) – (default = [])
- **packed** (*bool*) – (default = False)
- **packkw** (*None*) – (default = None)

Returns func_str

Return type str

CommandLine: python -m utool.util_str –exec-func_str

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_str import *  # NOQA
>>> func = byte_str
>>> args = [1024, 'MB']
>>> kwargs = dict(precision=2)
>>> type_aliases = []
>>> packed = False
>>> packkw = None
>>> _str = func_str(func, args, kwargs, type_aliases, packed, packkw)
>>> result = _str
>>> print(result)
byte_str(1024, 'MB', precision=2)
```

utool.util_str.**get_bytes**(*nUnits*, *unit*)

utool.util_str.**get_callable_name**(*func*)

Works on must functionlike objects including str, which has no func_name

Parameters **func** (*function*) –

Returns

Return type str

CommandLine: python -m utool.util_str –exec-get_callable_name

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_str import *  # NOQA
>>> func = len
>>> result = get_callable_name(func)
>>> print(result)
len
```

utool.util_str.**get_indentation**(*line_*)

returns the number of preceding spaces

utool.util_str.**get_itemstr_list**(*list_*, ***listkw*)

TODO: have this replace dict_itemstr list or at least most functionality in it. have it make two itemstr lists over keys and values and then combine them.

utool.util_str.**get_minimum_indentation**(*text*)

returns the number of preceding spaces

Parameters **text** (*str*) – unicode text

Returns indentation

Return type int

CommandLine: python -m utool.util_str –exec-get_minimum_indentation –show

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_str import *  # NOQA
>>> import utool as ut
>>> text = '    foo\n   bar'
>>> result = get_minimum_indentation(text)
>>> print(result)
3
```

utool.util_str.**get_textdiff**(*text1*, *text2*, *num_context_lines=0*, *ignore_whitespace=False*)

Uses difflib to return a difference string between two similar texts

Parameters

- **text1** (*str*) –
- **text2** (*str*) –

Returns formatted difference text message

Return type str

SeeAlso: ut.color_diff_text

References

http://www.java2s.com/Code/Python/Utility/IntelligentdiffbetweentextfilesTimPeters.htm

CommandLine: python -m utool.util_str –test-get_textdiff:1 python -m utool.util_str –test-get_textdiff:0

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_str import *  # NOQA
>>> # build test data
>>> text1 = 'one\ntwo\nthree'
>>> text2 = 'one\ntwo\nfive'
>>> # execute function
>>> result = get_textdiff(text1, text2)
>>> # verify results
>>> print(result)
- three
+ five
```

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_str import *  # NOQA
>>> # build test data
>>> text1 = 'one\ntwo\nthree\n3.1\n3.14\n3.1415\npi\n3.4\n3.5\n4'
>>> text2 = 'one\ntwo\nfive\n3.1\n3.14\n3.1415\npi\n3.4\n4'
>>> # execute function
>>> num_context_lines = 1
>>> result = get_textdiff(text1, text2, num_context_lines)
```

(continues on next page)

(continued from previous page)

```
>>> # verify results
>>> print(result)
```

utool.util_str.**highlight_code**(*text*, *lexer_name='python'*)

utool.util_str.**highlight_multi_regex**(*str_*, *pat_to_color*, *reflags=0*)
FIXME Use pygments instead. must be mututally exclusive

utool.util_str.**highlight_regex**(*str_*, *pat*, *reflags=0*, *color='red'*)
FIXME Use pygments instead

utool.util_str.**highlight_text**(*text*, *lexer_name='python'*, ***kwargs*)

SeeAlso: color_text

utool.util_str.**horiz_string**(**args*, ***kwargs*)
Horizontally concatenates strings reprs preserving indentation

Concats a list of objects ensuring that the next item in the list is all the way to the right of any previous items.

Parameters

- **`*args`** – list of strings to concat
- **`**kwargs`** – precision, sep

CommandLine: python -m utool.util_str –test-horiz_string

Example

```
>>> # ENABLE_DOCTEST
>>> # Pretty printing of matrices demo / test
>>> import utool
>>> import numpy as np
>>> # Wouldn't it be nice if we could print this operation easily?
>>> B = np.array(((1, 2), (3, 4)))
>>> C = np.array(((5, 6), (7, 8)))
>>> A = B.dot(C)
>>> # Eg 1:
>>> result = (utool.hz_str('A = ', A, ' = ', B, ' * ', C))
>>> print(result)
A = [[19 22]  = [[1 2]  * [[5 6]
     [43 50]]    [3 4]]    [7 8]]
```

Exam2:

```
>>> # Eg 2:
>>> str_list = ['A = ', str(B), ' * ', str(C)]
>>> horizstr = (utool.horiz_string(*str_list))
>>> result = (horizstr)
>>> print(result)
A = [[1 2]  * [[5 6]
     [3 4]]    [7 8]]
```

utool.util_str.**hz_str**(**args*, ***kwargs*)
Horizontally concatenates strings reprs preserving indentation

Concats a list of objects ensuring that the next item in the list is all the way to the right of any previous items.

Parameters

- ***args** – list of strings to concat
- ****kwargs** – precision, sep

CommandLine: python -m utool.util_str –test-horiz_string

Example

```
>>> # ENABLE_DOCTEST
>>> # Pretty printing of matrices demo / test
>>> import utool
>>> import numpy as np
>>> # Wouldn't it be nice if we could print this operation easily?
>>> B = np.array(((1, 2), (3, 4)))
>>> C = np.array(((5, 6), (7, 8)))
>>> A = B.dot(C)
>>> # Eg 1:
>>> result = (utool.hz_str('A = ', A, ' = ', B, ' * ', C))
>>> print(result)
A = [[19 22]  = [[1 2]  * [[5 6]
     [43 50]]    [3 4]]    [7 8]]
```

Exam2:

```
>>> # Eg 2:
>>> str_list = ['A = ', str(B), ' * ', str(C)]
>>> horizstr = (utool.horiz_string(*str_list))
>>> result = (horizstr)
>>> print(result)
A = [[1 2]  * [[5 6]
     [3 4]]    [7 8]]
```

utool.util_str.**indent**(*str_*, *indent=' '*)

Indents a block of text

Parameters

- **str_** (*str*) –
- **indent** (*str*) – (default = ' ') TODO rename to indent or rename func

Returns

Return type str

CommandLine: python -m utool.util_str –test-indent

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_str import *  # NOQA
>>> str_ = 'foobar\nbazbiz'
>>> indent = '    '
```

(continues on next page)

(continued from previous page)

```
>>> result = indent(str_, indent)
>>> print(result)
```

utool.util_str.**indent_list**(*indent*, *list_*)

utool.util_str.**indent_rest**(*str_*, *indent='   '*)
TODO fix name Indents every part of the string except the beginning SeeAlso: wbia/templates/generate_notebook.py

utool.util_str.**indentjoin**(*strlist*, *indent='\n   '*, *suffix=''*)
Convineince indentjoin

similar to '\n '.join(strlist) but indent is also prefixed

Parameters

- **strlist** –
- **indent** (*str*) –
- **suffix** (*str*) –

Returns joined list

Return type str

utool.util_str.**insert_block_between_lines**(*text*, *row1*, *row2*, *line_list*, *inplace=False*)

utool.util_str.**is_byte_encoded_unicode**(*str_*)

utool.util_str.**is_url**(*str_*)
heuristic check if str is url formatted

utool.util_str.**joins**(*string*, *list_*, *with_head=True*, *with_tail=False*, *tostrip='\n'*)

utool.util_str.**list_str**(*list_*, ***listkw*)
Makes a pretty list string

Parameters

- **list_** (*list*) – input list
- ****listkw*** – nl, newlines, packed, truncate, nobr, nobraces, itemsep, trailing_sep, truncatekw, strvals, recursive, indent_, precision, use_numpy, with_dtype, force_dtype, stritems, strkeys, align, explicit, sorted_, key_order, key_order_metric, maxlen

Returns retstr

Return type str

CommandLine: python -m utool.util_str –test-list_str python -m utool.util_str –exec-list_str –truncate=True python -m utool.util_str –exec-list_str –truncate=0

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_str import *  # NOQA
>>> import utool as ut
>>> list_ = [[(('--verbose-qt', '--verbqt'), 1, False, ''),
>>>           (('--verbose-qt', '--verbqt'), 1, False, ''),
>>>           (('--verbose-qt', '--verbqt'), 1, False, ''),
```

(continues on next page)

(continued from previous page)

```
>>>           (('--verbose-qt', '--verbqt'), 1, False, '')],
>>>          [(['--nodyn'], 1, False, ''), (['--nodyn'], 1, False, '')]]
>>> listkw = {'nl':  2}
>>> result = list_str(list_, **listkw)
>>> print(result)
[
    [
        (('--verbose-qt', '--verbqt'), 1, False, ''),
        (('--verbose-qt', '--verbqt'), 1, False, ''),
        (('--verbose-qt', '--verbqt'), 1, False, ''),
        (('--verbose-qt', '--verbqt'), 1, False, ''),
    ],
    [
        (['--nodyn'], 1, False, ''),
        (['--nodyn'], 1, False, ''),
    ],
]
```

utool.util_str.**list_str_summarized**(*list_*, *list_name*, *maxlen=5*)

prints the list members when the list is small and the length when it is large

utool.util_str.**long_fname_format**(*fmt_str*, *fmt_dict*, *hashable_keys=[]*, *max_len=64*, *hashlen=16*, *ABS_MAX_LEN=255*, *hack27=False*)

DEPRICATE

Formats a string and hashes certain parts if the resulting string becomes too long. Used for making filenames fit onto disk.

Parameters

- **fmt_str** (*str*) – format of fname
- **fmt_dict** (*str*) – dict to format fname with
- **hashable_keys** (*list*) – list of dict keys you are willing to have hashed
- **max_len** (*int*) – tries to fit fname into this length
- **ABS_MAX_LEN** (*int*) – throws AssertionError if fname over this length

CommandLine: python -m utool.util_str –exec-long_fname_format

Example

