UBelt Documentation

Release 0.1.1

Jon Crall

Apr 21, 2018

API

1 ubelt **3**

1.1 ubelt package . . . 3

1.1.1 Subpackages . . . 3

1.1.1.1 ubelt.meta package . . . 3

1.1.1.1.1 Submodules . . . 3

1.1.1.1.1.1 ubelt.meta.docscrape_google module . . . 3

1.1.1.1.2 Example . . . 3

1.1.1.1.3 Example . . . 4

1.1.1.1.4 Example . . . 4

1.1.1.1.5 Example . . . 4

1.1.1.1.6 References . . . 5

1.1.1.1.7 Example . . . 5

1.1.1.1.8 Example . . . 6

1.1.1.1.8.1 ubelt.meta.dynamic_analysis module . . . 6

1.1.1.1.9 Example . . . 6

1.1.1.1.10 Example . . . 7

1.1.1.1.11 Module contents . . . 7

1.1.2 Submodules . . . 7

1.1.2.1 ubelt.orderedset module . . . 7

1.1.2.1.1 References . . . 7

1.1.2.1.2 Example . . . 8

1.1.2.1.3 Example . . . 8

1.1.2.1.4 Notes . . . 8

1.1.2.1.5 Example . . . 8

1.1.2.1.6 Example . . . 8

1.1.2.1.7 Example . . . 9

1.1.2.1.8 Example . . . 9

1.1.2.1.9 Example . . . 9

1.1.2.1.10 Example . . . 9

1.1.2.1.11 Example . . . 10

1.1.2.1.12 Example . . . 10

1.1.2.1.13 Example . . . 10

1.1.2.1.14 Example . . . 10

1.1.2.1.15 Example . . . 10

1.1.2.1.16 Example . . . 11

1.1.2.1.17 Example . . . 11

1.1.2.1.18 Example . . . 11
1.1.2.1.19 Example . . . 11
1.1.2.1.20 Example . . . 12
1.1.2.2 ubelt.progiter module . . . 12
1.1.2.2.1 Example . . . 12
1.1.2.2.2 Notes . . . 13
1.1.2.2.3 Example . . . 13
1.1.2.2.4 Example . . . 13
1.1.2.2.5 Example . . . 14
1.1.2.2.6 Example . . . 14
1.1.2.2.7 Example . . . 14
1.1.2.2.8 Example . . . 15
1.1.2.3 ubelt.util_arg module . . . 15
1.1.2.4 ubelt.util_cache module . . . 16
1.1.2.4.1 Example . . . 17
1.1.2.4.2 Example . . . 17
1.1.2.4.3 Example . . . 18
1.1.2.4.4 Example . . . 18
1.1.2.4.5 Example . . . 18
1.1.2.4.6 Example . . . 19
1.1.2.4.7 Example . . . 19
1.1.2.4.8 Example . . . 20
1.1.2.5 ubelt.util_cmd module . . . 20
1.1.2.5.1 Notes . . . 20
1.1.2.5.2 References . . . 20
1.1.2.5.3 Example . . . 21
1.1.2.6 ubelt.util_colors module . . . 21
1.1.2.6.1 Example . . . 22
1.1.2.6.2 Example . . . 22
1.1.2.7 ubelt.util_const module . . . 22
1.1.2.8 ubelt.util_dict module . . . 23
1.1.2.8.1 References . . . 23
1.1.2.8.2 Example . . . 23
1.1.2.8.3 Example . . . 23
1.1.2.8.4 Example . . . 24
1.1.2.8.5 Example . . . 24
1.1.2.8.6 Example . . . 25
1.1.2.8.7 Example . . . 25
1.1.2.8.8 Example . . . 26
1.1.2.8.9 Example . . . 26
1.1.2.8.10 Example . . . 27
1.1.2.8.11 Example . . . 27
1.1.2.8.12 Example . . . 27
1.1.2.8.13 Example . . . 27
1.1.2.8.14 Example . . . 28
1.1.2.8.15 Example . . . 28
1.1.2.8.16 Notes . . . 29
1.1.2.8.17 Example . . . 29
1.1.2.8.18 Example . . . 29
1.1.2.8.19 Example . . . 29
1.1.2.9 ubelt.util_download module . . . 30
1.1.2.9.1 Notes . . . 30
1.1.2.9.2 References . . . 30
1.1.2.9.3 Example . . . 30

1.1.2.9.4 Example . . . 31
1.1.2.10 ubelt.util_format module . . . 31
1.1.2.10.1 Example . . . 31
1.1.2.10.2 Example . . . 32
1.1.2.11 ubelt.util_func module . . . 32
1.1.2.11.1 Example . . . 32
1.1.2.11.2 Example . . . 33
1.1.2.12 ubelt.util_hash module . . . 33
1.1.2.12.1 Example . . . 33
1.1.2.12.2 Notes . . . 34
1.1.2.12.3 References . . . 34
1.1.2.12.4 Example . . . 34
1.1.2.13 ubelt.util_import module . . . 34
1.1.2.13.1 Example . . . 34
1.1.2.13.2 Example . . . 35
1.1.2.13.3 Example . . . 36
1.1.2.13.4 Example . . . 36
1.1.2.13.5 Example . . . 36
1.1.2.13.6 References . . . 37
1.1.2.13.7 Notes . . . 37
1.1.2.13.8 Example . . . 37
1.1.2.14 ubelt.util_io module . . . 37
1.1.2.14.1 Example . . . 38
1.1.2.14.2 Example . . . 38
1.1.2.14.3 References . . . 39
1.1.2.14.4 Example . . . 39
1.1.2.15 ubelt.util_links module . . . 40
1.1.2.15.1 Example . . . 40
1.1.2.15.2 Example . . . 40
1.1.2.16 ubelt.util_list module . . . 41
1.1.2.16.1 References . . . 41
1.1.2.16.2 Example . . . 42
1.1.2.16.3 Example . . . 43
1.1.2.16.4 Example . . . 43
1.1.2.16.5 Example . . . 43
1.1.2.16.6 Example . . . 44
1.1.2.16.7 Example . . . 44
1.1.2.16.8 Example . . . 44
1.1.2.16.9 Example . . . 44
1.1.2.16.10 Example . . . 45
1.1.2.16.11 Example . . . 45
1.1.2.16.12 Example . . . 46
1.1.2.16.13 Example . . . 46
1.1.2.16.14 Example . . . 46
1.1.2.16.15 Example . . . 46
1.1.2.16.16 Example . . . 47
1.1.2.16.17 Example . . . 47
1.1.2.16.18 Example . . . 48
1.1.2.16.19 Example . . . 48
1.1.2.17 ubelt.util_memoize module . . . 48
1.1.2.17.1 References . . . 48
1.1.2.17.2 Example . . . 49
1.1.2.17.3 References . . . 49
1.1.2.17.4 Example . . . 49

1.1.2.18 ubelt.util_mixins module . . . 50
1.1.2.18.1 Example . . . 50
1.1.2.19 ubelt.util_path module . . . 50
1.1.2.19.1 Example . . . 51
1.1.2.19.2 Example . . . 51
1.1.2.19.3 Example . . . 51
1.1.2.19.4 Example . . . 52
1.1.2.19.5 Example . . . 52
1.1.2.19.6 Notes . . . 53
1.1.2.19.7 Example . . . 53
1.1.2.19.8 Example . . . 53
1.1.2.19.9 Example . . . 53
1.1.2.20 ubelt.util_platform module . . . 54
1.1.2.20.1 Example . . . 54
1.1.2.20.2 Example . . . 55
1.1.2.20.3 References . . . 55
1.1.2.21 ubelt.util_str module . . . 55
1.1.2.21.1 Notes . . . 56
1.1.2.21.2 Example . . . 56
1.1.2.21.3 Example . . . 56
1.1.2.21.4 Example . . . 57
1.1.2.21.5 References . . . 58
1.1.2.21.6 Example . . . 58
1.1.2.22 ubelt.util_time module . . . 58
1.1.2.22.1 Example . . . 58
1.1.2.22.2 Example . . . 58
1.1.2.22.3 Example . . . 59
1.1.2.22.4 Example . . . 59
1.1.2.22.5 Example . . . 60
1.1.2.22.6 Example . . . 60
1.1.2.22.7 Example . . . 60
1.1.2.22.8 Example . . . 61
1.1.2.22.9 Example . . . 61
1.1.3 Module contents . . . 61

2 Indices and tables **63**

Python Module Index **65**

UBelt is a “utility belt” of commonly needed utility and helper functions.

CHAPTER 1

ubelt

1.1 ubelt package

1.1.1 Subpackages

1.1.1.1 ubelt.meta package

1.1.1.1.1 Submodules

1.1.1.1.1.1 ubelt.meta.docscrape_google module

Handles parsing of information out of google style docstrings

CommaneLine: # Run the doctests python -m ubelt.meta.docscrape_google all

ubelt.meta.docscrape_google.**parse_google_args**(*docstr*)
Generates dictionaries of argument hints based on a google docstring

Parameters **docstr** (*str*) – a google-style docstring

Yields *dict* – dictionaries of parameter hints

1.1.1.1.2 Example

```
>>> from ubelt.meta.docscrape_google import *  # NOQA
>>> docstr = parse_google_args.__doc__
>>> argdict_list = list(parse_google_args(docstr))
>>> print([sorted(d.items()) for d in argdict_list])
[[('desc', 'a google-style docstring'), ('name', 'docstr'), ('type', 'str')]]
```

ubelt.meta.docscrape_google.**parse_google_returns**(*docstr*, *return_annot=None*)
Generates dictionaries of possible return hints based on a google docstring

Parameters

- **docstr** (*str*) – a google-style docstring
- **return_annot** (*str*) – the return type annotation (if one exists)

Yields *dict* – dictionaries of return value hints

1.1.1.1.3 Example

```
>>> from ubelt.meta.docscrape_google import *  # NOQA
>>> docstr = parse_google_returns.__doc__
>>> retdict_list = list(parse_google_returns(docstr))
>>> print([sorted(d.items()) for d in retdict_list])
[[('desc', 'dictionaries of return value hints'), ('type', 'dict')]]
```

1.1.1.1.4 Example

```
>>> from ubelt.meta.docscrape_google import *  # NOQA
>>> docstr = split_google_docblocks.__doc__
>>> retdict_list = list(parse_google_returns(docstr))
>>> print([sorted(d.items())[1] for d in retdict_list])
[('type', 'list')]
```

ubelt.meta.docscrape_google.**parse_google_retblock**(*lines*, *return_annot=None*)

Parameters

- **lines** (*str*) – unindented lines from a Returns or Yields section
- **return_annot** (*str*) – the return type annotation (if one exists)

Yeilds: dict: each dict specifies the return type and its description

CommandLine: python -m ubelt.meta.docscrape_google parse_google_retblock

1.1.1.1.5 Example

```
>>> from ubelt.meta.docscrape_google import *  # NOQA
>>> # Test various ways that retlines can be written
>>> assert len(list(parse_google_retblock('list: a desc'))) == 1
>>> assert len(list(parse_google_retblock('no type, just desc'))) == 0
>>> # ---
>>> hints = list(parse_google_retblock('\n'.join([
...     'entire line can be desc',
...     ' ',
...     ' if a return type annotation is given',
... ]), return_annot='int'))
>>> assert len(hints) == 1
>>> # ---
>>> hints = list(parse_google_retblock('\n'.join([
...     'bool: a description',
...     ' with a newline',
... ])))
>>> assert len(hints) == 1
```

```
>>> # ---
>>> hints = list(parse_google_retblock('\n'.join([
...     'int or bool: a description',
...     ' ',
...     ' with a separated newline',
...     ' ',
... ])))
>>> assert len(hints) == 1
>>> # ---
>>> hints = list(parse_google_retblock('\n'.join([
...     # Multiple types can be specified
...     'threading.Thread: a description',
...     '(int, str): a tuple of int and str',
...     'tuple: a tuple of int and str',
...     'Tuple[int, str]: a tuple of int and str',
... ])))
>>> assert len(hints) == 4
>>> # ---
>>> hints = list(parse_google_retblock('\n'.join([
...     # If the colon is not specified nothing will be parsed
...     'list',
...     'Tuple[int, str]',
... ])))
>>> assert len(hints) == 0
```

ubelt.meta.docscrape_google.**parse_google_argblock**(*lines*)

Parameters **lines** (*str*) – the unindented lines from an Args docstring section

1.1.1.1.6 References

It is not clear which of these is *the* standard or if there is one https://sphinxcontrib-napoleon.readthedocs.io/en/latest/example_google.html#example-google http://www.sphinx-doc.org/en/stable/ext/example_google.html#example-google

CommandLine: python -m ubelt.meta.docscrape_google parse_google_argblock

1.1.1.1.7 Example

```
>>> from ubelt.meta.docscrape_google import *  # NOQA
>>> # Test various ways that arglines can be written
>>> line_list = [
...     '',
...     'foo1 (int): a description',
...     'foo2: a description\n    with a newline',
...     'foo3 (int or str): a description',
...     'foo4 (int or threading.Thread): a description',
...     #
...     # this is sphynx-like typing style
...     'param1 (:obj:`str`, optional): ',
...     'param2 (:obj:`list` of :obj:`str`):',
...     #
...     # the Type[type] syntax is defined by the python typeing module
...     'attr1 (Optional[int]): Description of `attr1`.',
...     'attr2 (List[str]): Description of `attr2`.',
```

```
...     'attr3 (Dict[str, str]): Description of `attr3`.',
... ]
>>> lines = '\n'.join(line_list)
>>> argdict_list = list(parse_google_argblock(lines))
>>> # All lines except the first should be accepted
>>> assert len(argdict_list) == len(line_list) - 1
>>> assert argdict_list[1]['desc'] == 'a description with a newline'
```

ubelt.meta.docscrape_google.**split_google_docblocks**(*docstr*)

Parameters **docstr** (*str*) – a docstring

Returns

list of 2-tuples where the first item is a google style docstring tag and the second item is the bock corresponding to that tag.