```
>>> # ENABLE_DOCTET
>>> import utool as ut
>>> fmt_str = 'qaid={qaid}_res_{cfgstr}_quuid={quuid}'
>>> quuid_str = 'blahblahblahblahblahblah'
>>> cfgstr = 'big_long_string_____________________________________'
>>> qaid = 5
>>> fmt_dict = dict(cfgstr=cfgstr, qaid=qaid, quuid=quuid_str)
>>> hashable_keys = ['cfgstr', 'quuid']
>>> max_len = 64
>>> hashlen = 8
>>> fname0 = ut.long_fname_format(fmt_str, fmt_dict, max_len=None)
>>> fname1 = ut.long_fname_format(fmt_str, fmt_dict, hashable_keys,
>>>                               max_len=64, hashlen=8)
>>> fname2 = ut.long_fname_format(fmt_str, fmt_dict, hashable_keys, max_len=42,
```

(continues on next page)

(continued from previous page)

```
>>>                           hashlen=8)
>>> result = fname0 + '\n' + fname1 + '\n' + fname2
>>> print(result)
qaid=5_res_big_long_string_____________________________________
 ↪quuid=blahblahblahblahblahblah
qaid=5_res_racfntgq_quuid=blahblahblahblahblahblah
qaid=5_res_racfntgq_quuid=yvuaffrp
```

utool.util_str.**lorium_ipsum**()
Standard testing string

utool.util_str.**msgblock**(*key*, *text*, *side='|'*)
puts text inside a visual ascii block

utool.util_str.**multi_replace**(*str_*, *search_list*, *repl_list*)
Performs multiple replace functions foreach item in search_list and repl_list.

Parameters

- **str_** (*str*) – string to search
- **search_list** (*list*) – list of search strings
- **repl_list** (*list or str*) – one or multiple replace strings

Returns str

CommandLine: python -m utool.util_str –exec-multi_replace

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_str import *  # NOQA
>>> str_ = 'foo. bar: baz; spam-eggs --- eggs+spam'
>>> search_list = ['.', ':', '---']
>>> repl_list = '@'
>>> str_ = multi_replace(str_, search_list, repl_list)
>>> result = ('str_ = %s' % (str(str_),))
>>> print(result)
str_ = foo@ bar@ baz; spam-eggs @ eggs+spam
```

utool.util_str.**number_text_lines**(*text*)

Parameters **text** (*str*) –

Returns text_with_lineno - string with numbered lines

Return type str

utool.util_str.**numpy_str**(*arr*, *strvals=False*, *precision=None*, *pr=None*, *force_dtype=False*, *with_dtype=None*, *suppress_small=None*, *max_line_width=None*, *threshold=None*, ***kwargs*)
suppress_small = False turns off scientific representation

utool.util_str.**order_of_magnitude_str**(*num*, *base=10.0*, *prefix_list=None*, *exponent_list=None*, *suffix=''*, *prefix=None*)
TODO: Rewrite byte_str to use this func :returns: str

utool.util_str.**pack_into**(*text*, *textwidth=160*, *breakchars=' '*, *break_words=True*, *newline_prefix=''*, *wordsep=' '*, *remove_newlines=True*)

DEPRICATE IN FAVOR OF textwrap.wrap

TODO: Look into textwrap.wrap

Inserts newlines into a string enforcing a maximum textwidth. Similar to vim's gq command in visual select mode.

breakchars is a string containing valid characters to insert a newline before or after.

break_words is True if words are allowed to be split over multiple lines.

all inserted newlines are prefixed with newline_prefix

#FIXME:

Example

```
>>> # DISABLE_DOCTEST
>>> text = "set_image_uris(ibs<139684018194000>, [1, 2, 3, 4, 5, 6, 7, 8, 9, 10,␣
↪11, 12, 13], [u'66ec193a-1619-b3b6-216d-1784b4833b61.jpg', u'd8903434-942f-e0f5-
↪d6c2-0dcbe3137bf7.jpg', u'b73b72f4-4acb-c445-e72c-05ce02719d3d.jpg', u'0cd05978-
↪3d83-b2ee-2ac9-798dd571c3b3.jpg', u'0a9bc03d-a75e-8d14-0153-e2949502aba7.jpg', u
↪'2deeff06-5546-c752-15dc-2bd0fdb1198a.jpg', u'a9b70278-a936-c1dd-8a3b-
↪bc1e9a998bf0.png', u'42fdad98-369a-2cbc-67b1-983d6d6a3a60.jpg', u'c459d381-fd74-
↪1d99-6215-e42e3f432ea9.jpg', u'33fd9813-3a2b-774b-3fcc-4360d1ae151b.jpg', u
↪'97e8ea74-873f-2092-b372-f928a7be30fa.jpg', u'588bc218-83a5-d400-21aa-
↪d499832632b0.jpg', u'163a890c-36f2-981e-3529-c552b6d668a3.jpg'], ) "  # NOQA
>>> textwidth = 160
>>> breakchars = ' '
>>> break_words = True
>>> newline_prefix = '    '
>>> wordsep = ' '
>>> packstr1 = pack_into(text, textwidth, breakchars, break_words, newline_prefix,
↪ wordsep)
>>> break_words = False
>>> packstr2 = pack_into(text, textwidth, breakchars, break_words, newline_prefix,
↪ wordsep)
>>> print(packstr1)
>>> print(packstr2)
```

CommandLine: python -c "import utool" –dump-utool-init

utool.util_str.**packstr**(*instr*, *textwidth=160*, *breakchars=' '*, *break_words=True*, *newline_prefix=''*, *indentation=''*, *nlprefix=None*, *wordsep=' '*, *remove_newlines=True*)

alias for pack_into. has more up to date kwargs

utool.util_str.**packtext**(*text*, *width=80*)

Parameters **text** (*str*) –

CommandLine: python -m utool.util_str –exec-pack_paragraph –show

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_str import *  # NOQA
>>> import utool as ut
>>> width = 80
>>> text = lorium_ipsum()
>>> result = packtext(text)
>>> print(result)
```

utool.util_str.**parse_bytes**(*bytes_str*)

Parse bytes from string

Ignore:

```
>>> uint8_size = ut.parse_bytes('1B')
>>> image_size = ut.parse_bytes('3.5MB')
>>> float32_size = ut.parse_bytes('32bit')
>>> desc_size = 128 * uint8_size
>>> kpts_size = 6 * float32_size
>>> chip_size = ut.parse_bytes('400 KB')
>>> probchip_size = ut.parse_bytes('50 KB')
>>> nImgs = 80000 # 80,000
>>> nAnnots = nImgs * 2
>>> desc_per_img = 3000
>>> size_stats = {
>>>     'image': nImgs * image_size,
>>>     'chips': nAnnots * chip_size,
>>>     'probchips': nAnnots * probchip_size,
>>>     'desc': nAnnots * desc_size * desc_per_img,
>>>     'kpts': nAnnots * kpts_size * desc_per_img,
>>> }
>>> print(ut.repr3(ut.map_dict_vals(ut.byte_str2, size_stats), align=True))
>>> print('total = ' + ut.byte_str2(sum(size_stats.values())))
```

utool.util_str.**pluralize**(*wordtext*, *num=2*, *plural_suffix='s'*)

Heuristically changes a word to its plural form if *num* is not 1

Parameters

- **wordtext** (*str*) – word in singular form
- **num** (*int*) – a length of an associated list if applicable (default = 2)
- **plural_suffix** (*str*) – heurstic plural form (default = 's')

Returns pluralized form. Can handle some genitive cases

Return type str

CommandLine: python -m utool.util_str pluralize

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_str import *  # NOQA
>>> wordtext = 'foo'
>>> result = pluralize(wordtext)
>>> print(result)
foos
```

utool.util_str.**quantstr**(*typestr*, *num*, *plural_suffix='s'*)

Heuristically generates an english phrase relating to the quantity of something. This is useful for writing user messages.

Parameters

- **typestr** (*str*) – singular form of the word
- **num** (*int*) – quanity of the type
- **plural_suffix** (*str*) – heurstic plural form (default = 's')

Returns quantity phrase

Return type str

CommandLine: python -m utool.util_str quantity_str

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_str import *  # NOQA
>>> items = [1, 2, 3]
>>> result = 'The list contains ' + (quantstr('item', len(items)))
>>> items = [1]
>>> result += '\nThe list contains ' + (quantstr('item', len(items)))
>>> items = []
>>> result += '\nThe list contains ' + (quantstr('item', len(items)))
>>> print(result)
The list contains 3 items
The list contains 1 item
The list contains 0 items
```

utool.util_str.**regex_reconstruct_split**(*pattern*, *text*, *debug=False*)

utool.util_str.**remove_chars**(*str_*, *char_list*)

removes all chars in char_list from str_

Parameters

- **str_** (*str*) –
- **char_list** (*list*) –

Returns outstr

Return type str

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_str import *  # NOQA
>>> str_ = '1, 2, 3, 4'
>>> char_list = [',']
>>> result = remove_chars(str_, char_list)
>>> print(result)
1 2 3 4
```

utool.util_str.**remove_doublenewlines**(*text*)

utool.util_str.**remove_doublspaces**(*text*)

utool.util_str.**replace_between_tags**(*text*, *repl_*, *start_tag*, *end_tag=None*)

Replaces text between sentinal lines in a block of text.