Return type list

1.1.1.1.8 Example

```
>>> from ubelt.meta.docscrape_google import *  # NOQA
>>> docstr = split_google_docblocks.__doc__
>>> groups = split_google_docblocks(docstr)
>>> #print('groups = %s' % (groups,))
>>> assert len(groups) == 3
>>> print([k for k, v in groups])
['Args', 'Returns', 'Example']
```

1.1.1.1.8.1 ubelt.meta.dynamic_analysis module

ubelt.meta.dynamic_analysis.**get_stack_frame**(*n=0*, *strict=True*)

Gets the current stack frame or any of its ancestors dynamically

Parameters

- **n** (*int*) – n=0 means the frame you called this function in. n=1 is the parent frame.
- **strict** (*bool*) – (default = True)

Returns frame_cur

Return type frame

CommandLine: python -m dynamic_analysis get_stack_frame

1.1.1.1.9 Example

```
>>> from ubelt.meta.dynamic_analysis import *  # NOQA
>>> frame_cur = get_stack_frame(n=0)
>>> print('frame_cur = %r' % (frame_cur,))
>>> assert frame_cur.f_globals['frame_cur'] is frame_cur
```

ubelt.meta.dynamic_analysis.**get_parent_frame**(*n=0*)

Returns the frame of that called you. This is equivalent to *get_stack_frame(n=1)*

Parameters **n** (*int*) – n=0 means the frame you called this function in. n=1 is the parent frame.

Returns parent_frame

Return type frame

CommandLine: python -m dynamic_analysis get_parent_frame

1.1.1.1.10 Example

```
>>> from ubelt.meta.dynamic_analysis import *  # NOQA
>>> root0 = get_stack_frame(n=0)
>>> def foo():
>>>     child = get_stack_frame(n=0)
>>>     root1 = get_parent_frame(n=0)
>>>     root2 = get_stack_frame(n=1)
>>>     return child, root1, root2
>>> # Note this wont work in IPython because several
>>> # frames will be inserted between here and foo
>>> child, root1, root2 = foo()
>>> print('root0 = %r' % (root0,))
>>> print('root1 = %r' % (root1,))
>>> print('root2 = %r' % (root2,))
>>> print('child = %r' % (child,))
>>> assert root0 == root1
>>> assert root1 == root2
>>> assert child != root1
```

1.1.1.1.11 Module contents

1.1.2 Submodules

1.1.2.1 ubelt.orderedset module

class ubelt.orderedset.**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.

1.1.2.1.1 References

http://code.activestate.com/recipes/576696/ http://code.activestate.com/recipes/576694/ http://stackoverflow.com/questions/1653970/does-python-have-an-ordered-set

1.1.2.1.2 Example

```
>>> from ubelt.orderedset import *
>>> oset([1, 2, 3])
OrderedSet([1, 2, 3])
```

isdisjoint(*other*)

add(*key*)

Adds an element to the ends of the ordered set if it. This has no effect if the element is already present.

1.1.2.1.3 Example

```
>>> self = OrderedSet()
>>> self.append(3)
>>> print(self)
OrderedSet([3])
```

append(*key*)

Adds an element to the ends of the ordered set if it. This has no effect if the element is already present.

1.1.2.1.4 Notes

This is an alias of *add* for API compatibility with list

1.1.2.1.5 Example

```
>>> self = OrderedSet()
>>> self.append(3)
>>> self.append(2)
>>> self.append(5)
>>> print(self)
OrderedSet([3, 2, 5])
```

discard(*key*)

Remove an element from a set if it is a member. If the element is not a member, do nothing.

1.1.2.1.6 Example

```
>>> self = OrderedSet([1, 2, 3])
>>> self.discard(2)
>>> print(self)
OrderedSet([1, 3])
>>> self.discard(2)
>>> print(self)
OrderedSet([1, 3])
```

pop(*last=True*)

Remove and return a the first or last element in the ordered set. Raises KeyError if the set is empty.

Parameters **last** (*bool*) – if True return the last element otherwise the first (defaults to True).

1.1.2.1.7 Example

```
>>> import pytest
>>> self = oset([2, 3, 1])
>>> assert self.pop(last=True) == 1
>>> assert self.pop(last=False) == 2
>>> assert self.pop() == 3
>>> with pytest.raises(KeyError):
...     self.pop()
```

union(**sets*)
Combines all unique items. Each items order is defined by its first appearance.

1.1.2.1.8 Example

```
>>> self = OrderedSet.union(oset([3, 1, 4, 1, 5]), [1, 3], [2, 0])
>>> print(self)
OrderedSet([3, 1, 4, 5, 2, 0])
>>> self.union([8, 9])
OrderedSet([3, 1, 4, 5, 2, 0, 8, 9])
>>> self | {10}
OrderedSet([3, 1, 4, 5, 2, 0, 10])
```

intersection(**sets*)
Returns elements in common between all sets. Order is defined only by the first set.

1.1.2.1.9 Example

```
>>> self = OrderedSet.intersection(oset([0, 1, 2, 3]), [1, 2, 3])
>>> print(self)
OrderedSet([1, 2, 3])
>>> self.intersection([2, 4, 5], [1, 2, 3, 4])
OrderedSet([2])
```

update(*other*)
Update a set with the union of itself and others. Preserves ordering of *other*.

1.1.2.1.10 Example

```
>>> self = OrderedSet([1, 2, 3])
>>> self.update([3, 1, 5, 1, 4])
>>> print(self)
OrderedSet([1, 2, 3, 5, 4])
```

extend(*other*)
Update a set with the union of itself and others. Preserves ordering of *other*.

1.1.2.1.11 Example

```
>>> self = OrderedSet([1, 2, 3])
>>> self.update([3, 1, 5, 1, 4])
>>> print(self)
OrderedSet([1, 2, 3, 5, 4])
```

index(*item*)
Find the index of *item* in the OrderedSet

1.1.2.1.12 Example

```
>>> import pytest
>>> self = oset([1, 2, 3])
>>> assert self.index(1) == 0
>>> assert self.index(2) == 1
>>> assert self.index(3) == 2
>>> with pytest.raises(IndexError):
...     self[4]
```

copy()
Return a shallow copy of the ordered set.

1.1.2.1.13 Example

```
>>> self = OrderedSet([1, 2, 3])
>>> other = self.copy()
>>> assert self == other and self is not other
```

difference(**sets*)
Returns all elements that are in this set but not the others.

1.1.2.1.14 Example

```
>>> OrderedSet([1, 2, 3]).difference(OrderedSet([2]))
OrderedSet([1, 3])
>>> OrderedSet([1, 2, 3]) - OrderedSet([2])
OrderedSet([1, 3])
```

issubset(*other*)
Report whether another set contains this set.

1.1.2.1.15 Example

```
>>> OrderedSet([1, 2, 3]).issubset({1, 2})
False
>>> OrderedSet([1, 2, 3]).issubset({1, 2, 3, 4})
True
>>> OrderedSet([1, 2, 3]).issubset({1, 4, 3, 5})
False
```

issuperset(*other*)
Report whether this set contains another set.

1.1.2.1.16 Example

```
>>> OrderedSet([1, 2]).issuperset([1, 2, 3])
False
>>> OrderedSet([1, 2, 3, 4]).issuperset({1, 2, 3})
True
>>> OrderedSet([1, 4, 3, 5]).issuperset({1, 2, 3})
False
```

symmetric_difference(*other*)
Return the symmetric difference of two sets as a new set. (I.e. all elements that are in exactly one of the sets.)

1.1.2.1.17 Example

```
>>> self = OrderedSet([1, 4, 3, 5, 7])
>>> other = OrderedSet([9, 7, 1, 3, 2])
>>> self.symmetric_difference(other)
OrderedSet([4, 5, 9, 2])
```

difference_update(**sets*)
Returns a copy of self with items from other removed

1.1.2.1.18 Example

```
>>> self = OrderedSet([1, 2, 3])
>>> self.difference_update(OrderedSet([2]))
>>> print(self)
OrderedSet([1, 3])
```

intersection_update(*other*)
Update a set with the intersection of itself and another. Order depends only on the first element

1.1.2.1.19 Example

```
>>> self = OrderedSet([1, 4, 3, 5, 7])
>>> other = OrderedSet([9, 7, 1, 3, 2])
>>> self.intersection_update(other)
>>> print(self)
OrderedSet([1, 3, 7])
```

symmetric_difference_update(*other*)
Update a set with the intersection of itself and another. Order depends only on the first element

1.1.2.1.20 Example

```
>>> self = OrderedSet([1, 4, 3, 5, 7])
>>> other = OrderedSet([9, 7, 1, 3, 2])
>>> self.symmetric_difference_update(other)
>>> print(self)
OrderedSet([4, 5, 9, 2])
```

ubelt.orderedset.**oset**
alias of *OrderedSet*

1.1.2.2 ubelt.progiter module

A Progress Iterator:

The API is compatible with TQDM!

We have our own ways of running too! You can divide the runtime overhead by two as many times as you want.

CommandLine: python -m ubelt.progiter __doc__:0

1.1.2.2.1 Example

```
>>> # SCRIPT
>>> import ubelt as ub
>>> def is_prime(n):
...     return n >= 2 and not any(n % i == 0 for i in range(2, n))
>>> for n in ub.ProgIter(range(1000000), verbose=1):
>>>     # do some work
>>>     is_prime(n)
```

class ubelt.progiter.**ProgIter**(*iterable=None*, *desc=None*, *total=None*, *freq=1*, *initial=0*, *eta_window=64*, *clearline=True*, *adjust=True*, *time_thresh=2.0*, *show_times=True*, *enabled=True*, *verbose=None*, *stream=None*, ***kwargs*)

Bases: ubelt.progiter._TQDMCompat, ubelt.progiter._BackwardsCompat

Prints progress as an iterator progresses

Note: USE *tqdm* INSTEAD. The main difference between *ProgIter* and *tqdm* is that ProgIter does not use threading where as *tqdm* does. *ProgIter* is simpler than *tqdm* and thus more stable in certain circumstances. However, *tqdm* is recommended for the majority of use cases.

Note: The API on *ProgIter* will change to become inter-compatible with *tqdm*.

Variables

- ***iterable*** (iterable) – An iterable iterable
- **desc** (*str*) – description label to show with progress
- **total** (*int*) – Maximum length of the process (estimated from iterable if not specified)

- **freq** (*int*) – How many iterations to wait between messages.
- **adjust** (*bool*) – if True freq is adjusted based on time_thresh
- **eta_window** (*int*) – number of previous measurements to use in eta calculation
- **clearline** (*bool*) – if true messages are printed on the same line
- **adjust** – if True *freq* is adjusted based on time_thresh
- **time_thresh** (*float*) – desired amount of time to wait between messages if adjust is True otherwise does nothing
- **show_times** (*bool*) – shows rate, eta, and wall (defaults to True)
- **initial** (*int*) – starting index offset (defaults to 0)
- **stream** (*file*) – defaults to sys.stdout
- **enabled** (*bool*) – if False nothing happens.
- **verbose** (*int*) – verbosity mode 0 - no verbosity, 1 - verbosity with clearline=True and adjust=True 2 - verbosity without clearline=False and adjust=True 3 - verbosity without clearline=False and adjust=False

SeeAlso: tqdm - https://pypi.python.org/pypi/tqdm

Reference: http://datagenetics.com/blog/february12017/index.html

1.1.2.2.2 Notes

Either use ProgIter in a with statement or call prog.end() at the end of the computation if there is a possibility that the entire iterable may not be exhausted.

1.1.2.2.3 Example

```
>>>
>>> import ubelt as ub
>>> def is_prime(n):
...     return n >= 2 and not any(n % i == 0 for i in range(2, n))
>>> for n in ub.ProgIter(range(100), verbose=1):
>>>     # do some work
>>>     is_prime(n)
100/100... rate=... Hz, total=..., wall=... EST
```

set_extra(*extra*)

specify a custom info appended to the end of the next message TODO: come up with a better name and rename

1.1.2.2.4 Example

```
>>> import ubelt as ub
>>> prog = ub.ProgIter(range(100, 300, 100), show_times=False, verbose=3)
>>> for n in prog:
>>>     prog.set_extra('processesing num {}'.format(n))
0/2...
```

```
1/2...processesing num 100
2/2...processesing num 200
```

step(*inc=1*)
Manually step progress update, either directly or by an increment.

Parameters

- **idx** (*int*) – current step index (default None) if specified, takes precidence over *inc*
- **inc** (*int*) – number of steps to increment (defaults to 1)

1.1.2.2.5 Example

```
>>> import ubelt as ub
>>> n = 3
>>> prog = ub.ProgIter(desc='manual', total=n, verbose=3)
>>> # Need to manually begin and end in this mode
>>> prog.begin()
>>> for _ in range(n):
...     prog.step()
>>> prog.end()
```

1.1.2.2.6 Example

```
>>> import ubelt as ub
>>> n = 3
>>> # can be used as a context manager in manual mode
>>> with ub.ProgIter(desc='manual', total=n, verbose=3) as prog:
...     for _ in range(n):
...         prog.step()
```

begin()
Initializes information used to measure progress

end()

format_message()
builds a formatted progres message with the current values. This contains the special characters needed to clear lines.

CommandLine: python -m ubelt.progiter ProgIter.format_message

1.1.2.2.7 Example

```
>>> self = ProgIter(clearline=False, show_times=False)
>>> print(repr(self.format_message()))
'    0/?... \n'
>>> self.begin()
>>> self.step()
>>> print(repr(self.format_message()))
' 1/?... \n'
```

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

1.1.2.2.8 Example

```
>>> # Unsafe version may write your message on the wrong line
>>> import ubelt as ub
>>> prog = ub.ProgIter(range(4), show_times=False, verbose=1)
>>> for n in prog:
...     print('unsafe message')
 0/4...  unsafe message
 1/4...  unsafe message
unsafe message
unsafe message
 4/4...
>>> # apparently the safe version does this too.
>>> print('---')
---
>>> prog = ub.ProgIter(range(4), show_times=False, verbose=1)
>>> for n in prog:
...     prog.ensure_newline()
...     print('safe message')
 0/4...
safe message
 1/4...
safe message
safe message
safe message
 4/4...
```

display_message()
Writes current progress to the output stream

1.1.2.3 ubelt.util_arg module

ubelt.util_arg.**argval**(*key*, *default=NoParam*, *argv=None*)
Get the value of a keyword argument specified on the command line.

Values can be specified as <*key*> <*value*> or <*key*>=<*value*>

Parameters

- **key** (*str or tuple*) – string or tuple of strings. Each key should be prefixed with two hyphens (i.e. –)
- **default** (*object*) – value to return if not specified
- **argv** (*list*) – overrides *sys.argv* if specified

Returns

value [the value specified after the key. It they key is] specified multiple times, then the first value is returned.

Return type str

Doctest:

```
>>> import ubelt as ub
>>> argv = ['--ans', '42', '--quest=the grail', '--ans=6', '--bad']
>>> assert ub.argval('--spam', argv=argv) == ub.NoParam
>>> assert ub.argval('--quest', argv=argv) == 'the grail'
>>> assert ub.argval('--ans', argv=argv) == '42'
>>> assert ub.argval('--bad', argv=argv) == ub.NoParam
>>> assert ub.argval(('--bad', '--bar'), argv=argv) == ub.NoParam
```

ubelt.util_arg.**argflag**(*key*, *argv=None*)

Determines if a key is specified on the command line

Parameters

- **key** (*str or tuple*) – string or tuple of strings. Each key should be prefixed with two hyphens (i.e. –)
- **argv** (*list*) – overrides *sys.argv* if specified

Returns flag : True if the key (or any of the keys) was specified

Return type bool

Doctest:

```
>>> import ubelt as ub
>>> argv = ['--spam', '--eggs', 'foo']
>>> assert ub.argflag('--eggs', argv=argv) is True
>>> assert ub.argflag('--ans', argv=argv) is False
>>> assert ub.argflag('foo', argv=argv) is True
>>> assert ub.argflag(('bar', '--spam'), argv=argv) is True
```

1.1.2.4 ubelt.util_cache module

class ubelt.util_cache.**Cacher**(*fname*, *cfgstr=None*, *dpath=None*, *appname='ubelt'*, *ext='.pkl'*, *meta=None*, *verbose=None*, *enabled=True*, *log=None*, *protocol=2*)

Bases: object

Cacher designed to be quickly integrated into existing scripts.

Parameters

- **fname** (*str*) – A file name. This is the prefix that will be used by the cache. It will alwasys be used as-is.
- **cfgstr** (*str*) – indicates the state. Either this string or a hash of this string will be used to identify the cache. A cfgstr should always be reasonably readable, thus it is good practice to hash extremely detailed cfgstrs to a reasonable readable level. Use meta to store make original details persist.
- **dpath** (*str*) – Specifies where to save the cache. If unspecified, Cacher defaults to an application resource dir as given by appname.
- **appname** (*str*) – application name (default = 'ubelt') Specifies a folder in the application resource directory where to cache the data if dpath is not specified.
- **ext** (*str*) – extension (default = '.pkl')
- **meta** (*object*) – cfgstr metadata that is also saved with the cfgstr. This data is not used in the hash, but if useful to send in if the cfgstr itself contains hashes.

- **verbose** (*int*) – level of verbosity. Can be 1, 2 or 3. (default=1)
- **enabled** (*bool*) – if set to False, then the load and save methods will do nothing. (default = True)
- **log** (*func*) – overloads the print function. Useful for sending output to loggers (e.g. logging.info, tqdm.tqdm.write, ...)
- **protocol** (*int*) – protocol version used by pickle. If python 2 compatibility is not required, then it is better to use protocol 4. (default=2)

CommandLine: python -m ubelt.util_cache Cacher

1.1.2.4.1 Example

```
>>> import ubelt as ub
>>> cfgstr = 'repr-of-params-that-uniquely-determine-the-process'
>>> # Create a cacher and try loading the data
>>> cacher = ub.Cacher('test_process', cfgstr)
>>> cacher.clear()
>>> data = cacher.tryload()
>>> if data is None:
>>>     # Put expensive functions in if block when cacher misses
>>>     myvar1 = 'result of expensive process'
>>>     myvar2 = 'another result'
>>>     # Tell the cacher to write at the end of the if block
>>>     # It is idomatic to put results in a tuple named data
>>>     data = myvar1, myvar2
>>>     cacher.save(data)
>>> # Last part of the Cacher pattern is to unpack the data tuple
>>> myvar1, myvar2 = data
```

1.1.2.4.2 Example

```
>>> # The previous example can be shorted if only a single value
>>> from ubelt.util_cache import Cacher
>>> cfgstr = 'repr-of-params-that-uniquely-determine-the-process'
>>> # Create a cacher and try loading the data
>>> cacher = Cacher('test_process', cfgstr)
>>> myvar = cacher.tryload()
>>> if myvar is None:
>>>     myvar = ('result of expensive process', 'another result')
>>>     cacher.save(myvar)
>>> assert cacher.exists(), 'should now exist'
```

VERBOSE = 1

get_fpath(*cfgstr=None*)

Reports the filepath that the cacher will use. It will attempt to use '{fname}_{cfgstr}{ext}' unless that is too long. Then cfgstr will be hashed.

1.1.2.4.3 Example

```
>>> from ubelt.util_cache import Cacher
>>> import pytest
>>> with pytest.warns(UserWarning):
>>>     cacher = Cacher('test_cacher1')
>>>     cacher.get_fpath()
>>> self = Cacher('test_cacher2', cfgstr='cfg1')
>>> self.get_fpath()
>>> self = Cacher('test_cacher3', cfgstr='cfg1' * 32)
>>> self.get_fpath()
```

exists(*cfgstr=None*)
Check to see if the cache exists

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

1.1.2.4.4 Example

```
>>> from ubelt.util_cache import Cacher
>>> # Ensure that some data exists
>>> known_fnames = set()
>>> cacher = Cacher('versioned_data', cfgstr='1')
>>> cacher.ensure(lambda: 'data1')
>>> known_fnames.add(cacher.get_fpath())
>>> cacher = Cacher('versioned_data', cfgstr='2')
>>> cacher.ensure(lambda: 'data2')
>>> known_fnames.add(cacher.get_fpath())
>>> # List previously computed configs for this type
>>> from os.path import basename
>>> cacher = Cacher('versioned_data', cfgstr='2')
>>> exist_fpaths = set(cacher.existing_versions())
>>> exist_fnames = list(map(basename, exist_fpaths))
>>> print(exist_fnames)
>>> assert exist_fpaths == known_fnames
```

['versioned_data_1.pkl', 'versioned_data_2.pkl']

clear(*cfgstr=None*)
Removes the saved cache and metadata from disk

tryload(*cfgstr=None*, *on_error='raise'*)
Like load, but returns None if the load fails due to a cache miss.

Parameters **on_error** (*str*) – how to handle non-io errors errors. Either raise, which re-raises the exception, or clear which clears the cache and returns None.

load(*cfgstr=None*)

1.1.2.4.5 Example

```
>>> from ubelt.util_cache import *  # NOQA
>>> # Setting the cacher as enabled=False turns it off
>>> cacher = Cacher('test_disabled_load', '', enabled=True)
```

```
>>> cacher.save('data')
>>> assert cacher.load() == 'data'
>>> cacher.enabled = False
>>> assert cacher.tryload() is None
```

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

Writes data to path specified by *self.fpath(cfgstr)*.

Metadata containing information about the cache will also be appended to an adjacent file with the *.meta* suffix.

1.1.2.4.6 Example

```
>>> from ubelt.util_cache import *  # NOQA
>>> # Normal functioning
>>> cfgstr = 'long-cfg' * 32
>>> cacher = Cacher('test_enabled_save', cfgstr)
>>> cacher.save('data')
>>> assert exists(cacher.get_fpath()), 'should be enabeled'
>>> assert exists(cacher.get_fpath() + '.meta'), 'missing metadata'
>>> # Setting the cacher as enabled=False turns it off
>>> cacher2 = Cacher('test_disabled_save', 'params', enabled=False)
>>> cacher2.save('data')
>>> assert not exists(cacher2.get_fpath()), 'should be disabled'
```

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

Wraps around a function. A cfgstr must be stored in the base cacher.

Parameters

- **func** (*callable*) – function that will compute data on cache miss
- ***args** – passed to func
- ****kwargs** – passed to func

1.1.2.4.7 Example

```
>>> from ubelt.util_cache import *  # NOQA
>>> def func():
>>>     return 'expensive result'
>>> fname = 'test_cacher_ensure'
>>> cfgstr = 'func params'
>>> cacher = Cacher(fname, cfgstr)
>>> cacher.clear()
>>> data1 = cacher.ensure(func)
>>> data2 = cacher.ensure(func)
>>> assert data1 == 'expensive result'
>>> assert data1 == data2
>>> cacher.clear()
```

1.1.2.4.8 Example

```
>>> from ubelt.util_cache import *  # NOQA
>>> @Cacher(fname, cfgstr).ensure
>>> def func():
>>>     return 'expensive result'
```

1.1.2.5 ubelt.util_cmd module

ubelt.util_cmd.**cmd**(*command*, *shell=False*, *detatch=False*, *verbose=0*, *verbout=None*, *tee='auto'*)

Executes a command in a subprocess.

The advantage of this wrapper around subprocess is that (1) you control if the subprocess prints to stdout, (2) the text written to stdout and stderr is returned for parsing, (3) cross platform behavior that lets you specify the command as a string or tuple regardless of whether or not shell=True. (4) ability to detatch, return the process object and allow the process to run in the background (eventually we may return a Future object instead).

Parameters

- **command** (*str*) – bash-like command string or tuple of executable and args
- **shell** (*bool*) – if True, process is run in shell
- **detatch** (*bool*) – if True, process is detached and run in background.
- **verbose** (*int*) – verbosity mode. Can be 0, 1, 2, or 3.
- **verbout** (*int*) – if True, *command* writes to stdout in realtime. defaults to True iff verbose > 0. Note when detatch is True all stdout is lost.
- **tee** (*str*) – backend for tee output. Can be either: auto, select (POSIX only), or thread.

Returns

info - information about command status. if detatch is False *info* contains captured standard out, standard error, and the return code if detatch is False *info* contains a reference to the process.

Return type dict

1.1.2.5.1 Notes

Inputs can either be text or tuple based. On unix we ensure conversion to text if shell=True, and to tuple if shell=False. On windows, the input is always text based. See [3] for a potential cross-platform shlex solution for windows.

CommandLine: python -m ubelt.util_cmd cmd python -c "import ubelt as ub; ub.cmd('ping localhost -c 2', verbose=2)"

1.1.2.5.2 References

[1] https://stackoverflow.com/questions/11495783/redirect-subprocess-stderr-to-stdout [2] https://stackoverflow.com/questions/7729336/how-can-i-print-and-display-subprocess-stdout-and-stderr-output-without-distorti [3] https://stackoverflow.com/questions/33560364/python-windows-parsing-command-lines-with-shlex

1.1.2.5.3 Example

```
>>> info = cmd(('echo', 'simple cmdline interface'), verbose=1)
simple cmdline interface
>>> assert info['ret'] == 0
>>> assert info['out'].strip() == 'simple cmdline interface'
>>> assert info['err'].strip() == ''
```

Doctest:

```
>>> info = cmd('echo str noshell', verbose=0)
>>> assert info['out'].strip() == 'str noshell'
```

Doctest:

```
>>> # windows echo will output extra single quotes
>>> info = cmd(('echo', 'tuple noshell'), verbose=0)
>>> assert info['out'].strip().strip("'") == 'tuple noshell'
```

Doctest:

```
>>> # Note this command is formatted to work on win32 and unix
>>> info = cmd('echo str&&echo shell', verbose=0, shell=True)
>>> assert info['out'].strip() == 'str\nshell'
```

Doctest:

```
>>> info = cmd(('echo', 'tuple shell'), verbose=0, shell=True)
>>> assert info['out'].strip().strip("'") == 'tuple shell'
```

Doctest:

```
>>> import ubelt as ub
>>> from os.path import join, exists
>>> fpath1 = join(ub.get_app_cache_dir('ubelt'), 'cmdout1.txt')
>>> fpath2 = join(ub.get_app_cache_dir('ubelt'), 'cmdout2.txt')
>>> ub.delete(fpath1)
>>> ub.delete(fpath2)
>>> info1 = ub.cmd(('touch', fpath1), detatch=True)
>>> info2 = ub.cmd('echo writing2 > ' + fpath2, shell=True, detatch=True)
>>> while not exists(fpath1):
...     pass
>>> while not exists(fpath2):
...     pass
>>> assert ub.readfrom(fpath1) == ''
>>> assert ub.readfrom(fpath2).strip() == 'writing2'
>>> info1['proc'].wait()
>>> info2['proc'].wait()
```

1.1.2.6 ubelt.util_colors module

ubelt.util_colors.**highlight_code**(*text*, *lexer_name='python'*, ***kwargs*)

Highlights a block of text using ansii tags based on language syntax.