Parameters

- **text** (*str*) –
- **repl_** (*str*) –
- **start_tag** (*str*) –
- **end_tag** (*str*) – (default=None)

Returns new_text

Return type str

CommandLine: python -m utool.util_str –exec-replace_between_tags

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_str import *  # NOQA
>>> text = ut.codeblock(
    '''
    class:
        # <FOO>
        bar
        # </FOO>
        baz
    ''')
>>> repl_ = 'spam'
>>> start_tag = '# <FOO>'
>>> end_tag = '# </FOO>'
>>> new_text = replace_between_tags(text, repl_, start_tag, end_tag)
>>> result = ('new_text =\n%s' % (str(new_text),))
>>> print(result)
new_text =
class:
    # <FOO>
spam
    # </FOO>
    baz
```

utool.util_str.**repr2**(*obj_*, ***kwargs*)

Attempt to replace repr more configurable pretty version that works the same in both 2 and 3

utool.util_str.**repr2_json**(*obj_*, ***kwargs*)

hack for json reprs

utool.util_str.**repr3**(*obj_*, ***kwargs*)

utool.util_str.**repr4**(*obj_*, ***kwargs*)

utool.util_str.**reprfunc**(*val*, *precision=None*)

Parameters

- **val** –

- **precision** (*None*) – (default = None)

Returns

Return type str

CommandLine: python -m utool.util_str reprfunc –show

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_str import *  # NOQA
>>> import utool as ut
>>> #vals = [{'foo': [1/3, 2]}, np.float64, 1/3, 'foo']
>>> vals = [{'foo': [1, 2]}, np.float64, 1/3, 'foo']
>>> precision = 2
>>> result = ut.repr3([reprfunc(val, precision) for val in vals], nobr=True)
>>> print(result)
"{u'foo': [1, 2]}",
'numpy.float64',
'0.33',
"'foo'",
```

utool.util_str.**scalar_str**(*val*, *precision=None*, *max_precision=None*)

utool.util_str.**second_str**(*nsecs*, *unit=None*, *precision=None*, *abbrev=True*)

utool.util_str.**seconds_str**(*num*, *prefix=None*)

Returns str

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_str import *  # NOQA
>>> import utool as ut
>>> num_list = sorted([4.2 / (10.0 ** exp_)
>>>                    for exp_ in range(-13, 13, 4)])
>>> secstr_list = [seconds_str(num, prefix=None) for num in num_list]
>>> result = (', '.join(secstr_list))
>>> print(result)
0.04 ns, 0.42 us, 4.20 ms, 0.04 ks, 0.42 Ms, 4.20 Gs, 42.00 Ts
```

utool.util_str.**split_sentences2**(*text*, *debug=0*)

utool.util_str.**str_between**(*str_*, *startstr*, *endstr*)
gets substring between two sentianl strings

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_str import *  # NOQA
>>> import utool as ut
>>> str_ = '\n        INSERT INTO vsone(\n'
>>> startstr = 'INSERT'
```

(continues on next page)

(continued from previous page)

```
>>> endstr = '('
>>> result = str_between(str_, startstr, endstr)
>>> print(result)
```

utool.util_str.**strip_ansi**(*text*)

Removes all ansi directives from the string Helper to remove ansi from length calculation References: http://stackoverflow.com/questions/14693701remove-ansi

utool.util_str.**testdata_text**(*num=1*)

utool.util_str.**textblock**(*multiline_text*)

Parameters **block_str** (*str*) –

CommandLine: python -m utool.util_str –test-textblock

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_str import *  # NOQA
>>> # build test data
>>> multiline_text = ''' a big string
>>> that should be layed out flat
>>> yet still provide nice python
>>> code that doesnt go too far over
>>> 80 characters.
>>>
>>> Two newlines should be respected though
>>> '''
>>> # execute function
>>> new_text = textblock(multiline_text)
>>> # verify results
>>> result = new_text
>>> print(result)
```

utool.util_str.**theta_str**(*theta*, *taustr='*2pi'*, *fmtstr='{coeff:*, *.1f}{taustr}'*)

Format theta so it is interpretable in base 10

Parameters

- **theta** (*float*) –
- **taustr** (*str*) – default 2pi

Returns theta_str - the angle in tau units

Return type str

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_str import *  # NOQA
>>> theta = 3.1415
>>> result = theta_str(theta)
>>> print(result)
0.5*2pi
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_str import *  # NOQA
>>> theta = 6.9932
>>> taustr = 'tau'
>>> result = theta_str(theta, taustr)
>>> print(result)
1.1tau
```

utool.util_str.**to_camel_case**(*underscore_case*, *mixed=False*)

Parameters **underscore_case** –

Returns camel_case_str

Return type str

CommandLine: python -m utool.util_str –exec-to_camel_case

References

https://en.wikipedia.org/wiki/CamelCase

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_str import *  # NOQA
>>> underscore_case = 'underscore_funcname'
>>> camel_case_str = to_camel_case(underscore_case)
>>> result = ('camel_case_str = %s' % (str(camel_case_str),))
>>> print(result)
camel_case_str = UnderscoreFuncname
```

utool.util_str.**to_title_caps**(*underscore_case*)

Parameters **underscore_case** –

Returns title_str

Return type str

CommandLine: python -m utool.util_str –exec-to_title_caps

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_str import *  # NOQA
>>> underscore_case = 'the_foo_bar_func'
>>> title_str = to_title_caps(underscore_case)
>>> result = ('title_str = %s' % (str(title_str),))
>>> print(result)
title_str = The Foo Bar Func
```

utool.util_str.**to_underscore_case**(*camelcase_str*)

References

http://stackoverflow.com/questions/1175208/convert-camelcase

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_str import *  # NOQA
>>> camelcase_str = 'UnderscoreFuncname'
>>> camel_case_str = to_underscore_case(camelcase_str)
>>> result = ('underscore_str = %s' % (str(camel_case_str),))
>>> print(result)
underscore_str = underscore_funcname
```

utool.util_str.**toggle_comment_lines**(*text*, *pattern*, *flag*, *char='#'*)

utool.util_str.**trunc_repr**(*obj*, *maxlen=50*)

utool.util_str.**truncate_str**(*str_*, *maxlen=110*, *truncmsg=' ~~~TRUNCATED~~~ '*)
Removes the middle part of any string over maxlen characters.

utool.util_str.**unformat_text_as_docstr**(*formated_text*)

CommandLine: python ~/local/vim/rc/pyvim_funcs.py –test-unformat_text_as_docstr

Example

```
>>> # DISABLE_DOCTEST
>>> from pyvim_funcs import *  # NOQA
>>> text = testdata_text()
>>> formated_text = format_text_as_docstr(text)
>>> unformated_text = unformat_text_as_docstr(formated_text)
>>> result = ('unformated_text = \n%s' % (str(unformated_text),))
>>> print(result)
```

utool.util_str.**unindent**(*string*)
Unindent a block of text

Alias for textwrap.dedent

utool.util_str.**utf8_len**(*str_*)
returns num printed characters in utf8

Returns //stackoverflow.com/questions/2247205/python-returning-the-wrong-length-of-string-when-using-special-characters

Return type http

utool.util_str.**varinfo_str**(*varval*, *varname*, *onlyrepr=False*, *canshowrepr=True*, *varcolor='yellow'*, *colored=True*)

utool.util_str.**verts_str**(*verts*, *pad=1*)
makes a string from a list of integer verticies

1.55 utool.util_sysreq module

utool.util_sysreq.**ensure_in_pythonpath**(*dname*)

utool.util_sysreq.**get_global_dist_packages_dir**()
Attempts to work around virtualenvs and find the system dist_pacakges. Essentially this is implmenented as a lookuptable

utool.util_sysreq.**get_local_dist_packages_dir**()
Attempts to work around virtualenvs and find the system dist_pacakges. Essentially this is implmenented as a lookuptable

utool.util_sysreq.**get_site_packages_dir**()

CommandLine: python -m utool.util_sysreq get_site_packages_dir

Notes

It seems IPython does not respect virtual environments properly. TODO: find a solution http://stackoverflow.com/questions/7335992/ipython-and-virtualenv-ignoring-site-packages

utool.util_sysreq.**in_virtual_env**()
returns True if you are running inside a python virtual environment. (DOES NOT WORK IF IN IPYTHON AND USING A VIRTUALENV)

sys.prefix gives the location of the virtualenv

Notes

It seems IPython does not respect virtual environments properly. TODO: find a solution http://stackoverflow.com/questions/7335992/ipython-and-virtualenv-ignoring-site-packages

References

http://stackoverflow.com/questions/1871549/python-determine-if-running-inside-virtualenv

CommandLine: python -m utool.util_sysreq in_virtual_env

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_sysreq import *  # NOQA
>>> import utool as ut
>>> result = in_virtual_env()
>>> print(result)
```

utool.util_sysreq.**is_running_as_root**()

References

http://stackoverflow.com/questions/5721529/running-python-script-as-root-with-sudo-what-is-the-username-of-the-effectiv
http://stackoverflow.com/questions/2806897/what-is-the-best-practices-for-checking-if-the-user-of-a-python-script-has-root

utool.util_sysreq.**locate_path**(*dname*, *recurse_down=True*)
Search for a path

utool.util_sysreq.**total_purge_developed_repo**(*repodir*)
Outputs commands to help purge a repo

Parameters **repodir** (*str*) – path to developed repository

CommandLine: python -m utool.util_sysreq total_purge_installed_repo –show

Ignore: repodir = ut.truepath('~/code/Lasagne')

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_sysreq import *  # NOQA
>>> import utool as ut
>>> repodir = ut.get_argval('--repodir', default=None)
>>> result = total_purge_installed_repo(repodir)
```

1.56 utool.util_tags module

utool.util_tags.**alias_tags**(*tags_list*, *alias_map*)

update tags to new values

Parameters

- **tags_list** (*list*) –
- **alias_map** (*list*) – list of 2-tuples with regex, value

Returns updated tags

Return type list

CommandLine: python -m utool.util_tags alias_tags –show

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_tags import *  # NOQA
>>> import utool as ut
>>> tags_list = [['t1', 't2'], [], ['t3'], ['t4', 't5']]
>>> ut.build_alias_map()
>>> result = alias_tags(tags_list, alias_map)
>>> print(result)
```

utool.util_tags.**build_alias_map**(*regex_map*, *tag_vocab*)

Constructs explicit mapping. Order of items in regex map matters. Items at top are given preference.