Parameters

- **text** (*str*) – plain text to highlight
- **lexer_name** (*str*) – name of language
- ****kwargs** – passed to pygments.lexers.get_lexer_by_name

Returns

text [highlighted text] If pygments is not installed, the plain text is returned.

Return type str

CommandLine: python -c "import pygments.formatters; print(list(pygments.formatters.get_all_formatters()))"

1.1.2.6.1 Example

```
>>> import ubelt as ub
>>> text = 'import ubelt as ub; print(ub)'
>>> new_text = ub.highlight_code(text)
>>> print(new_text)
```

ubelt.util_colors.**color_text**(*text*, *color*)

Colorizes text a single color using ansii tags.

Parameters

- **text** (*str*) – text to colorize
- **color** (*str*) – may be one of the following: 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

Returns

text [colorized text.] If pygments is not installed plain text is returned.

Return type str

CommandLine: python -c "import pygments.console; print(sorted(pygments.console.codes.keys()))" python -m ubelt.util_colors color_text

1.1.2.6.2 Example

```
>>> from ubelt.util_colors import *  # NOQA
>>> text = 'raw text'
>>> assert color_text(text, 'red') == '\x1b[31;01mraw text\x1b[39;49;00m'
>>> assert color_text(text, None) == 'raw text'
```

1.1.2.7 ubelt.util_const module

This module defines *ub.NoParam*. This is a robust setinal value that can act like None when None might be a valid value. The value of *NoParam* is robust to reloading, pickling, and copying (i.e. var is ub.NoParam will return True after these operations)

1.1.2.8 ubelt.util_dict module

ubelt.util_dict.**odict**
alias of OrderedDict

ubelt.util_dict.**ddict**
alias of defaultdict

class ubelt.util_dict.**AutoDict**
Bases: dict

An infinitely nested default dict of dicts.

Implementation of perl's autovivification feature.

SeeAlso: ub.AutoOrderedDict - the ordered version

1.1.2.8.1 References

http://stackoverflow.com/questions/651794/init-dict-of-dicts

1.1.2.8.2 Example

```
>>> import ubelt as ub
>>> auto = ub.AutoDict()
>>> auto[0][10][100] = None
>>> assert str(auto) == '{0: {10: {100: None}}}'
```

to_dict()
Recursively casts a AutoDict into a regular dictionary. All nested AutoDict values are also converted.

Returns a copy of this dict without autovivification

Return type dict

1.1.2.8.3 Example

```
>>> from ubelt.util_dict import AutoDict
>>> auto = AutoDict()
>>> auto[1] = 1
>>> auto['n1'] = AutoDict()
>>> static = auto.to_dict()
>>> assert not isinstance(static, AutoDict)
>>> assert not isinstance(static['n1'], AutoDict)
```

class ubelt.util_dict.**AutoOrderedDict**
Bases: collections.OrderedDict, *ubelt.util_dict.AutoDict*

An an infinitely nested default dict of dicts that maintains the ordering of items.

SeeAlso: ub.AutoDict - the unordered version

Example0:

```
>>> import ubelt as ub
>>> auto = ub.AutoOrderedDict()
>>> auto[0][3] = 3
>>> auto[0][2] = 2
>>> auto[0][1] = 1
>>> assert list(auto[0].values()) == [3, 2, 1]
```

ubelt.util_dict.**dzip**(*items1*, *items2*)

Zips elementwise pairs between items1 and items2 into a dictionary. Values from items2 can be broadcast onto items1.

Parameters

- **items1** (*Sequence*) – full sequence
- **items2** (*Sequence*) – can either be a sequence of one item or a sequence of equal length to *items1*

Returns similar to dict(zip(items1, items2))

Return type dict

1.1.2.8.4 Example

```
>>> 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}
>>> assert dzip([], [4]) == {}
```

ubelt.util_dict.**group_items**(*item_list*, *groupid_list*, *sorted_=True*)

Groups a list of items by group id.

Parameters

- **item_list** (*list*) – a list of items to group
- **groupid_list** (*list*) – a corresponding list of item groupids
- **sorted_** (*bool*) – if True preserves the ordering of items within groups (default = True)

Todo:

- [ ] change names from item_list->values and groupid_list->keys
- **[ ] allow keys to be an iterable or a function so this can work** similar to itertools.groupby

---

Returns groupid_to_items: maps a groupid to a list of items

Return type dict

CommandLine: python -m ubelt.util_dict group_items

1.1.2.8.5 Example

```
>>> import ubelt as ub
>>> item_list    = ['ham',     'jam',   'spam',     'eggs',    'cheese', 'banana']
>>> groupid_list = ['protein', 'fruit', 'protein',  'protein', 'dairy',  'fruit']
>>> groupid_to_items = ub.group_items(item_list, groupid_list)
>>> print(ub.repr2(groupid_to_items, nl=0))
{'dairy': ['cheese'], 'fruit': ['jam', 'banana'], 'protein': ['ham', 'spam', 'eggs
↪']}
```

ubelt.util_dict.**dict_hist**(*item_list*, *weight_list=None*, *ordered=False*, *labels=None*)

Builds a histogram of items

Parameters

- **item_list** (*list*) – list with hashable items (usually containing duplicates)
- **weight_list** (*list*) – list of weights for each items
- **ordered** (*bool*) – if True the result is ordered by frequency
- **labels** (*list*) – expected labels (default None) if specified the frequency of each label is initialized to zero and item_list can only contain items specified in labels.

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 ubelt.util_dict dict_hist

1.1.2.8.6 Example

```
>>> import ubelt as ub
>>> item_list = [1, 2, 39, 900, 1232, 900, 1232, 2, 2, 2, 900]
>>> hist = ub.dict_hist(item_list)
>>> print(ub.repr2(hist, nl=0))
{1: 1, 2: 4, 39: 1, 900: 3, 1232: 2}
```

1.1.2.8.7 Example

```
>>> import ubelt as ub
>>> item_list = [1, 2, 39, 900, 1232, 900, 1232, 2, 2, 2, 900]
>>> hist1 = ub.dict_hist(item_list)
>>> hist2 = ub.dict_hist(item_list, ordered=True)
>>> try:
>>>     hist3 = ub.dict_hist(item_list, labels=[])
>>> except KeyError:
>>>     pass
>>> else:
>>>     raise AssertionError('expected key error')
>>> #result = ub.repr2(hist_)
>>> weight_list = [1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1]
>>> hist4 = ub.dict_hist(item_list, weight_list=weight_list)
>>> print(ub.repr2(hist1, nl=0))
{1: 1, 2: 4, 39: 1, 900: 3, 1232: 2}
```

```
>>> print(ub.repr2(hist4, nl=0))
{1: 1, 2: 4, 39: 1, 900: 1, 1232: 0}
```

ubelt.util_dict.**find_duplicates**(*items*, *k=2*)

Find all duplicate items in a list.

Search for all items that appear more than *k* times and return a mapping from each duplicate item to the positions it appeared in.

Parameters

- **items** (*list*) – a list of hashable items possibly containing duplicates
- **k** (*int*) – only return items that appear at least *k* times (default=2)

Returns maps each duplicate item to the indices at which it appears

Return type dict

CommandLine: python -m ubelt.util_dict find_duplicates

1.1.2.8.8 Example

```
>>> import ubelt as ub
>>> items = [0, 0, 1, 2, 3, 3, 0, 12, 2, 9]
>>> duplicates = ub.find_duplicates(items)
>>> print('items = %r' % (items,))
>>> print('duplicates = %r' % (duplicates,))
>>> assert duplicates == {0: [0, 1, 6], 2: [3, 8], 3: [4, 5]}
>>> assert ub.find_duplicates(items, 3) == {0: [0, 1, 6]}
```

1.1.2.8.9 Example

```
>>> import ubelt as ub
>>> items = [0, 0, 1, 2, 3, 3, 0, 12, 2, 9]
>>> # note: k can be 0
>>> duplicates = ub.find_duplicates(items, k=0)
>>> print(ub.repr2(duplicates, nl=0))
{0: [0, 1, 6], 1: [2], 2: [3, 8], 3: [4, 5], 9: [9], 12: [7]}
```

ubelt.util_dict.**dict_subset**(*dict_*, *keys*, *default=NoParam*)

Get a subset of a dictionary

Parameters

- **dict_** (*dict*) – superset dictionary
- **keys** (*list*) – keys to take from *dict_*

Returns subset dictionary

Return type dict

1.1.2.8.10 Example

```
>>> import ubelt as ub
>>> dict_ = {'K': 3, 'dcvs_clip_max': 0.2, 'p': 0.1}
>>> keys = ['K', 'dcvs_clip_max']
>>> subdict_ = ub.dict_subset(dict_, keys)
>>> print(ub.repr2(subdict_, nl=0))
{'K': 3, 'dcvs_clip_max': 0.2}
```

ubelt.util_dict.**dict_take**(*dict_*, *keys*, *default=NoParam*)

Generates values from a dictionary

Parameters

- **dict_** (*dict*)
- **keys** (*list*)
- **default** (*Optional*) – if specified uses default if keys are missing

CommandLine: python -m ubelt.util_dict dict_take_gen

1.1.2.8.11 Example

```
>>> import ubelt as ub
>>> dict_ = {1: 'a', 2: 'b', 3: 'c'}
>>> keys = [1, 2, 3, 4, 5]
>>> result = list(ub.dict_take(dict_, keys, None))
>>> assert result == ['a', 'b', 'c', None, None]
```

1.1.2.8.12 Example

```
>>> import ubelt as ub
>>> dict_ = {1: 'a', 2: 'b', 3: 'c'}
>>> keys = [1, 2, 3, 4, 5]
>>> try:
>>>     print(list(ub.dict_take(dict_, keys)))
>>>     raise AssertionError('did not get key error')
>>> except KeyError:
>>>     print('correctly got key error')
```

ubelt.util_dict.**dict_union**(**args*)

Combines the disjoint keys in multiple dictionaries. For intersecting keys, dictionaries towards the end of the sequence are given precidence.

Parameters ***args** – a sequence of dictionaries

Returns OrderedDict if the first argument is an OrderedDict, otherwise dict

1.1.2.8.13 Example

```
>>> result = dict_union({'a': 1, 'b': 1}, {'b': 2, 'c': 2})
>>> assert result == {'a': 1, 'b': 2, 'c': 2}
>>> dict_union(odict([('a', 1), ('b', 2)]), odict([('c', 3), ('d', 4)]))
OrderedDict([('a', 1), ('b', 2), ('c', 3), ('d', 4)])
>>> dict_union()
{}
```

ubelt.util_dict.**map_vals**(*func*, *dict_*)
applies a function to each of the keys in a dictionary

Parameters

- **func** (*callable*) – a function or indexable object
- **dict_** (*dict*) – a dictionary

Returns transformed dictionary

Return type newdict

CommandLine: python -m ubelt.util_dict map_vals

1.1.2.8.14 Example

```
>>> import ubelt as ub
>>> dict_ = {'a': [1, 2, 3], 'b': []}
>>> func = len
>>> newdict = ub.map_vals(func, dict_)
>>> assert newdict ==  {'a': 3, 'b': 0}
>>> print(newdict)
>>> # Can also use indexables as `func`
>>> dict_ = {'a': 0, 'b': 1}
>>> func = [42, 21]
>>> newdict = ub.map_vals(func, dict_)
>>> assert newdict ==  {'a': 42, 'b': 21}
>>> print(newdict)
```

ubelt.util_dict.**map_keys**(*func*, *dict_*)
applies a function to each of the keys in a dictionary

Parameters

- **func** (*callable*) – a function or indexable object
- **dict_** (*dict*) – a dictionary

Returns transformed dictionary

Return type newdict

CommandLine: python -m ubelt.util_dict map_keys

1.1.2.8.15 Example

```
>>> import ubelt as ub
>>> dict_ = {'a': [1, 2, 3], 'b': []}
>>> func = ord
```

```
>>> newdict = ub.map_keys(func, dict_)
>>> print(newdict)
>>> assert newdict == {97: [1, 2, 3], 98: []}
>>> #ut.assert_raises(AssertionError, map_keys, len, dict_)
>>> dict_ = {0: [1, 2, 3], 1: []}
>>> func = ['a', 'b']
>>> newdict = ub.map_keys(func, dict_)
>>> print(newdict)
>>> assert newdict == {'a': [1, 2, 3], 'b': []}
>>> #ut.assert_raises(AssertionError, map_keys, len, dict_)
```

ubelt.util_dict.**invert_dict**(*dict_*, *unique_vals=True*)

Swaps the keys and values in a dictionary.

Parameters

- **dict_** (*dict*) – dictionary to invert
- **unique_vals** (*bool*) – if False, inverted keys are returned in a set. The default is True.

Returns inverted_dict

Return type dict

1.1.2.8.16 Notes

The must values be hashable.

If the original dictionary contains duplicate values, then only one of the corresponding keys will be returned and the others will be discarded. This can be prevented by setting *unique_vals=True*, causing the inverted keys to be returned in a set.

CommandLine: python -m ubelt.util_dict invert_dict

1.1.2.8.17 Example

```
>>> import ubelt as ub
>>> dict_ = {'a': 1, 'b': 2}
>>> inverted_dict = ub.invert_dict(dict_)
>>> assert inverted_dict == {1: 'a', 2: 'b'}
```

1.1.2.8.18 Example

```
>>> import ubelt as ub
>>> dict_ = ub.odict([(2, 'a'), (1, 'b'), (0, 'c'), (None, 'd')])
>>> inverted_dict = ub.invert_dict(dict_)
>>> assert list(inverted_dict.keys())[0] == 'a'
```

1.1.2.8.19 Example

```
>>> import ubelt as ub
>>> dict_ = {'a': 1, 'b': 0, 'c': 0, 'd': 0, 'f': 2}
>>> inverted_dict = ub.invert_dict(dict_, unique_vals=False)
>>> assert inverted_dict == {0: {'b', 'c', 'd'}, 1: {'a'}, 2: {'f'}}
```

1.1.2.9 ubelt.util_download module

Helpers for downloading data

ubelt.util_download.**download**(*url*, *fpath=None*, *hash_prefix=None*, *chunksize=8192*, *verbose=1*)
downloads a url to a fpath.

Parameters

- **url** (*str*) – url to download
- **fpath** (*str*) – path to download to. Defaults to basename of url and ubelt's application cache.
- **chunksize** (*int*) – download chunksize
- **verbose** (*bool*) – verbosity

1.1.2.9.1 Notes

Original code taken from pytorch in torch/utils/model_zoo.py and slightly modified.

1.1.2.9.2 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

1.1.2.9.3 Example

```
>>> from ubelt.util_download import *  # NOQA
>>> url = 'http://i.imgur.com/rqwaDag.png'
>>> fpath = download(url)
>>> print(basename(fpath))
rqwaDag.png
```

ubelt.util_download.**grabdata**(*url*, *fpath=None*, *dpath=None*, *fname=None*, *redo=False*, *verbose=1*, *appname=None*, ***download_kw*)
Downloads a file, caches it, and returns its local path.

Parameters

- **url** (*str*) – url to the file to download
- **fpath** (*str*) – The full path to download the file to. If unspecified, the arguments *dpath* and *fname* are used to determine this.
- **dpath** (*str*) – where to download the file. If unspecified *appname* is used to determine this. Mutually exclusive with fpath.

- **fname** (*str*) – What to name the downloaded file. Defaults to the url basename. Mutually exclusive with fpath.
- **redo** (*bool*) – if True forces redownload of the file (default = False)
- **verbose** (*bool*) – verbosity flag (default = True)
- **appname** (*str*) – set dpath to *ub.get_app_cache_dir(appname)*. Mutually exclusive with dpath and fpath.
- ****download_kw** – additional kwargs to pass to ub.download

Returns fpath - file path string

Return type str

1.1.2.9.4 Example

```
>>> import ubelt as ub
>>> file_url = 'http://i.imgur.com/rqwaDag.png'
>>> lena_fpath = ub.grabdata(file_url, fname='mario.png')
>>> result = basename(lena_fpath)
>>> print(result)
mario.png
```

1.1.2.10 ubelt.util_format module

ubelt.util_format.**repr2**(*val*, ***kwargs*)

Constructs a "pretty" string representation.

This is an alternative to repr, and *pprint.pformat* that attempts to be both more configurable and generate output that is consistent between python versions.

Parameters

- **val** (*object*) – an arbitrary python object
- ****kwargs** – si, stritems, strkeys, strvals, sk, sv, nl, newlines, nobr, nobraces, cbr, compact_brace, trailsep, trailing_sep, explicit, itemsep, precision, kvsep, sort

Returns output string

Return type str

CommandLine: python -m ubelt.util_format repr2:0 python -m ubelt.util_format repr2:1

1.1.2.10.1 Example

```
>>> from ubelt.util_format import *
>>> import ubelt as ub
>>> dict_ = {
...     'custom_types': [slice(0, 1, None), 1/3],
...     'nest_dict': {'k1': [1, 2, {3: {4, 5}}],
...                   'key2': [1, 2, {3: {4, 5}}],
...                   'key3': [1, 2, {3: {4, 5}}],
...                   },
...     'nest_dict2': {'k': [1, 2, {3: {4, 5}}]},
```

```
...     'nested_tuples': [tuple([1]), tuple([2, 3]), frozenset([4, 5, 6])],
...     'one_tup': tuple([1]),
...     'simple_dict': {'spam': 'eggs', 'ham': 'jam'},
...     'simple_list': [1, 2, 'red', 'blue'],
...     'odict': ub.odict([(1, '1'), (2, '2')]),
... }
>>> result = repr2(dict_, nl=3, precision=2); print(result)
>>> result = repr2(dict_, nl=2, precision=2); print(result)
>>> result = repr2(dict_, nl=1, precision=2); print(result)
>>> result = repr2(dict_, nl=1, precision=2, itemsep='', explicit=True);␣
↪print(result)
>>> result = repr2(dict_, nl=1, precision=2, nobr=1, itemsep='', explicit=True);␣
↪print(result)
>>> result = repr2(dict_, nl=3, precision=2, cbr=True); print(result)
>>> result = repr2(dict_, nl=3, precision=2, si=True); print(result)
>>> result = repr2(dict_, nl=3, sort=True); print(result)
>>> result = repr2(dict_, nl=3, sort=False, trailing_sep=False); print(result)
>>> result = repr2(dict_, nl=3, sort=False, trailing_sep=False, nobr=True);␣
↪print(result)
```

1.1.2.10.2 Example

```
>>> from ubelt.util_format import *
>>> def _nest(d, w):
...     if d == 0:
...         return {}
...     else:
...         return {'n{}'.format(d): _nest(d - 1, w + 1), 'm{}'.format(d): _
↪nest(d - 1, w + 1)}
>>> dict_ = _nest(d=4, w=1)
>>> result = repr2(dict_, nl=6, precision=2, cbr=1)
>>> print('---')
>>> print(result)
```

1.1.2.11 ubelt.util_func module

Helpers for functional programming

ubelt.util_func.**identity**(*arg*)

The identity function. Simply returns its inputs.

1.1.2.11.1 Example

```
>>> assert identity(42) == 42
```

ubelt.util_func.**inject_method**(*self*, *func*, *name=None*)

Injects a function into an object instance as a bound method

Parameters

- **self** (*object*) – instance to inject a function into
- **func** (*func*) – the function to inject (must contain an arg for self)

- **name** (*str*) – name of the method. optional. If not specified the name of the function is used.

1.1.2.11.2 Example

```
>>> class Foo(object):
>>>     def bar(self):
>>>         return 'bar'
>>> def baz(self):
>>>     return 'baz'
>>> self = Foo()
>>> assert self.bar() == 'bar'
>>> assert not hasattr(self, 'baz')
>>> inject_method(self, baz)
>>> assert not hasattr(Foo, 'baz'), 'should only change one instance'
>>> assert self.baz() == 'baz'
>>> inject_method(self, baz, 'bar')
>>> assert self.bar() == 'baz'
```

1.1.2.12 ubelt.util_hash module

Wrappers around hashlib functions to generate hash signatures for common data.

The hashes should be determenistic across platforms.

Note: The exact hashes generated for data object and files may change in the future. When this happens the *HASH_VERSION* attribute will be incremented.

---

ubelt.util_hash.**hash_data**(*data*, *hasher=NoParam*, *hashlen=NoParam*, *base=NoParam*)

Get a unique hash depending on the state of the data.

Parameters

- **data** (*object*) – any sort of loosely organized data
- **hasher** (*HASH*) – hash algorithm from hashlib, defaults to *sha512*.
- **hashlen** (*int*) – maximum number of symbols in the returned hash. If not specified, all are returned.
- **base** (*list*) – list of symbols or shorthand key. Defaults to base 26

Returns text - hash string

Return type str

1.1.2.12.1 Example

```
>>> print(hash_data([1, 2, (3, '4')], hashlen=8, hasher='sha512'))
iugjngof
```

frqkjbsq

ubelt.util_hash.**hash_file**(*fpath*, *blocksize=65536*, *stride=1*, *hasher=NoParam*, *hashlen=NoParam*, *base=NoParam*)

Hashes the data in a file on disk.

Parameters

- **fpath** (*str*) – file path string
- **blocksize** (*int*) – 2 ** 16. Affects speed of reading file
- **stride** (*int*) – strides > 1 skip data to hash, useful for faster hashing, but less accurate, also makes hash dependant on blocksize.
- **hasher** (*HASH*) – hash algorithm from hashlib, defaults to *sha512*.
- **hashlen** (*int*) – maximum number of symbols in the returned hash. If not specified, all are returned.
- **base** (*list*) – list of symbols or shorthand key. Defaults to base 26

1.1.2.12.2 Notes

For better hashes keep stride = 1 For faster hashes set stride > 1 blocksize matters when stride > 1

1.1.2.12.3 References

http://stackoverflow.com/questions/3431825/md5-checksum-of-a-file http://stackoverflow.com/questions/5001893/when-to-use-sha-1-vs-sha-2

1.1.2.12.4 Example

```
>>> import ubelt as ub
>>> from os.path import join
>>> fpath = join(ub.ensure_app_cache_dir('ubelt'), 'tmp.txt')
>>> ub.writeto(fpath, 'foobar')
>>> print(ub.hash_file(fpath, hasher='sha512', hashlen=8))
vkiodmcj
```

1.1.2.13 ubelt.util_import module

ubelt.util_import.**split_modpath**(*modpath*)

Splits the modpath into the dir that must be in PYTHONPATH for the module to be imported and the modulepath relative to this directory.

Parameters **modpath** (*str*) – module filepath

Returns (directory, rel_modpath)

Return type tuple

1.1.2.13.1 Example

```
>>> from xdoctest import static_analysis
>>> from os.path import join
>>> modpath = static_analysis.__file__
>>> modpath = modpath.replace('.pyc', '.py')
>>> dpath, rel_modpath = split_modpath(modpath)
```

```
>>> assert join(dpath, rel_modpath) == modpath
>>> assert rel_modpath == join('xdoctest', 'static_analysis.py')
```

ubelt.util_import.**modpath_to_modname**(*modpath*, *hide_init=True*, *hide_main=False*)

Determines importable name from file path

Converts the path to a module (__file__) to the importable python name (__name__) without importing the module.

The filename is converted to a module name, and parent directories are recursively included until a directory without an __init__.py file is encountered.

Parameters

- **modpath** (*str*) – module filepath
- **hide_init** (*bool*) – removes the __init__ suffix (default True)
- **hide_main** (*bool*) – removes the __main__ suffix (default False)

Returns modname

Return type str

1.1.2.13.2 Example

```
>>> import ubelt.util_import
>>> modpath = ubelt.util_import.__file__
>>> print(modpath_to_modname(modpath))
ubelt.util_import
```

ubelt.util_import.**modname_to_modpath**(*modname*, *hide_init=True*, *hide_main=True*, *sys_path=None*)

Finds the path to a python module from its name.

Determines the path to a python module without directly import it

Converts the name of a module (__name__) to the path (__file__) where it is located without importing the module. Returns None if the module does not exist.

Parameters

- **modname** (*str*) – module filepath
- **hide_init** (*bool*) – if False, __init__.py will be returned for packages
- **hide_main** (*bool*) – if False, and hide_init is True, __main__.py will be returned for packages, if it exists.
- **sys_path** (*list*) – if specified overrides *sys.path* (default None)

Returns modpath - path to the module, or None if it doesn't exist

Return type str

CommandLine: python -m ubelt.util_import modname_to_modpath

1.1.2.13.3 Example

```
>>> from ubelt.util_import import *  # NOQA
>>> import sys
>>> modname = 'ubelt.progiter'
>>> already_exists = modname in sys.modules
>>> modpath = modname_to_modpath(modname)
>>> print('modpath = {!r}'.format(modpath))
>>> assert already_exists or modname not in sys.modules
```

1.1.2.13.4 Example

```
>>> from ubelt.util_import import *  # NOQA
>>> import sys
>>> modname = 'ubelt.__main__'
>>> modpath = modname_to_modpath(modname, hide_main=False)
>>> print('modpath = {!r}'.format(modpath))
>>> assert modpath.endswith('__main__.py')
>>> modname = 'ubelt'
>>> modpath = modname_to_modpath(modname, hide_init=False)
>>> print('modpath = {!r}'.format(modpath))
>>> assert modpath.endswith('__init__.py')
>>> modname = 'ubelt'
>>> modpath = modname_to_modpath(modname, hide_init=False, hide_main=False)
>>> print('modpath = {!r}'.format(modpath))
>>> assert modpath.endswith('__init__.py')
```

ubelt.util_import.**import_module_from_name**(*modname*)

Imports a module from its string name (__name__)

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

Returns module

Return type module

1.1.2.13.5 Example

```
>>> # test with modules that wont be imported in normal circumstances
>>> # todo write a test where we gaurentee this
>>> modname_list = [
>>>     #'test',
>>>     'pickletools',
>>>     'lib2to3.fixes.fix_apply',
>>> ]
>>> #assert not any(m in sys.modules for m in modname_list)
>>> modules = [import_module_from_name(modname) for modname in modname_list]
>>> assert [m.__name__ for m in modules] == modname_list
>>> assert all(m in sys.modules for m in modname_list)
```

ubelt.util_import.**import_module_from_path**(*modpath*)

Imports a module via its path

Parameters **modpath** (*str*) – path to the module

Returns the imported module

Return type module

1.1.2.13.6 References

https://stackoverflow.com/questions/67631/import-module-given-path

1.1.2.13.7 Notes

If the module is part of a package, the package will be imported first. These modules may cause problems when reloading via IPython magic

Warning: It is best to use this with paths that will not conflict with previously existing modules.