Example

```
>>> # DISABLE_DOCTEST
>>> tags_list = [['t1', 't2'], [], ['t3'], ['t4', 't5']]
>>> tag_vocab = ut.flat_unique(*tags_list)
>>> regex_map = [('t[3-4]', 'A9'), ('t0', 'a0')]
>>> unmapped = list(set(tag_vocab) - set(alias_map.keys()))
```

utool.util_tags.**filterflags_general_tags**(*tags_list*, *has_any=None*, *has_all=None*, *has_none=None*, *min_num=None*, *max_num=None*, *any_startswith=None*, *any_endswith=None*, *in_any=None*, *any_match=None*, *none_match=None*, *logic='and'*, *ignore_case=True*)

maybe integrate into utool? Seems pretty general

Parameters

- **tags_list** (*list*) –
- **has_any** (*None*) – (default = None)
- **has_all** (*None*) – (default = None)
- **min_num** (*None*) – (default = None)
- **max_num** (*None*) – (default = None)

Notes

in_any should probably be ni_any

TODO: make this function more natural

CommandLine: python -m utool.util_tags –exec-filterflags_general_tags python -m utool.util_tags –exec-filterflags_general_tags:0 –helpx python -m utool.util_tags –exec-filterflags_general_tags:0 python -m utool.util_tags –exec-filterflags_general_tags:0 –none_match n python -m utool.util_tags –exec-filterflags_general_tags:0 –has_none=n,o python -m utool.util_tags –exec-filterflags_general_tags:1 python -m utool.util_tags –exec-filterflags_general_tags:2

Ignore:

```
>>> # ENABLE_DOCTEST
>>> from utool.util_tags import *  # NOQA
>>> import utool as ut
>>> tags_list = [['v'], [], ['P'], ['P', 'o'], ['n', 'o'], [], ['n', 'N'], ['e
↪', 'i', 'p', 'b', 'n'], ['q', 'v'], ['n'], ['n'], ['N']]
>>> kwargs = ut.argparse_dict(ut.get_kwdefaults2(filterflags_general_tags),␣
↪type_hint=list)
>>> print('kwargs = %r' % (kwargs,))
>>> flags = filterflags_general_tags(tags_list, **kwargs)
>>> print(flags)
>>> result = ut.compress(tags_list, flags)
>>> print('result = %r' % (result,))
```

Ignore:

```
>>> # ENABLE_DOCTEST
>>> from utool.util_tags import *  # NOQA
>>> import utool as ut
>>> tags_list = [['v'], [], ['P'], ['P'], ['n', 'o'], [], ['n', 'N'], ['e', 'i
↪', 'p', 'b', 'n'], ['n'], ['n'], ['N']]
>>> has_all = 'n'
>>> min_num = 1
>>> flags = filterflags_general_tags(tags_list, has_all=has_all, min_num=min_
↪num)
>>> result = ut.compress(tags_list, flags)
>>> print('result = %r' % (result,))
```

Ignore:

```
>>> # ENABLE_DOCTEST
>>> from utool.util_tags import *  # NOQA
>>> import utool as ut
>>> tags_list = [['vn'], ['vn', 'no'], ['P'], ['P'], ['n', 'o'], [], ['n', 'N
↪'], ['e', 'i', 'p', 'b', 'n'], ['n'], ['n', 'nP'], ['NP']]
>>> kwargs = {
>>>     'any_endswith': 'n',
>>>     'any_match': None,
>>>     'any_startswith': 'n',
>>>     'has_all': None,
>>>     'has_any': None,
>>>     'has_none': None,
>>>     'max_num': 3,
>>>     'min_num': 1,
>>>     'none_match': ['P'],
>>> }
>>> flags = filterflags_general_tags(tags_list, **kwargs)
>>> filtered = ut.compress(tags_list, flags)
>>> result = ('result = %s' % (ut.repr2(filtered),))
result = [['vn', 'no'], ['n', 'o'], ['n', 'N'], ['n'], ['n', 'nP']]
```

utool.util_tags.**modify_tags**(*tags_list*, *direct_map=None*, *regex_map=None*, *regex_aug=None*, *delete_unmapped=False*, *return_unmapped=False*, *return_map=False*)

utool.util_tags.**tag_coocurrence**(*tags_list*)

utool.util_tags.**tag_hist**(*tags_list*)

1.57 utool.util_tests module

Helpers for tests

This module contains a more sane reimplementation of doctest functionality. (I.E. asserts work and you don't have to worry about stdout mucking things up) The code isn't super clean though due to time constriaints. Many functions probably belong elsewhere and the parsers need a big cleanup.

Notes

DEPRICATE THIS IN FAVOR OF pytest and xdoctest

Todo: report the line of the doctest in the file when reporting errors as well as the relative line restructure so there is a test collection step, a filtering step, and an execution step Fix finding tests when running with @profile

---

class utool.util_tests.**ModuleDoctestTup**(*enabled_testtup_list*, *frame_fpath*, *all_testflags*, *module*)

Bases: tuple

all_testflags
Alias for field number 2

enabled_testtup_list
Alias for field number 0

frame_fpath
Alias for field number 1

module
Alias for field number 3

class utool.util_tests.**TestTuple**(*name*, *num*, *src*, *want*, *flag*, *tags=None*, *frame_fpath=None*, *mode=None*, *nametup=None*, *test_namespace=None*, *shortname=None*, *total=None*)
Bases: *utool.util_dev.NiceRepr*

Simple container for test objects to replace old tuple format exec mode specifies if the test is being run as a script

exec_mode

full_name

modname

name

namespace

namespace_levels

utool.util_tests.**doctest_funcs**(*testable_list=None*, *check_flags=True*, *module=None*, *allexamples=None*, *needs_enable=None*, *strict=False*, *verbose=True*, *return_error_report=True*, *seen_=None*)
Main entry point into utools main module doctest harness Imports a module and checks flags for the function to run Depth 1)

Parameters

- **testable_list** (*list*) –
- **check_flags** (*bool*) – Force checking of the –test- and –exec- flags
- **module** (*None*) –
- **allexamples** (*None*) –
- **needs_enable** (*None*) –

Returns (nPass, nTotal, failed_cmd_list)

Return type tuple

CommandLine: python -m wbia.algo.preproc.preproc_chip –all-examples

References

http://legacy.python.org/dev/peps/pep-0338/ https://docs.python.org/2/library/runpy.html

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_tests import *  # NOQA
>>> testable_list = []
>>> check_flags = True
>>> module = None
```

(continues on next page)

(continued from previous page)

```
>>> allexamples = None
>>> needs_enable = None
>>> # careful might infinitely recurse
>>> (nPass, nTotal) = doctest_funcs(testable_list, check_flags, module,
...                                 allexamples, needs_enable)
>>> print((nPass, nTotal))
```

utool.util_tests.**doctest_module_list**(*module_list*)

Runs many module tests

Entry point for batch run Depth 0)

Ignore: :'<,'>!sort -n -k 2

utool.util_tests.**doctest_was_requested**()

lets a __main__ codeblock know that util_test should do its thing

utool.util_tests.**execute_doctest**(*func*, *testnum=0*, *module=None*)

Execute a function doctest. Can optionaly specify func name and module to run from ipython notebooks.

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_tests import *  # NOQA
```

IPython: import utool as ut ut.execute_doctest(func='dummy_example_depcacahe', module='dtool.example_depcache')

utool.util_tests.**find_doctestable_modnames**(*dpath_list=None*, *exclude_doctests_fnames=[]*, *exclude_dirs=[]*, *allow_nonpackages=False*)

Tries to find files with a call to ut.doctest_funcs in the __main__ part Implementation is very hacky. Should find a better heuristic

Parameters

- **dpath_list** (*list*) – list of python package directories
- **exclude_doctests_fnames** (*list*) – (default = [])
- **exclude_dirs** (*list*) – (default = [])
- **allow_nonpackages** (*bool*) – (default = False)

Returns list of filepaths

Return type list

CommandLine: python -m utool.util_tests find_doctestable_modnames –show

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_tests import *  # NOQA
>>> import utool as ut
>>> from os.path import dirname
```

(continues on next page)

(continued from previous page)

```
>>> dpath_list = [ut.get_module_dir(ut)]
>>> exclude_doctests_fnames = []
>>> exclude_dirs = []
>>> allow_nonpackages = False
>>> result = find_doctestable_modnames(dpath_list, exclude_doctests_fnames,␣
↪exclude_dirs, allow_nonpackages)
>>> print(result)
```

utool.util_tests.**find_testfunc**(*module*, *test_funcname*, *ignore_prefix=[]*, *ignore_suffix=[]*, *func_to_module_dict={}*, *return_mod=False*)

utool.util_tests.**find_untested_modpaths**(*dpath_list=None*, *exclude_doctests_fnames=[]*, *exclude_dirs=[]*)

utool.util_tests.**get_doctest_examples**(*func_or_class*, *modpath=None*)

Depth 3) called by get_module_doctest_tup

Parameters **func_or_class** (*function*) –

Returns example_list, want_list

Return type tuple (list, list)

CommandLine: python -m utool.util_tests –test-get_doctest_examples

Example

```
>>> from utool.util_tests import *  # NOQA
>>> func_or_class = get_doctest_examples
>>> tup  = get_doctest_examples(func_or_class)
>>> testsrc_list, testwant_list, testlinenum_list, func_lineno, docstr = tup
>>> result = str(len(testsrc_list) + len(testwant_list))
>>> print(testsrc_list)
>>> print(testlinenum_list)
>>> print(func_lineno)
>>> print(testwant_list)
>>> print(result)
6
```

Example

```
>>> from utool.util_tests import *  # NOQA
>>> import utool as ut
>>> func_or_class = ut.tryimport
>>> tup = get_doctest_examples(func_or_class)
>>> testsrc_list, testwant_list, testlinenum_list, func_lineno, docstr = tup
>>> result = str(len(testsrc_list) + len(testwant_list))
>>> print(testsrc_list)
>>> print(testlinenum_list)
>>> print(func_lineno)
>>> print(testwant_list)
>>> print(result)
4
```

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_tests import *  # NOQA
>>> import wbia
>>> func_or_class = wbia.control.manual_annot_funcs.add_annots
>>> tup = get_doctest_examples(func_or_class)
>>> testsrc_list, testwant_list, testlinenum_list, func_lineno, docstr = tup
>>> result = str(len(testsrc_list) + len(testwant_list))
>>> print(testsrc_list)
>>> print(testlinenum_list)
>>> print(func_lineno)
>>> print(testwant_list)
>>> print(result)
2
```

utool.util_tests.**get_module_completions**(*module*)

utool.util_tests.**get_module_doctest_tup**(*testable_list=None*, *check_flags=True*, *module=None*, *allexamples=None*, *needs_enable=None*, *N=0*, *verbose=True*, *testslow=False*)

Parses module for testable doctesttups

enabled_testtup_list (list): a list of testtup

testtup (tuple): (name, num, src, want, flag) describes a valid doctest in the module name (str): test name num (str): test number of the module / function / class / method src (str): test source code want (str): expected test result flag (str): a valid commandline flag to enable this test

frame_fpath (str): module fpath that will be tested all_testflags (list): the command line arguments that will enable different tests module (module): the actual module that will be tested exclude_inherited (bool): does not included tests defined in other modules

Parameters

- **testable_list** (*list*) – a list of functions (default = None)
- **check_flags** (*bool*) – (default = True)
- **module** (*None*) – (default = None)
- **allexamples** (*None*) – (default = None)
- **needs_enable** (*None*) – (default = None)
- **N** (*int*) – (default = 0)
- **verbose** (*bool*) – verbosity flag(default = True)
- **testslow** (*bool*) – (default = False)

Returns (enabled_testtup_list, frame_fpath, all_testflags, module)

Return type *ModuleDoctestTup*

CommandLine: python -m utool.util_tests –exec-get_module_doctest_tup

Example

```
>>> from utool.util_tests import *  # NOQA
>>> import utool as ut
>>> #testable_list = [ut.util_import.package_contents]
>>> testable_list = None
>>> check_flags = False
>>> module = ut.util_cplat
>>> allexamples = False
>>> needs_enable = None
>>> N = 0
>>> verbose = True
>>> testslow = False
>>> mod_doctest_tup = get_module_doctest_tup(testable_list, check_flags,
>>>                                          module, allexamples,
>>>                                          needs_enable, N, verbose,
>>>                                          testslow)
>>> result = ('mod_doctest_tup = %s' % (ut.repr4(mod_doctest_tup, nl=4),))
>>> print(result)
```

utool.util_tests.**get_module_testlines**(*module_list*, *remove_pyc=True*, *verbose=True*, *pythoncmd=None*)

Builds test commands for autogen tests called by autogen test scripts

utool.util_tests.**get_package_testables**(*module=None*, ***tagkw*)

New command that should eventually be used intead of old stuff?

Parameters

- **module_list** (*list*) – (default = None)
- **test_flags** (*None*) – (default = None)

CommandLine: python -m utool get_package_testables –show –mod wbia python -m utool get_package_testables –show –mod utool –tags SCRIPT python -m utool get_package_testables –show –mod utool –tags ENABLE

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_tests import *  # NOQA
>>> import utool as ut
>>> has_any = ut.get_argval('--tags', type_=str, default=None)
>>> module = ut.get_argval('--mod', default='utool')
>>> test_tuples = get_package_testables(module, has_any=has_any)
>>> result = ut.repr3(test_tuples)
>>> print(result)
>>> #print(ut.repr3(ut.list_getattr(test_tuples, 'tags')))
```

utool.util_tests.**main_function_tester**(*module*, *ignore_prefix=[]*, *ignore_suffix=[]*, *test_funcname=None*, *func_to_module_dict={}*)

Allows a shorthand for __main__ packages of modules to run tests with unique function names

utool.util_tests.**parse_docblocks_from_docstr**(*docstr*, *offsets=False*)

Depth 5) called by parse_doctest_from_docstr

TODO: move to util_inspect

Parameters **docstr** (*str*) – a documentation string formatted in google style.

Returns

docstr_blocks tuples [(blockname, blockstr)]

Return type list

CommandLine: python -m utool.util_tests parse_docblocks_from_docstr

Example

```
>>> from utool.util_tests import *  # NOQA
>>> import utool as ut
>>> #func_or_class = ut.flatten2
>>> #func_or_class = ut.list_depth
>>> func_or_class = ut.iter_module_doctestable
>>> func_or_class = ut.all_multi_paths
>>> docstr = ut.get_docstr(func_or_class)
>>> docstr_blocks = parse_docblocks_from_docstr(docstr)
>>> result = str(ut.repr4(docstr_blocks))
>>> print(result)
```

utool.util_tests.**parse_doctest_from_docstr**(*docstr*)

because doctest itself doesnt do what I want it to do

CAREFUL, IF YOU GET BACK WRONG RESULTS MAKE SURE YOUR DOCSTR IS PREFFIXED WITH R

CommandLine: python -m utool.util_tests –exec-parse_doctest_from_docstr

Example

```
>>> from utool.util_tests import *  # NOQA
>>> import utool as ut
>>> func_or_class = parse_doctest_from_docstr
>>> func_or_class = ut.list_depth
>>> #func_or_class = ut.util_depricated.cartesian
>>> func_or_class = ut.iter_module_doctestable
>>> docstr = ut.get_docstr(func_or_class)
>>> testsrc_list, testwant_list, testlinenum_list, func_lineno, docstr = get_
↪doctest_examples(func_or_class)
>>> print('\n\n'.join(testsrc_list))
>>> assert len(testsrc_list) == len(testwant_list)
```

utool.util_tests.**qt4ensure**()

utool.util_tests.**qtensure**()

utool.util_tests.**quit_if_noshow**()

utool.util_tests.**read_exampleblock**(*docblock*)

utool.util_tests.**run_test**(*func_or_testtup*, **args*, ***kwargs*)

Runs the test function. Prints a success / failure report.

Parameters

- **func_or_testtup** (*func or tuple*) – function or doctest tuple
- **args*** – args to be forwarded to *func_or_testtup*

- **kwargs*** – keyword args to be forwarded to *func_or_testtup*

utool.util_tests.**show_if_requested**()

utool.util_tests.**show_was_requested**()
returns True if –show is specified on the commandline or you are in IPython (and presumably want some sort of interaction

1.58 utool.util_time module

TODO: This file seems to care about timezone

TODO: Use UTC/GMT time here for EVERYTHING

References

http://www.timeanddate.com/time/aboututc.html

class utool.util_time.**Timer**(*msg=''*, *verbose=True*, *newline=True*)
Bases: object

Timer with-statment context object.

Example

```
>>> # ENABLE_DOCTEST
>>> import utool as ut
>>> timer = ut.Timer('Timer test!', verbose=1)
>>> with timer:
>>>     prime = ut.get_nth_prime(400)
>>> assert timer.ellapsed > 0
```

start()

stop()

tic()

toc()

class utool.util_time.**Timerit**(*num*, *label=None*, *unit=None*, *verbose=1*)
Bases: object

Reports the average time to run a block of code.

Unlike *timeit*, *Timerit* can handle multiline blocks of code

Parameters

- **num** (*int*) – number of times to run the loop
- **label** (*str*) – identifier for printing
- **unit** (*str*) – reporting unit of time (e.g. 'ms', 's', None)

CommandLine: python -m utool.util_time Timerit python -m utool.util_time Timerit:0 python -m utool.util_time Timerit:1

Notes

```
>>> # Minimal syntax with less-precise timing
>>> import utool as ut
>>> for timer in ut.Timerit(100):
>>>     # <write code to time here>
>>>
>>> # Full syntax with most-precise timing
>>> import utool as ut
>>> for timer in ut.Timerit(100):
>>>     # <write untimed setup code here>
>>>     with timer:
>>>         # <write code to time here>
>>>
>>> # Can also keep track of the Timerit object for extra statistics
>>> import utool as ut
>>> t1 = ut.Timerit(100)
>>> for timer in t1:
>>>     # <write untimed setup code here>
>>>     with timer:
>>>         # <write code to time here>
>>> # <you can now access Timerit attributes like t1.total_time>
```

Example

```
>>> # ENABLE_DOCTEST
>>> import utool as ut
>>> num = 15
>>> t1 = ut.Timerit(num)
>>> for timer in t1:
>>>     # <write untimed setup code here> this example has no setup
>>>     with timer:
>>>         # <write code to time here> for example...
>>>         ut.get_nth_prime_bruteforce(100)
>>> # <you can now access Timerit attributes>
>>> print('t1.total_time = %r' % (t1.total_time,))
>>> assert t1.total_time > 0
>>> assert t1.n_loops == t1.num
>>> assert t1.n_loops == num
```

Example