If the modpath conflicts with a previously existing module name. And the target module does imports of its own relative to this conflicting path. In this case, the module that was loaded first will win.

For example if you try to import '/foo/bar/pkg/mod.py' from the folder structure:

- foo/ +- bar/

  +- pkg/

  – __init__.py

  |- mod.py |- helper.py

If there exists another module named *pkg* already in sys.modules and mod.py does something like *from . import helper*, Python will assume helper belongs to the *pkg* module already in sys.modules. This can cause a NameError or worse — a incorrect helper module.

Todo: handle modules inside of zipfiles

1.1.2.13.8 Example

```
>>> from ubelt import util_import
>>> modpath = util_import.__file__
>>> module = import_module_from_path(modpath)
>>> assert module is util_import
```

1.1.2.14 ubelt.util_io module

Functions for reading and writing files on disk.

writeto and *readfrom* wrap *open().write()* and *open().read()* and primarilly serve to indicate that the type of data being written and read is unicode text.

delete wraps *os.unlink* and *shutil.rmtree* and does not throw an error if the file or directory does not exist.

`ubelt.util_io.`**`writeto`**(*fpath*, *to_write*, *aslines=False*, *verbose=None*)

Writes (utf8) text to a file.

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

CommandLine: python -m ubelt.util_io writeto –verbose

1.1.2.14.1 Example

```
>>> from ubelt.util_io import *  # NOQA
>>> import ubelt as ub
>>> dpath = ub.ensure_app_cache_dir('ubelt')
>>> fpath = dpath + '/' + 'testwrite.txt'
>>> if exists(fpath):
>>>     os.remove(fpath)
>>> to_write = 'utf-8 symbols Δ, , , , , , , , and .'
>>> writeto(fpath, to_write)
>>> read_ = ub.readfrom(fpath)
>>> print('read_    = ' + read_)
>>> print('to_write = ' + to_write)
>>> assert read_ == to_write
```

1.1.2.14.2 Example

```
>>> from ubelt.util_io import *  # NOQA
>>> import ubelt as ub
>>> dpath = ub.ensure_app_cache_dir('ubelt')
>>> fpath = dpath + '/' + 'testwrite2.txt'
>>> if exists(fpath):
>>>     os.remove(fpath)
>>> to_write = ['a\n', 'b\n', 'c\n', 'd\n']
>>> writeto(fpath, to_write, aslines=True)
>>> read_ = ub.readfrom(fpath, aslines=True)
>>> print('read_    = {}'.format(read_))
>>> print('to_write = {}'.format(to_write))
>>> assert read_ == to_write
```

ubelt.util_io.**readfrom**(*fpath*, *aslines=False*, *errors='replace'*, *verbose=None*)

Reads (utf8) text from a file.

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

ubelt.util_io.**touch**(*fpath*, *mode=438*, *dir_fd=None*, *verbose=0*, ***kwargs*)
change file timestamps

Works like the touch unix utility

Parameters

- **fpath** (*str*) – name of the file
- **mode** (*int*) – file permissions (python3 and unix only)
- **dir_fd** (*file*) – optional directory file descriptor. If specified, fpath is interpreted as relative to this descriptor (python 3 only).
- **verbose** (*int*) – verbosity
- ****kwargs** – extra args passed to *os.utime* (python 3 only).

1.1.2.14.3 References

https://stackoverflow.com/questions/1158076/implement-touch-using-python

1.1.2.14.4 Example

```
>>> from ubelt.util_io import *  # NOQA
>>> import ubelt as ub
>>> dpath = ub.ensure_app_cache_dir('ubelt')
>>> fpath = join(dpath, 'touch_file')
>>> assert not exists(fpath)
>>> ub.touch(fpath)
>>> assert exists(fpath)
>>> os.unlink(fpath)
```

ubelt.util_io.**delete**(*path*, *verbose=False*)
Removes a file or recursively removes a directory. If a path does not exist, then this is a noop

Parameters

- **path** (*str*) – file or directory to remove
- **verbose** (*bool*) – if True prints what is being done

Doctest:

```
>>> import ubelt as ub
>>> from os.path import join, exists
>>> base = ub.ensure_app_cache_dir('ubelt', 'delete_test')
>>> dpath1 = ub.ensuredir(join(base, 'dir'))
>>> ub.ensuredir(join(base, 'dir', 'subdir'))
>>> ub.touch(join(base, 'dir', 'to_remove1.txt'))
>>> fpath1 = join(base, 'dir', 'subdir', 'to_remove3.txt')
>>> fpath2 = join(base, 'dir', 'subdir', 'to_remove2.txt')
>>> ub.touch(fpath1)
>>> ub.touch(fpath2)
>>> assert all(map(exists, (dpath1, fpath1, fpath2)))
>>> ub.delete(fpath1)
>>> assert all(map(exists, (dpath1, fpath2)))
>>> assert not exists(fpath1)
```

```
>>> ub.delete(dpath1)
>>> assert not any(map(exists, (dpath1, fpath1, fpath2)))
```

1.1.2.15 ubelt.util_links module

Logic for dealing with symlinks.

Notice that there is an aditional helper file for everyone's favorite pathological OS.

ubelt.util_links.**symlink**(*real_path*, *link_path*, *overwrite=False*, *verbose=0*)

Create a symbolic link.

This will work on linux or windows, however windows does have some corner cases. For more details see notes in *ubelt._win32_links*.

Parameters

- **path** (*str*) – path to real file or directory
- **link_path** (*str*) – path to desired location for symlink
- **overwrite** (*bool*) – overwrite existing symlinks. This will not overwrite real files on systems with proper symlinks. However, on older versions of windows junctions are indistinguishable from real files, so we cannot make this guarantee. (default = False)
- **verbose** (*int*) – verbosity level (default=0)

Returns link path

Return type str

CommandLine: python -m ubelt.util_links symlink:0

1.1.2.15.1 Example

```
>>> import ubelt as ub
>>> dpath = ub.ensure_app_cache_dir('ubelt', 'test_symlink0')
>>> real_path = join(dpath, 'real_file.txt')
>>> link_path = join(dpath, 'link_file.txt')
>>> [ub.delete(p) for p in [real_path, link_path]]
>>> ub.writeto(real_path, 'foo')
>>> result = symlink(real_path, link_path)
>>> assert ub.readfrom(result) == 'foo'
>>> [ub.delete(p) for p in [real_path, link_path]]
```

1.1.2.15.2 Example

```
>>> import ubelt as ub
>>> from os.path import dirname
>>> dpath = ub.ensure_app_cache_dir('ubelt', 'test_symlink1')
>>> ub.delete(dpath)
>>> ub.ensuredir(dpath)
>>> _dirstats(dpath)
>>> real_dpath = ub.ensuredir((dpath, 'real_dpath'))
>>> link_dpath = ub.augpath(real_dpath, base='link_dpath')
```

```
>>> real_path = join(dpath, 'afile.txt')
>>> link_path = join(dpath, 'afile.txt')
>>> [ub.delete(p) for p in [real_path, link_path]]
>>> ub.writeto(real_path, 'foo')
>>> result = symlink(real_dpath, link_dpath)
>>> assert ub.readfrom(link_path) == 'foo', 'read should be same'
>>> ub.writeto(link_path, 'bar')
>>> _dirstats(dpath)
>>> assert ub.readfrom(link_path) == 'bar', 'very bad bar'
>>> assert ub.readfrom(real_path) == 'bar', 'changing link did not change real'
>>> ub.writeto(real_path, 'baz')
>>> _dirstats(dpath)
>>> assert ub.readfrom(real_path) == 'baz', 'very bad baz'
>>> assert ub.readfrom(link_path) == 'baz', 'changing real did not change link'
>>> ub.delete(link_dpath, verbose=1)
>>> _dirstats(dpath)
>>> assert not exists(link_dpath), 'link should not exist'
>>> assert exists(real_path), 'real path should exist'
>>> _dirstats(dpath)
>>> ub.delete(dpath, verbose=1)
>>> _dirstats(dpath)
>>> assert not exists(real_path)
```

1.1.2.16 ubelt.util_list module

class ubelt.util_list.**chunks**(*sequence*, *chunksize=None*, *nchunks=None*, *total=None*, *bordermode='none'*)

Bases: object

Generates successive n-sized chunks from *sequence*. If the last chunk has less than n elements, *bordermode* is used to determine fill values.

Parameters

- **sequence** (*list*) – input to iterate over
- **chunksize** (*int*) – size of each sublist yielded
- **nchunks** (*int*) – number of chunks to create (cannot be specified with chunksize)
- **bordermode** (*str*) – determines how to handle the last case if the length of the sequence is not divisible by chunksize valid values are: {'none', 'cycle', 'replicate'}
- **total** (*int*) – hints about the length of the sequence

Todo: should this handle the case when sequence is a string?

1.1.2.16.1 References

http://stackoverflow.com/questions/434287/iterate-over-a-list-in-chunks

CommandLine: python -m ubelt.util_list chunks

1.1.2.16.2 Example

```
>>> import ubelt as ub
>>> sequence = [1, 2, 3, 4, 5, 6, 7]
>>> genresult = ub.chunks(sequence, chunksize=3, bordermode='none')
>>> assert list(genresult) == [[1, 2, 3], [4, 5, 6], [7]]
>>> genresult = ub.chunks(sequence, chunksize=3, bordermode='cycle')
>>> assert list(genresult) == [[1, 2, 3], [4, 5, 6], [7, 1, 2]]
>>> genresult = ub.chunks(sequence, chunksize=3, bordermode='replicate')
>>> assert list(genresult) == [[1, 2, 3], [4, 5, 6], [7, 7, 7]]
```

Doctest:

```
>>> import ubelt as ub
>>> assert len(list(ub.chunks(range(2), nchunks=2))) == 2
>>> assert len(list(ub.chunks(range(3), nchunks=2))) == 2
>>> assert len(list(ub.chunks([], 2, None, 'none'))) == 0
>>> assert len(list(ub.chunks([], 2, None, 'cycle'))) == 0
>>> assert len(list(ub.chunks([], 2, None, 'replicate'))) == 0
```

Doctest:

```
>>> def _check_len(self):
...     assert len(self) == len(list(self))
>>> _check_len(chunks(list(range(3)), nchunks=2))
>>> _check_len(chunks(list(range(2)), nchunks=2))
>>> _check_len(chunks(list(range(2)), nchunks=3))
```

Doctest:

```
>>> import pytest
>>> assert pytest.raises(ValueError, chunks, range(9))
>>> assert pytest.raises(ValueError, chunks, range(9), chunksize=2, nchunks=2)
>>> assert pytest.raises(TypeError, len, chunks((_ for _ in range(2)), 2))
```

static noborder(*sequence*, *chunksize*)

static cycle(*sequence*, *chunksize*)

static replicate(*sequence*, *chunksize*)

ubelt.util_list.**iterable**(*obj*, *strok=False*)

Checks if the input implements the iterator interface. An exception is made for strings, which return False unless *strok* is True

Parameters

- **obj** (*object*) – a scalar or iterable input
- **strok** (*bool*) – if True allow strings to be interpreted as iterable

Returns True if the input is iterable

Return type bool

1.1.2.16.3 Example

```
>>> obj_list = [3, [3], '3', (3,), [3, 4, 5], {}]
>>> result = [iterable(obj) for obj in obj_list]
>>> assert result == [False, True, False, True, True, True]
>>> result = [iterable(obj, strok=True) for obj in obj_list]
>>> assert result == [False, True, True, True, True, True]
```

ubelt.util_list.**take**(*items*, *indices*)

Selects a subset of a list based on a list of indices. This is similar to np.take, but pure python.

Parameters

- **items** (*list*) – an indexable object to select items from
- **indices** (*Sequence*) – sequence of indexing objects

Returns subset of the list

Return type iter or scalar

SeeAlso: ub.dict_subset

1.1.2.16.4 Example

```
>>> import ubelt as ub
>>> items = [0, 1, 2, 3]
>>> indices = [2, 0]
>>> list(ub.take(items, indices))
[2, 0]
```

ubelt.util_list.**compress**(*items*, *flags*)

Selects items where the corresponding value in flags is True This is similar to np.compress and it.compress

Parameters

- **items** (*Sequence*) – a sequence to select items from
- **flags** (*Sequence*) – corresponding sequence of bools

Returns a subset of masked items

Return type list

1.1.2.16.5 Example

```
>>> import ubelt as ub
>>> items = [1, 2, 3, 4, 5]
>>> flags = [False, True, True, False, True]
>>> list(ub.compress(items, flags))
[2, 3, 5]
```

ubelt.util_list.**flatten**(*nested_list*)

Parameters **nested_list** (*list*) – list of lists

Returns flat list

Return type list

1.1.2.16.6 Example

```
>>> import ubelt as ub
>>> nested_list = [['a', 'b'], ['c', 'd']]
>>> list(ub.flatten(nested_list))
['a', 'b', 'c', 'd']
```

ubelt.util_list.**unique**(*items*, *key=None*)

Generates unique items in the order they appear.