```
>>> # DISABLE_DOCTEST
>>> import utool as ut
>>> num = 10000
>>> n = 50
>>> # If the timer object is unused, time will still be recoreded,
>>> # but with less precision.
>>> for _ in ut.Timerit(num, 'inprecise'):
>>>     ut.get_nth_prime_bruteforce(n)
>>> # Using the timer object results in the most precise timeings
>>> for timer in ut.Timerit(num, 'precise'):
>>>     with timer:
>>>         ut.get_nth_prime_bruteforce(n)
```

ave_secs

utool.util_time.**date_to_datetime**(*date*, *fraction=0.0*)
fraction is how much through the day you are. 0=start of the day, 1=end of the day.

utool.util_time.**datetime_to_posixtime**(*dt*)

utool.util_time.**determine_timestamp_format**(*datetime_str*, *warn=True*)

Parameters **datetime_str** (*str*) –

Returns

Return type str

References

https://docs.python.org/2/library/datetime.html#strftime-and-strptime-behavior

CommandLine: python -m utool.util_time –exec-determine_timestamp_format

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_time import *  # NOQA
>>> import utool as ut
>>> datetime_str_list = [
>>>     '0000:00:00 00:00:00',
>>>     '    :  :     :  :  ',
>>>     '2015:04:01 00:00:00',
>>>     '2080/04/01 00:00:00',
>>>     '2005-10-27T14:35:20+02:00',
>>>     '6:35:01\x002006:03:19 1',
>>>     '2016/05/03 16:34:57 EST'
>>> ]
>>> result = ut.repr4([determine_timestamp_format(datetime_str)
>>>            for datetime_str in datetime_str_list])
>>> print(result)
```

utool.util_time.**ensure_timedelta**(*str_or_num*)

utool.util_time.**exiftime_to_unixtime**(*datetime_str*, *timestamp_format=None*, *strict=None*)
converts a datetime string to posixtime (unixtime) Use parse timestamp instead

Parameters

- **datetime_str** (*str*) –
- **timestamp_format** (*int*) –

Returns unixtime seconds from 1970 (currently not UTC; this will change)

Return type int

CommandLine: python -m utool.util_time –test-exiftime_to_unixtime:2

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_time import *  # NOQA
>>> datetime_str = '0000:00:00 00:00:00'
>>> timestamp_format = 1
>>> result = exiftime_to_unixtime(datetime_str, timestamp_format)
>>> print(result)
-1
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_time import *  # NOQA
>>> datetime_str = '2015:04:01 00:00:00'
>>> timestamp_format = 1
>>> result = exiftime_to_unixtime(datetime_str, timestamp_format)
>>> print(result)
1427846400
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_time import *  # NOQA
>>> datetime_str = '2005-10-27T14:35:20+02:00'
>>> timestamp_format = None
>>> result = exiftime_to_unixtime(datetime_str, timestamp_format)
>>> print(result)
1130423720
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_time import *  # NOQA
>>> datetime_str = '6:35:01\x002006:03:19 1'
>>> timestamp_format = None
>>> result = exiftime_to_unixtime(datetime_str, timestamp_format)
>>> print(result)
1142750101
```

utool.util_time.**get_datestamp**(*explicit=True*, *isutc=False*)

utool.util_time.**get_posix_timedelta_str**(*posixtime*, *year=False*, *approx=True*)

get_timedelta_str

TODO: rectify this function with get_unix_timedelta_str (unix_timedelta_str probably has better implementation)

Returns timedelta_str, formated time string

Return type str

CommandLine: python -m utool.util_time –test-get_posix_timedelta_str

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_time import *  # NOQA
>>> import utool as ut
>>> posixtime_list = [-13, 10.2, 10.2 ** 2, 10.2 ** 3, 10.2 ** 4, 10.2 ** 5, 10.2␣
↪** 8, 60 * 60 * 60 * 24 * 7]
>>> posixtime = posixtime_list[-1]
>>> timedelta_str = [get_posix_timedelta_str(posixtime) for posixtime in␣
↪posixtime_list]
>>> result = ut.repr2(timedelta_str)
>>> print(result)
['-00:00:13', '00:00:10', '00:01:44', '00:17:41', '03:00:24', '1 day', '193 weeks
↪', '60 weeks']
```

Timeit: import datetime # Seems like like timedelta is just faster. must be because it is builtin %timeit get_posix_timedelta_str(posixtime) %timeit str(datetime.timedelta(seconds=posixtime))

utool.util_time.**get_posix_timedelta_str2**(*posixtime*)

utool.util_time.**get_timedelta_str**(*timedelta*, *exclude_zeros=False*)

Returns timedelta_str, formated time string

Return type str

References

http://stackoverflow.com/questions/8906926/formatting-python-timedelta-objects

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_time import *  # NOQA
>>> timedelta = get_unix_timedelta(10)
>>> timedelta_str = get_timedelta_str(timedelta)
>>> result = (timedelta_str)
>>> print(result)
10 seconds
```

utool.util_time.**get_timestamp**(*format_='iso'*, *use_second=False*, *delta_seconds=None*, *isutc=False*, *timezone=False*)

Parameters

- **format** (*str*) – (tag, printable, filename, other)
- **use_second** (*bool*) –
- **delta_seconds** (*None*) –

Returns stamp

Return type str

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_time import *  # NOQA
>>> format_ = 'printable'
>>> use_second = False
>>> delta_seconds = None
>>> stamp = get_timestamp(format_, use_second, delta_seconds)
>>> print(stamp)
>>> assert len(stamp) == len('15:43:04 2015/02/24')
```

utool.util_time.**get_timestats_dict**(*unixtime_list*, *full=True*, *isutc=True*)

utool.util_time.**get_timestats_str**(*unixtime_list*, *newlines=1*, *full=True*, *isutc=True*)

Parameters

- **unixtime_list** (*list*) –
- **newlines** (*bool*) –

Returns timestat_str

Return type str

CommandLine: python -m utool.util_time –test-get_timestats_str

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_time import *  # NOQA
>>> import utool as ut
>>> unixtime_list = [0, 0 + 60 * 60 * 5 , 10 + 60 * 60 * 5, 100 + 60 * 60 * 5,␣
↪1000 + 60 * 60 * 5]
>>> newlines = 1
>>> full = False
>>> timestat_str = get_timestats_str(unixtime_list, newlines, full=full,␣
↪isutc=True)
>>> result = ut.align(str(timestat_str), ':')
>>> print(result)
{
    'max'  : '1970/01/01 05:16:40',
    'mean' : '1970/01/01 04:03:42',
    'min'  : '1970/01/01 00:00:00',
    'range': '5:16:40',
    'std'  : '2:02:01',
}
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_time import *  # NOQA
>>> import utool as ut
>>> unixtime_list = [0, 0 + 60 * 60 * 5 , 10 + 60 * 60 * 5, 100 + 60 * 60 * 5,␣
↪1000 + 60 * 60 * 5, float('nan'), 0]
>>> newlines = 1
```

(continues on next page)

(continued from previous page)

```
>>> timestat_str = get_timestats_str(unixtime_list, newlines, isutc=True)
>>> result = ut.align(str(timestat_str), ':')
>>> print(result)
{
    'max'    : '1970/01/01 05:16:40',
    'mean'   : '1970/01/01 03:23:05',
    'min'    : '1970/01/01 00:00:00',
    'nMax'   : 1,
    'nMin'   : 2,
    'num_nan': 1,
    'range'  : '5:16:40',
    'shape'  : (7,),
    'std'    : '2:23:43',
}
```

utool.util_time.**get_unix_timedelta**(*unixtime_diff*)

utool.util_time.**get_unix_timedelta_str**(*unixtime_diff*)

TODO: rectify this function with get_posix_timedelta_str

Parameters

- **unixtime_diff** (*int*) – number of seconds
- **unixtime_diff** –

Returns formated time string

Return type timestr (str)

Returns timestr

Return type str

CommandLine: python -m utool.util_time –test-get_unix_timedelta_str

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_time import *  # NOQA
>>> import utool as ut
>>> unixtime_diff = 0
>>> timestr = get_unix_timedelta_str(unixtime_diff)
>>> timestr_list = [get_unix_timedelta_str(_) for _ in [-9001, -1, 0, 1, 9001]]
>>> result = ut.repr2(timestr_list)
>>> print(result)
['2 hours 30 minutes 1 second', '1 second', '0 seconds', '1 second', '2 hours 30␣
↪minutes 1 second']
```

utool.util_time.**local_timezone**()

utool.util_time.**parse_timedelta_str**(*str_*)

Parameters **str** (*str*) –

Returns timedelta

Return type float

CommandLine: python -m utool.util_time –exec-parse_timedelta_str

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_time import *  # NOQA
>>> str_ = '24h'
>>> timedelta = parse_timedelta_str(str_)
>>> result = ('timedelta = %s' % (str(timedelta),))
>>> print(result)
timedelta = 86400.0
```

utool.util_time.**parse_timestamp**(*timestamp*, *zone='UTC'*, *timestamp_format=None*)
pip install delorean

Parameters

- **timestamp** (*str*) – timestampe string
- **zone** (*bool*) – assumes input is zone (only if not specified) and gives output in zone.

CommandLine: python -m utool.util_time –test-parse_timestamp python -m utool.util_time parse_timestamp

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_time import *  # NOQA
>>> import utool as ut
>>> utc = True
>>> timestampe_format = None
>>> timestamps = [
>>>     ('2015:04:01 00:00:00',),
>>>     ('2005-10-27T14:35:20+02:00',),
>>>     ('2000-01-01T09:00:00-05:00', 'UTC'),
>>>     ('2000-01-01T09:00:00-05:00', 'EST'),
>>>     ('2000-01-01T09:00:00', 'EST'),
>>>     ('2000-01-01T09:00:00', 'UTC'),
>>>     ('6:35:01\x002006:03:19 1',),
>>>     ('2016/08/18 10:51:02 EST',),
>>>     ('2016-08-18T10:51:02-05:00',),
>>> ]
>>> timestamp = timestamps[-1][0]
>>> dn_list = [parse_timestamp(*args) for args in timestamps]
>>> result = ut.NEWLINE.join([str(dn) for dn in dn_list])
>>> print(result)
2015-04-01 00:00:00+00:00
2005-10-27 12:35:20+00:00
2000-01-01 14:00:00+00:00
2000-01-01 09:00:00-05:00
2000-01-01 09:00:00-05:00
2000-01-01 09:00:00+00:00
2006-03-19 06:35:01+00:00
2016-08-18 15:51:02+00:00
2016-08-18 15:51:02+00:00
```

utool.util_time.**tic**(*msg=None*)
similar to matlab tic

SeeAlso: ut.toc

utool.util_time.**timestamp**(*format_='iso'*, *use_second=False*, *delta_seconds=None*, *isutc=False*, *timezone=False*)

get_timestamp

Parameters

- **format** (*str*) – (tag, printable, filename, other)
- **use_second** (*bool*) –
- **delta_seconds** (*None*) –

Returns stamp

Return type str

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_time import *  # NOQA
>>> format_ = 'printable'
>>> use_second = False
>>> delta_seconds = None
>>> stamp = get_timestamp(format_, use_second, delta_seconds)
>>> print(stamp)
>>> assert len(stamp) == len('15:43:04 2015/02/24')
```

utool.util_time.**toc**(*tt*, *return_msg=False*, *write_msg=True*, *verbose=None*)

similar to matlab toc

SeeAlso: ut.tic

utool.util_time.**unixtime_to_datetimeobj**(*unixtime*, *isutc=True*)

utool.util_time.**unixtime_to_datetimestr**(*unixtime*, *timefmt='%Y/%m/%d %H:%M:%S'*, *isutc=True*)

TODO: ranme to datetimestr

utool.util_time.**unixtime_to_timedelta**(*unixtime_diff*)

alias for get_unix_timedelta

utool.util_time.**utcnow_tz**()

1.59 utool.util_type module

utool.util_type.**assert_int**(*var*, *lbl='var'*)

utool.util_type.**bool_from_str**(*str_*)

utool.util_type.**fuzzy_int**(*str_*)

lets some special strings be interpreted as ints

utool.util_type.**fuzzy_subset**(*str_*)

converts a string into an argument to list_take

utool.util_type.**get_homogenous_list_type**(*list_*)

Returns the best matching python type even if it is an ndarray assumes all items in the list are of the same type. does not check this

utool.util_type.**get_type**(*var*)
Gets types accounting for numpy

Ignore: import utool as ut import pandas as pd var = np.array(['a', 'b', 'c']) ut.get_type(var) var = pd.Index(['a', 'b', 'c']) ut.get_type(var)

utool.util_type.**is_bool**(*var*)

utool.util_type.**is_comparable_type**(*var*, *type_*)
Check to see if *var* is an instance of known compatible types for *type_*

Parameters

- **var** –
- **type** –

Returns

Return type bool

CommandLine: python -m utool.util_type is_comparable_type –show

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_type import *  # NOQA
>>> import utool as ut
>>> flags = []
>>> flags += [is_comparable_type(0, float)]
>>> flags += [is_comparable_type(0, np.float32)]
>>> flags += [is_comparable_type(0, np.int32)]
>>> flags += [is_comparable_type(0, int)]
>>> flags += [is_comparable_type(0.0, int)]
>>> result = ut.repr2(flags)
>>> print(result)
[True, True, True, True, False]
```

utool.util_type.**is_dict**(*var*)

utool.util_type.**is_float**(*var*)

Parameters **var** (*ndarray or scalar*) –

Returns

Return type var

CommandLine: python -m utool.util_type –test-is_float

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_type import *  # NOQA
>>> # build test data
>>> var = np.array([1.0, 2.0, 3.0])
>>> # execute function
>>> assert is_float(var) is True, 'var is a float'
```

(continues on next page)

(continued from previous page)

```
>>> # verify results
>>> print(result)
```

utool.util_type.**is_func_or_method**(*var*)

utool.util_type.**is_func_or_method_or_partial**(*var*)

utool.util_type.**is_funclike**(*var*)

utool.util_type.**is_int**(*var*)

Returns True if var is an integer.

Return type bool

Note: Yuck, isinstance(True, int) returns True. This function does not have that flaw.

References

http://www.peterbe.com/plog/bool-is-int

CommandLine: python -m utool.util_type –test-is_int

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_type import *  # NOQA
>>> var1 = 1
>>> var2 = np.array([1, 2, 3])
>>> var3 = True
>>> var4 = np.array([True, True, False])
>>> result = [is_int(var) for var in [var1, var2, var3, var4]]
>>> print(result)
[True, True, False, False]
```

utool.util_type.**is_list**(*var*)

utool.util_type.**is_listlike**(*var*)

utool.util_type.**is_method**(*var*)

utool.util_type.**is_str**(*var*)

utool.util_type.**is_tuple**(*var*)

utool.util_type.**is_type**(*var*, *valid_types*)
Checks for types accounting for numpy

utool.util_type.**is_valid_floattype**(*type_*)

Parameters **type** (type) – type to check

Returns if a type_ is a valid float type_ (not variable)

Return type bool

utool.util_type.**smart_cast**(*var*, *type_*)
casts var to type, and tries to be clever when var is a string

Parameters

- **var** (*object*) – variable to cast
- **type** (*type or str*) – type to attempt to cast to

Returns

Return type object

CommandLine: python -m utool.util_type –exec-smart_cast

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_type import *  # NOQA
>>> var = '1'
>>> type_ = 'fuzzy_subset'
>>> cast_var = smart_cast(var, type_)
>>> result = repr(cast_var)
>>> print(result)
[1]
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_type import *  # NOQA
>>> import utool as ut
>>> cast_var = smart_cast('1', None)
>>> result = ut.repr2(cast_var)
>>> print(result)
'1'
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_type import *  # NOQA
>>> cast_var = smart_cast('(1,3)', 'eval')
>>> result = repr(cast_var)
>>> print(result)
(1, 3)
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_type import *  # NOQA
>>> cast_var = smart_cast('(1,3)', eval)
>>> result = repr(cast_var)
>>> print(result)
(1, 3)
```

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_type import *  # NOQA
>>> cast_var = smart_cast('1::3', slice)
>>> result = repr(cast_var)
>>> print(result)
slice(1, None, 3)
```

utool.util_type.**smart_cast2**(*var*)

if the variable is a string tries to cast it to a reasonable value. maybe can just use eval. FIXME: funcname

Parameters **var** (*unknown*) –

Returns some var

Return type unknown

CommandLine: python -m utool.util_type –test-smart_cast2

Example

```
>>> # ENABLE_DOCTEST
>>> from utool.util_type import *  # NOQA
>>> import utool as ut
>>> # build test data
>>> var_list = ['?', 1, '1', '1.0', '1.2', 'True', None, 'None']
>>> # execute function
>>> castvar_list = [smart_cast2(var) for var in var_list]
>>> # verify results
>>> result = ut.repr4(castvar_list, nl=False)
>>> print(result)
['?', 1, 1, 1.0, 1.2, True, None, None]
```

utool.util_type.**try_cast**(*var*, *type_*, *default=None*)

utool.util_type.**type_str**(*type_*)

1.60 utool.util_ubuntu module

class utool.util_ubuntu.**XCtrl**

Bases: object

xdotool key ctrl+shift+i

References

http://superuser.com/questions/382616/detecting-currently-active-window http://askubuntu.com/questions/455762/xbindkeys-wont-work-properly

Ignore: xdotool keyup –window 0 7 type –clearmodifiers —window 0 '%paste'

List current windows: wmctrl -l

Get current window xdotool getwindowfocus getwindowname

#==== # Get last opened window #====

win_title=x-terminal-emulator.X-terminal-emulator key_ = 'x-terminal-emulator.X-terminal-emulator'

Get all windows in current workspace workspace_number='wmctrl -d | grep '*' | cut -d' ' -f 1' win_list='wmctrl -lx | grep $win_title | grep " $workspace_number " | awk '{print $1}''

Get stacking order of windows in current workspace win_order=$(xprop -rootlgrep "^_NET_CLIENT_LIST_STACKING" | tr "," " ") echo $win_order

CommandLine: python -m utool.util_ubuntu XCtrl

Example

```
>>> # DISABLE_DOCTEST
>>> # Script
>>> import utool as ut
>>> from utool import util_ubuntu
>>> orig_window = []
>>> ut.copy_text_to_clipboard(ut.lorium_ipsum())
>>> doscript = [
>>>     ('focus', 'x-terminal-emulator.X-terminal-emulator'),
>>>     ('type', '%paste'),
>>>     ('key', 'KP_Enter'),
>>>    # ('focus', 'GVIM')
>>> ]
>>> util_ubuntu.XCtrl.do(*doscript, sleeptime=.01)
```

Ignore:

```
>>> ut.copy_text_to_clipboard(text)
>>> if '\n' in text or len(text) > 20:
>>>     text = '\'%paste\''
>>> else:
>>>     import pipes
>>>     text = pipes.quote(text.lstrip(' '))
>>>     ('focus', 'GVIM'),
>>> #
>>> doscript = [
>>>     ('focus', 'x-terminal-emulator.X-terminal-emulator'),
>>>     ('type', text),
>>>     ('key', 'KP_Enter'),
>>> ]
>>> ut.util_ubuntu.XCtrl.do(*doscript, sleeptime=.01)
```

static copy_gvim_to_terminal_script(*text*, *return_to_win='1'*, *verbose=0*, *sleeptime=0.02*)

import utool.util_ubuntu utool.util_ubuntu.XCtrl.copy_gvim_to_terminal_script('print("hi")', verbose=1) python -m utool.util_ubuntu XCtrl.copy_gvim_to_terminal_script "echo hi" 1 1

If this doesn't work make sure pyperclip is installed and set to xsel

print('foobar') echo hi

static current_gvim_edit(*op='e'*, *fpath=''*)

CommandLine: python -m utool.util_ubuntu XCtrl.current_gvim_edit sp ~/.bashrc

static do(**cmd_list*, ***kwargs*)

static find_window_id(*pattern*, *method='mru'*, *error='raise'*)
xprop -id 0x00a00007 | grep "WM_CLASS(STRING)"

static findall_window_ids(*pattern*)

CommandLine: wmctrl -l python -m utool.util_ubuntu XCtrl.findall_window_ids gvim –src python -m utool.util_ubuntu XCtrl.findall_window_ids gvim –src python -m utool.util_ubuntu XCtrl.findall_window_ids joncrall –src

xprop -id

wmctrl -l | awk '{print $1}' | xprop -id

0x00a00007 | grep "WM_CLASS(STRING)"

static focus_window(*winhandle*, *path=None*, *name=None*, *sleeptime=0.01*)
sudo apt-get install xautomation apt-get install autokey-gtk

wmctrl -xa gnome-terminal.Gnome-terminal wmctrl -xl

static killold(*pattern*, *num=4*)
Leaves no more than *num* instances of a program alive. Ordering is determined by most recent usage.