Parameters

- **items** (*Sequence*) – list of items
- **key** (*Function, optional*) – custom normalization function. If specified returns items where *key(item)* is unique.

Yields *object* – a unique item from the input sequence

CommandLine: python -m utool.util_list –exec-unique_ordered

1.1.2.16.7 Example

```
>>> import ubelt as ub
>>> items = [4, 6, 6, 0, 6, 1, 0, 2, 2, 1]
>>> unique_items = list(ub.unique(items))
>>> assert unique_items == [4, 6, 0, 1, 2]
```

1.1.2.16.8 Example

```
>>> import ubelt as ub
>>> items = ['A', 'a', 'b', 'B', 'C', 'c', 'D', 'e', 'D', 'E']
>>> unique_items = list(ub.unique(items, key=six.text_type.lower))
>>> assert unique_items == ['A', 'b', 'C', 'D', 'e']
>>> unique_items = list(ub.unique(items))
>>> assert unique_items == ['A', 'a', 'b', 'B', 'C', 'c', 'D', 'e', 'E']
```

ubelt.util_list.**argunique**(*items*, *key=None*)

Returns indices corresponding to the first instance of each unique item.

Parameters

- **items** (*list*) – list of items
- **key** (*Function, optional*) – custom normalization function. If specified returns items where *key(item)* is unique.

Yields *int* – indices of the unique items

1.1.2.16.9 Example

```
>>> items = [0, 2, 5, 1, 1, 0, 2, 4]
>>> indices = list(argunique(items))
>>> assert indices == [0, 1, 2, 3, 7]
```

```
>>> indices = list(argunique(items, key=lambda x: x % 2 == 0))
>>> assert indices == [0, 2]
```

ubelt.util_list.**unique_flags**(*items*, *key=None*)

Returns a list of booleans corresponding to the first instance of each unique item.

Parameters

- **items** (*list*) – list of items
- **key** (*Function, optional*) – custom normalization function. If specified returns items where *key(item)* is unique.

Returns flags the items that are unique

Return type list of bools

1.1.2.16.10 Example

```
>>> import ubelt as ub
>>> items = [0, 2, 1, 1, 0, 9, 2]
>>> flags = unique_flags(items)
>>> assert flags == [True, True, True, False, False, True, False]
>>> flags = unique_flags(items, key=lambda x: x % 2 == 0)
>>> assert flags == [True, False, True, False, False, False, False]
```

ubelt.util_list.**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

1.1.2.16.11 Example

```
>>> import ubelt as ub
>>> indices = [0, 1, 4]
>>> mask = ub.boolmask(indices, maxval=6)
>>> assert mask == [True, True, False, False, True, False]
>>> mask = ub.boolmask(indices)
>>> assert mask == [True, True, False, False, True]
```

ubelt.util_list.**iter_window**(*iterable*, *size=2*, *step=1*, *wrap=False*)

Iterates through iterable with a window size. This is essentially a 1D sliding window.

Parameters

- **iterable** (*iter*) – an iterable sequence
- **size** (*int*) – sliding window size (default = 2)
- **step** (*int*) – sliding step size (default = 1)

- **wrap** (*bool*) – wraparound (default = False)

Returns returns windows in a sequence

Return type iter

1.1.2.16.12 Example

```
>>> 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)
>>> print('window_list = %r' % (window_list,))
window_list = [(1, 2, 3), (2, 3, 4), (3, 4, 5), (4, 5, 6), (5, 6, 1), (6, 1, 2)]
```

1.1.2.16.13 Example

```
>>> iterable = [1, 2, 3, 4, 5, 6]
>>> size, step, wrap = 3, 2, True
>>> window_iter = iter_window(iterable, size, step, wrap)
>>> window_list = list(window_iter)
>>> print('window_list = %r' % (window_list,))
window_list = [(1, 2, 3), (3, 4, 5), (5, 6, 1)]
```

1.1.2.16.14 Example

```
>>> iterable = [1, 2, 3, 4, 5, 6]
>>> size, step, wrap = 3, 2, False
>>> window_iter = iter_window(iterable, size, step, wrap)
>>> window_list = list(window_iter)
>>> print('window_list = %r' % (window_list,))
window_list = [(1, 2, 3), (3, 4, 5)]
```

1.1.2.16.15 Example

```
>>> iterable = []
>>> size, step, wrap = 3, 2, False
>>> window_iter = iter_window(iterable, size, step, wrap)
>>> window_list = list(window_iter)
>>> print('window_list = %r' % (window_list,))
window_list = []
```

ubelt.util_list.**allsame**(*iterable*, *eq=<built-in function eq>*)

Determine if all items in a sequence are the same

Parameters

- **iterable** (*iter*) – an iterable sequence
- **eq** (*Function, optional*) – function to determine equality (default: operator.eq)

1.1.2.16.16 Example

```
>>> allsame([1, 1, 1, 1])
True
>>> allsame([])
True
>>> allsame([0, 1])
False
>>> iterable = iter([0, 1, 1, 1])
>>> next(iterable)
>>> allsame(iterable)
True
>>> allsame(range(10))
False
>>> allsame(range(10), lambda a, b: True)
True
```

ubelt.util_list.**argsort**(*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
- **key** (*Function, optional*) – customizes the ordering of the indexable
- **reverse** (*bool, optional*) – if True returns in descending order

Returns indices: list of indices such that sorts the indexable

Return type list

1.1.2.16.17 Example

```
>>> import ubelt as ub
>>> # argsort works on dicts by returning keys
>>> dict_ = {'a': 3, 'b': 2, 'c': 100}
>>> indices = ub.argsort(dict_)
>>> assert list(ub.take(dict_, indices)) == sorted(dict_.values())
>>> # argsort works on lists by returning indices
>>> indexable = [100, 2, 432, 10]
>>> indices = ub.argsort(indexable)
>>> assert list(ub.take(indexable, indices)) == sorted(indexable)
>>> # Can use iterators, but be careful. It exhausts them.
>>> indexable = reversed(range(100))
>>> indices = ub.argsort(indexable)
>>> assert indices[0] == 99
>>> # Can use key just like sorted
>>> indexable = [[0, 1, 2], [3, 4], [5]]
>>> indices = ub.argsort(indexable, key=len)
>>> assert indices == [2, 1, 0]
>>> # Can use reverse just like sorted
>>> indexable = [0, 2, 1]
>>> indices = ub.argsort(indexable, reverse=True)
>>> assert indices == [1, 2, 0]
```

ubelt.util_list.**argmax**(*indexable*, *key=None*)
Returns index / key of the item with the largest value.

The current implementation is simply a convinience wrapper around *ub.argsort*, a more efficient version will be written in the future.

Parameters

- **indexable** (*list or dict*) – indexable to sort by
- **key** (*Function, optional*) – customizes the ordering of the indexable

1.1.2.16.18 Example

```
>>> assert argmax({'a': 3, 'b': 2, 'c': 100}) == 'c'
>>> assert argmax(['a', 'c', 'b', 'z', 'f']) == 3
>>> assert argmax([[0, 1], [2, 3, 4], [5]], key=len) == 1
>>> assert argmax({'a': 3, 'b': 2, 3: 100, 4: 4}) == 3
>>> #import pytest
>>> #with pytest.raises(TypeError):
>>> #    argmax({'a': 3, 'b': 2, 3: 100, 4: 'd'})
```

ubelt.util_list.**argmin**(*indexable*, *key=None*)
Returns index / key of the item with the smallest value.

The current implementation is simply a convinience wrapper around *ub.argsort*, a more efficient version will be written in the future.

Parameters

- **indexable** (*list or dict*) – indexable to sort by
- **key** (*Function, optional*) – customizes the ordering of the indexable

1.1.2.16.19 Example

```
>>> assert argmin({'a': 3, 'b': 2, 'c': 100}) == 'b'
>>> assert argmin(['a', 'c', 'b', 'z', 'f']) == 0
>>> assert argmin([[0, 1], [2, 3, 4], [5]], key=len) == 2
>>> assert argmin({'a': 3, 'b': 2, 3: 100, 4: 4}) == 'b'
>>> #import pytest
>>> #assert pytest.raises(TypeError):
>>> #    argmax({'a': 3, 'b': 2, 3: 100, 4: 'd'})
```

1.1.2.17 ubelt.util_memoize module

ubelt.util_memoize.**memoize**(*func*)
memoization decorator that respects args and kwargs

1.1.2.17.1 References

https://wiki.python.org/moin/PythonDecoratorLibrary#Memoize

Parameters **func** (*function*) – live python function

Returns memoized wrapper

Return type func

CommandLine: python -m ubelt.util_decor memoize

1.1.2.17.2 Example

```
>>> import ubelt as ub
>>> closure = {'a': 'b', 'c': 'd'}
>>> incr = [0]
>>> def foo(key):
>>>     value = closure[key]
>>>     incr[0] += 1
>>>     return value
>>> foo_memo = ub.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'
```

class ubelt.util_memoize.**memoize_method**(*func*)

Bases: object

memoization decorator for a method that respects args and kwargs

1.1.2.17.3 References

http://code.activestate.com/recipes/577452-a-memoize-decorator-for-instance-methods/

1.1.2.17.4 Example

```
>>> import ubelt as ub
>>> closure = {'a': 'b', 'c': 'd'}
>>> incr = [0]
>>> class Foo(object):
>>>     @memoize_method
>>>     def foo_memo(self, key):
>>>         value = closure[key]
>>>         incr[0] += 1
>>>         return value
>>>     def foo(self, key):
>>>         value = closure[key]
>>>         incr[0] += 1
>>>         return value
```

```
>>> self = Foo()
>>> assert self.foo('a') == 'b' and self.foo('c') == 'd'
>>> assert incr[0] == 2
>>> print('Call memoized version')
>>> assert self.foo_memo('a') == 'b' and self.foo_memo('c') == 'd'
>>> assert incr[0] == 4
>>> assert self.foo_memo('a') == 'b' and self.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 self.foo('a') == 0 and self.foo('c') == 1
>>> assert incr[0] == 6
>>> assert self.foo_memo('a') == 'b' and self.foo_memo('c') == 'd'
>>> print('Constructing a new object should get a new cache')
>>> self2 = Foo()
>>> self2.foo_memo('a')
>>> assert incr[0] == 7
>>> self2.foo_memo('a')
>>> assert incr[0] == 7
```

1.1.2.18 ubelt.util_mixins module

class ubelt.util_mixins.**NiceRepr**

Bases: object

Defines *__str__* and *__repr__* in terms of *__nice__* function Classes that inherit *NiceRepr* must define *__nice__*

1.1.2.18.1 Example

```
>>> import ubelt as ub
>>> class Foo(ub.NiceRepr):
...    pass
>>> class Bar(ub.NiceRepr):
...    def __nice__(self):
...        return 'info'
>>> foo = Foo()
>>> bar = Bar()
>>> assert str(bar) == '<Bar(info)>'
>>> assert repr(bar).startswith('<Bar(info) at ')
>>> import pytest
>>> with pytest.warns(None) as record:
>>>     assert 'object at' in str(foo)
>>>     assert 'object at' in repr(foo)
```

1.1.2.19 ubelt.util_path module

ubelt.util_path.**augpath**(*path*, *suffix=''*, *prefix=''*, *ext=None*, *base=None*)

Augments a path with a new basename, extension, prefix and/or suffix.

A prefix is inserted before the basename. A suffix is inserted between the basename and the extension. The basename and extension can be replaced with a new one.

Parameters

- **path** (*str*) – string representation of a path
- **suffix** (*str*) – placed in front of the basename
- **prefix** (*str*) – placed between the basename and trailing extension
- **ext** (*str*) – if specified, replaces the trailing extension
- **base** (*str*) – if specified, replaces the basename (without extension)

Returns newpath

Return type str

CommandLine: python -m ubelt.util_path augpath

1.1.2.19.1 Example

```
>>> import ubelt as ub
>>> path = 'foo.bar'
>>> suffix = '_suff'
>>> prefix = 'pref_'
>>> ext = '.baz'
>>> newpath = ub.augpath(path, suffix, prefix, ext=ext, base='bar')
>>> print('newpath = %s' % (newpath,))
newpath = pref_bar_suff.baz
```

1.1.2.19.2 Example

```
>>> augpath('foo.bar')
'foo.bar'
>>> augpath('foo.bar', ext='.BAZ')
'foo.BAZ'
>>> augpath('foo.bar', suffix='_')
'foo_.bar'
>>> augpath('foo.bar', prefix='_')
'_foo.bar'
>>> augpath('foo.bar', base='baz')
'baz.bar'
```

ubelt.util_path.**userhome**(*username=None*)

Returns the user's home directory. If *username* is None, this is the directory for the current user.