CommandLine: python -m utool.util_ubuntu XCtrl.killold gvim 2

```
>>> import utool as ut
>>> from utool import util_ubuntu
>>> XCtrl = util_ubuntu.XCtrl
>>> pattern = 'gvim'
>>> num = 2
```

static move_window(*win_key*, *bbox*)

CommandLine: # List windows wmctrl -l # List desktops wmctrl -d

Window info xwininfo -id 60817412

python -m utool.util_ubuntu XCtrl.move_window joncrall 0+1920,680,400,600,400 python -m utool.util_ubuntu XCtrl.move_window joncrall [0,0,1000,1000] python -m utool.util_ubuntu XCtrl.move_window GVIM special2 python -m utool.util_ubuntu XCtrl.move_window joncrall special2 python -m utool.util_ubuntu XCtrl.move_window x-terminal-emulator.X-terminal-emulator [0,0,1000,1000]

>>> import utool as ut # >>> from utool import util_ubuntu # >>> orig_window = [] # >>> X = util_ubuntu.XCtrl win_key = 'x-terminal-emulator.X-terminal-emulator' win_id = X.findall_window_ids(key)[0]

python -m utool.util_ubuntu XCtrl.findall_window_ids gvim –src

static sort_window_ids(*winid_list*, *order='mru'*)
Orders window ids by most recently used

static sorted_window_ids(*order='mru'*)
Returns window ids orderd by criteria default is mru (most recently used)

CommandLine: xprop -root | grep "^_NET_CLIENT_LIST_STACKING" | tr "," " " | python -m utool.util_ubuntu XCtrl.sorted_window_ids

utool.util_ubuntu.**add_new_mimetype_association**(*ext*, *mime_name*, *exe_fpath=None*, *dry=True*)
TODO: move to external manager and generalize

Parameters

- **ext** (*str*) – extension to associate

- **mime_name** (*str*) – the name of the mime_name to create (defaults to ext)
- **exe_fpath** (*str*) – executable location if this is for one specific file

References

https://wiki.archlinux.org/index.php/Default_applications#Custom_file_associations

Parameters

- **ext** (*str*) – extension to associate
- **exe_fpath** (*str*) – executable location
- **mime_name** (*str*) – the name of the mime_name to create (defaults to ext)

CommandLine: python -m utool.util_ubuntu –exec-add_new_mimetype_association # Add ability to open ipython notebooks via double click python -m utool.util_ubuntu –exec-add_new_mimetype_association –mime-name=ipynb+json –ext=.ipynb –exe-fpath=/usr/local/bin/ipynb python -m utool.util_ubuntu –exec-add_new_mimetype_association –mime-name=ipynb+json –ext=.ipynb –exe-fpath=jupyter-notebook –force

python -m utool.util_ubuntu –exec-add_new_mimetype_association –mime-name=sqlite –ext=.sqlite –exe-fpath=sqlitebrowser

Example

```
>>> # DISABLE_DOCTEST
>>> # SCRIPT
>>> from utool.util_ubuntu import *  # NOQA
>>> import utool as ut
>>> ext = ut.get_argval('--ext', type_=str, default=None)
>>> mime_name = ut.get_argval('--mime_name', type_=str, default=None)
>>> exe_fpath = ut.get_argval('--exe_fpath', type_=str, default=None)
>>> dry = not ut.get_argflag('--force')
>>> result = add_new_mimetype_association(ext, mime_name, exe_fpath, dry)
>>> print(result)
```

utool.util_ubuntu.**make_application_icon**(*exe_fpath*, *dry=True*, *props={}*)

CommandLine: python -m utool.util_ubuntu –exec-make_application_icon –exe=cockatrice –icon=/home/joncrall/code/Cockatrice/cockatrice/resources/cockatrice.png python -m utool.util_ubuntu –exec-make_application_icon –exe=cockatrice –icon=/home/joncrall/code/Cockatrice/cockatrice/resources/cockatrice.png python -m utool.util_ubuntu –exec-make_application_icon –exe=/opt/zotero/zotero –icon=/opt/zotero/chrome/icons/default/main-window.ico

python -m utool.util_ubuntu –exec-make_application_icon –exe “env WINEPREFIX=”/home/joncrall/.wine” wine C:\windows\command\start.exe /Unix /home/joncrall/.wine32-dotnet45/dosdevices/c:/users/Public/Desktop/Hearthstone.lnk” –path “/home/joncrall/.wine/dosdevices/c:/Program Files (x86)/Hearthstone” # Exec=env WINEPREFIX=”/home/joncrall/.wine” wine /home/joncrall/.wine/drive_c/ProgramFiles(x86)/Battle.net/Battle.net.exe

–icon=/opt/zotero/chrome/icons/default/main-window.ico

python -m utool.util_ubuntu –exec-make_application_icon –exe=/home/joncrall/code/build-ArenaTracker-Desktop_Qt_5_6_1_GCC_64bit-Debug

update-desktop-database ~/.local/share/applications

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_ubuntu import *  # NOQA
>>> import utool as ut
>>> exe_fpath = ut.get_argval('--exe', default='cockatrice')
>>> icon = ut.get_argval('--icon', default=None)
>>> dry = not ut.get_argflag(('--write', '-w'))
>>> props = {'terminal': False, 'icon': icon}
>>> result = make_application_icon(exe_fpath, dry, props)
>>> print(result)
```

utool.util_ubuntu.**monitor_mouse**()

CommandLine: python -m utool.util_ubuntu monitor_mouse

Example

```
>>> # DISABLE_DOCTEST
>>> # SCRIPT
>>> from utool.util_ubuntu import *  # NOQA
>>> import utool as ut
>>> monitor_mouse()
```

utool.util_ubuntu.**xctrl**
alias of *utool.util_ubuntu.XCtrl*

1.61 utool.util_web module

utool.util_web.**find_open_port**(*base=5000*)

utool.util_web.**get_localhost**()

utool.util_web.**is_local_port_open**(*port*)

Parameters **port** (*int*) –

Returns

Return type bool

References

http://stackoverflow.com/questions/7436801/identifying-listening-ports-using-python

CommandLine: python -m utool.util_web is_local_port_open –show

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_web import *  # NOQA
>>> port = 32183
>>> assert is_local_port_open(80) is False, 'port 80 should always be closed'
>>> assert is_local_port_open(port) is True, 'maybe this port is actually used?'
```

utool.util_web.**render_html**(*html_str*)
makes a temporary html rendering

utool.util_web.**start_simple_webserver**(*domain=None*, *port=5832*)
simple webserver that echos its arguments

Parameters

- **domain** (*None*) – (default = None)
- **port** (*int*) – (default = 5832)

CommandLine: python -m utool.util_web –exec-start_simple_webserver:0 python -m utool.util_web –exec-start_simple_webserver:1

Example

```
>>> # DISABLE_DOCTEST
>>> from utool.util_web import *  # NOQA
>>> domain = None
>>> port = 5832
>>> result = start_simple_webserver(domain, port)
>>> print(result)
```

1.62 utool.util_win32 module

utool.util_win32.**add_to_win32_PATH**(*script_fpath*, **add_path_list*)
Writes a registery script to update the PATH variable into the sync registry

CommandLine: python -m utool.util_win32 –test-add_to_win32_PATH –newpath “C:Program Files (x86)Graphviz2.38bin”

Example

```
>>> # DISABLE_DOCTEST
>>> # SCRIPT
>>> from utool.util_win32 import *  # NOQA
>>> script_fpath = join(ut.truepath('~'), 'Sync/win7/registry', 'UPDATE_PATH.reg')
>>> new_path = ut.get_argval('--newpath', str, default=None)
>>> result = add_to_win32_PATH(script_fpath, new_path)
>>> print(result)
```

utool.util_win32.**get_regstr**(*regtype*, *var*, *val*)

utool.util_win32.**make_regfile_str**(*key*, *varval_list*, *rtype*)

1.63 Module contents

UTool - Useful Utility Tools Your friendly neighborhood utility tools

TODO: INSERT APACHE VERSION 2.0 LICENCE IN ALL FILES (Although it should be implied that the entire module and repo is released under that licence.)

pip install git+https://github.com/Erotemic/utool.git@next

utool.**reassign_submodule_attributes**(*verbose=1*)
Updates attributes in the __init__ modules with updated attributes in the submodules.

utool.**reload_subs**(*verbose=1*)
Reloads utool and submodules

utool.**rrrr**(*verbose=1*)
Reloads utool and submodules

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

u

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z