Parameters **username** (*str*) – name of a user on the system

Returns userhome_dpath: path to the home directory

Return type str

1.1.2.19.3 Example

```
>>> import getpass
>>> username = getpass.getuser()
>>> assert userhome() == expanduser('~')
>>> assert userhome(username) == expanduser('~')
```

ubelt.util_path.**compressuser**(*path*, *home='~'*)
Inverse of *os.path.expanduser*

Parameters

- **path** (*str*) – path in system file structure
- **home** (*str*) – symbol used to replace the home path. Defaults to '~', but you might want to use '$HOME' or '%USERPROFILE%' instead.

Returns path: shortened path replacing the home directory with a tilde

Return type str

1.1.2.19.4 Example

```
>>> path = expanduser('~')
>>> assert path != '~'
>>> assert compressuser(path) == '~'
>>> assert compressuser(path + '1') == path + '1'
>>> assert compressuser(path + '/1') == join('~', '1')
>>> assert compressuser(path + '/1', '$HOME') == join('$HOME', '1')
```

ubelt.util_path.**truepath**(*path*, *real=False*)
Normalizes a string representation of a path and does shell-like expansion.

Parameters

- **path** (*str*) – string representation of a path
- **real** (*bool*) – if True, all symbolic links are followed. (default: False)

Returns normalized path

Return type str

Note: This function is simlar to the composition of expanduser, expandvars, normpath, and (realpath if *real* else abspath). However, on windows backslashes are then replaced with forward slashes to offer a consistent unix-like experience across platforms.

On windows expanduser will expand environment variables formatted as %name%, whereas on unix, this will not occur.

CommandLine: python -m ubelt.util_path truepath

1.1.2.19.5 Example

```
>>> import ubelt as ub
>>> assert ub.truepath('~/foo') == join(ub.userhome(), 'foo')
>>> assert ub.truepath('~/foo') == ub.truepath('~/foo/bar/..')
>>> assert ub.truepath('~/foo', real=True) == ub.truepath('~/foo')
```

ubelt.util_path.**ensuredir**(*dpath*, *mode=1023*, *verbose=None*)
Ensures that directory will exist. Creates new dir with sticky bits by default

Parameters

- **dpath** (*str*) – dir to ensure. Can also be a tuple to send to join
- **mode** (*int*) – octal mode of directory (default 0o1777)
- **verbose** (*int*) – verbosity (default 0)

Returns path: the ensured directory

Return type str

1.1.2.19.6 Notes

This function is not threadsafe in Python2

1.1.2.19.7 Example

```
>>> from ubelt.util_platform import *  # NOQA
>>> import ubelt as ub
>>> cache_dpath = ub.ensure_app_cache_dir('ubelt')
>>> dpath = join(cache_dpath, 'ensuredir')
>>> if exists(dpath):
...     os.rmdir(dpath)
>>> assert not exists(dpath)
>>> ub.ensuredir(dpath)
>>> assert exists(dpath)
>>> os.rmdir(dpath)
```

class ubelt.util_path.**TempDir**

Bases: object

Context for creating and cleaning up temporary directories.

1.1.2.19.8 Example

```
>>> with TempDir() as self:
>>>     dpath = self.dpath
>>>     assert exists(dpath)
>>> assert not exists(dpath)
```

1.1.2.19.9 Example

```
>>> self = TempDir()
>>> dpath = self.ensure()
>>> assert exists(dpath)
>>> self.cleanup()
>>> assert not exists(dpath)
```

ensure()

cleanup()

start()

1.1.2.20 ubelt.util_platform module

ubelt.util_platform.**platform_resource_dir**()
Returns a directory which should be writable for any application This should be used for persistent configuration files.

Returns path to the resource dir used by the current operating system

Return type str

ubelt.util_platform.**platform_cache_dir**()
Returns a directory which should be writable for any application This should be used for temporary deletable data.

Returns path to the cache dir used by the current operating system

Return type str

ubelt.util_platform.**get_app_resource_dir**(*appname*, **args*)
Returns a writable directory for an application This should be used for persistent configuration files.

Parameters

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

Returns dpath: writable resource directory for this application

Return type str

SeeAlso: ensure_app_resource_dir

ubelt.util_platform.**ensure_app_resource_dir**(*appname*, **args*)
Calls *get_app_resource_dir* but ensures the directory exists.

SeeAlso: get_app_resource_dir

1.1.2.20.1 Example

```
>>> import ubelt as ub
>>> dpath = ub.ensure_app_resource_dir('ubelt')
>>> assert exists(dpath)
```

ubelt.util_platform.**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 for this application

Return type str

SeeAlso: ensure_app_cache_dir

ubelt.util_platform.**ensure_app_cache_dir**(*appname*, **args*)
Calls *get_app_cache_dir* but ensures the directory exists.

SeeAlso: get_app_cache_dir

1.1.2.20.2 Example

```
>>> import ubelt as ub
>>> dpath = ub.ensure_app_cache_dir('ubelt')
>>> assert exists(dpath)
```

ubelt.util_platform.**startfile**(*fpath*, *verbose=True*)

Uses default program defined by the system to open a file. This is done via *os.startfile* on windows, *open* on mac, and *xdg-open* on linux.

Parameters

- **fpath** (*str*) – a file to open using the program associated with the files extension type.
- **verbose** (*int*) – verbosity

1.1.2.20.3 References

http://stackoverflow.com/questions/2692873/quote-posix

DisableExample:

```
>>> # This test interacts with a GUI frontend, not sure how to test.
>>> import ubelt as ub
>>> base = ub.ensure_app_cache_dir('ubelt')
>>> fpath1 = join(base, 'test_open.txt')
>>> ub.touch(fpath1)
>>> proc = ub.startfile(fpath1)
```

ubelt.util_platform.**editfile**(*fpath*, *verbose=True*)

Opens a file or code corresponding to a live python object in your preferred visual editor. This function is mainly useful in an interactive IPython session.

The visual editor is determined by the *VISUAL* environment variable. If this is not specified it defaults to gvim.

Parameters

- **fpath** (*str*) – a file path or python module / function
- **verbose** (*int*) – verbosity

DisableExample:

```
>>> # This test interacts with a GUI frontend, not sure how to test.
>>> import ubelt as ub
>>> ub.editfile(ub.util_platform.__file__)
>>> ub.editfile(ub)
>>> ub.editfile(ub.editfile)
```

1.1.2.21 ubelt.util_str module

class ubelt.util_str.**CaptureStdout**(*enabled=True*)

Bases: object

Context manager that captures stdout and stores it in an internal stream

Parameters **enabled** (*bool*) – (default = True)

CommandLine: python -m ubelt.util_str CaptureStdout

1.1.2.21.1 Notes

use version in xdoctest?

1.1.2.21.2 Example

```
>>> from ubelt.util_str import *  # NOQA
>>> self = CaptureStdout(enabled=True)
>>> print('dont capture the table flip (°° ')
>>> with self:
>>>     print('capture the heart ')
>>> print('dont capture look of disapproval _')
>>> assert isinstance(self.text, six.text_type)
>>> assert self.text == 'capture the heart \n', 'failed capture text'
```

ubelt.util_str.**indent**(*text*, *prefix=' '*)

Indents a block of text

Parameters

- **text** (*str*) – text to indent
- **prefix** (*str*) – prefix to add to each line (default = ‘ ‘)

Returns indented text

Return type str

CommandLine: python -m util_str indent

1.1.2.21.3 Example

```
>>> from ubelt.util_str import *  # NOQA
>>> text = 'Lorem ipsum\ndolor sit amet'
>>> prefix = '    '
>>> result = indent(text, prefix)
>>> assert all(t.startswith(prefix) for t in result.split('\n'))
```

ubelt.util_str.**codeblock**(*block_str*)

Wraps multiline string blocks and returns unindented code. Useful for templated code defined in indented parts of code.

Parameters **block_str** (*str*) – typically in the form of a multiline string

Returns the unindented string

Return type str

CommandLine: python -m ubelt.util_str codeblock

1.1.2.21.4 Example

```
>>> from ubelt.util_str import *  # NOQA
>>> # Simulate an indented part of code
>>> if True:
>>>     # notice the indentation on this will be normal
>>>     codeblock_version = codeblock(
...             '''
...             def foo():
...                 return 'bar'
...             '''
...         )
>>>     # notice the indentation and newlines on this will be odd
>>>     normal_version = ('''
...         def foo():
...             return 'bar'
...     ''')
>>> assert normal_version != codeblock_version
>>> print('Without codeblock')
>>> print(normal_version)
>>> print('With codeblock')
>>> print(codeblock_version)
```

ubelt.util_str.**hzcat**(*args*, *sep=''*)

Horizontally concatenates strings 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*) – strings to concat
- **sep** (*str*) – separator (defaults to '')

CommandLine: python -m ubelt.util_str hzcat

Example1:

```
>>> import ubelt as ub
>>> B = ub.repr2([[1, 2], [3, 457]], nl=1, cbr=True, trailsep=False)
>>> C = ub.repr2([[5, 6], [7, 8]], nl=1, cbr=True, trailsep=False)
>>> args = ['A = ', B, ' * ', C]
>>> print(ub.hzcat(args))
A = [[1, 2],   * [[5, 6],
     [3, 457]]    [7, 8]]
```

Example2:

```
>>> from ubelt.util_str import *
>>> import ubelt as ub
>>> aa = unicodedata.normalize('NFD', 'á')  # a unicode char with len2
>>> B = ub.repr2([['θ', aa], [aa, aa, aa]], nl=1, si=True, cbr=True,␣
↪trailsep=False)
>>> C = ub.repr2([[5, 6], [7, 'θ']], nl=1, si=True, cbr=True, trailsep=False)
>>> args = ['A', '=', B, '*', C]
>>> print(ub.hzcat(args, sep=''))
A=[[θ, á],   *[[5, 6],
   [á, á, á]]  [7, θ]]
```

ubelt.util_str.**ensure_unicode**(*text*)
Casts bytes into utf8 (mostly for python2 compatibility)

1.1.2.21.5 References

http://stackoverflow.com/questions/12561063/python-extract-data-from-file

1.1.2.21.6 Example

```
>>> from ubelt.util_str import *
>>> assert ensure_unicode('my ünicôdé strıng') == 'my ünicôdé strıng'
>>> assert ensure_unicode('text1') == 'text1'
>>> assert ensure_unicode('text1'.encode('utf8')) == 'text1'
>>> assert ensure_unicode('ï»¿text1'.encode('utf8')) == 'ï»¿text1'
>>> assert (codecs.BOM_UTF8 + 'text»¿'.encode('utf8')).decode('utf8')
```

1.1.2.22 ubelt.util_time module

class ubelt.util_time.**Timer**(*label=''*, *verbose=None*, *newline=True*)
Bases: object

Measures time elapsed between a start and end point. Can be used as a with-statement context manager, or using the tic/toc api.

Parameters

- **label** (*str*) – identifier for printing defaults to ‘’
- **verbose** (*int*) – verbosity flag, defaults to True if label is given
- **newline** (*bool*) – if False and verbose, print tic and toc on the same line

Variables

- **elapsed** (*float*) – number of seconds measured by the context manager
- **tstart** (*float*) – time of last *tic* reported by *default_timer()*

1.1.2.22.1 Example

```
>>> # Create and start the timer using the the context manager
>>> timer = Timer('Timer test!', verbose=1)
>>> with timer:
>>>     math.factorial(10000)
>>> assert timer.elapsed > 0
```

1.1.2.22.2 Example

```
>>> # Create and start the timer using the tic/toc interface
>>> timer = Timer().tic()
>>> elapsed1 = timer.toc()
>>> elapsed2 = timer.toc()
```

```
>>> elapsed3 = timer.toc()
>>> assert elapsed1 <= elapsed2
>>> assert elapsed2 <= elapsed3
```

tic()
starts the timer

toc()
stops the timer

class ubelt.util_time.**Timerit**(*num*, *label=None*, *bestof=3*, *verbose=None*)
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
- **bestof** (*int*) – takes the max over this number of trials
- **verbose** (*int*) – verbosity flag, defaults to True if label is given

CommandLine: python -m utool.util_time Timerit python -m utool.util_time Timerit:0 python -m utool.util_time Timerit:1

1.1.2.22.3 Example

```
>>> num = 15
>>> t1 = Timerit(num, verbose=2)
>>> for timer in t1:
>>>     # <write untimed setup code here> this example has no setup
>>>     with timer:
>>>         # <write code to time here> for example...
>>>         math.factorial(10000)
>>> # <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
```

1.1.2.22.4 Example

```
>>> num = 10
>>> # If the timer object is unused, time will still be recorded,
>>> # but with less precision.
>>> for _ in Timerit(num, 'imprecise'):
>>>     math.factorial(10000)
>>> # Using the timer object results in the most precise timings
>>> for timer in Timerit(num, 'precise'):
>>>     with timer: math.factorial(10000)
```

call (*func*, **args*, ***kwargs*)
Alternative way to time a simple function call using condensed syntax.

Returns Use *ave_secs*, *min*, or *mean* to get a scalar.

Return type self (*Timerit*)

1.1.2.22.5 Example

```
>>> ave_sec = Timerit(num=10).call(math.factorial, 50).ave_secs
>>> assert ave_sec > 0
```

ave_secs
The expected execution time of the timed code snippet in seconds. This is the minimum value recorded over all runs.

SeeAlso: self.min self.mean self.std

min()
The best time overall.

This is typically the best metric to consider when evaluating the execution time of a function. To understand why consider this quote from the docs of the original timeit module:

‘‘’ In a typical case, the lowest value gives a lower bound for how fast your machine can run the given code snippet; higher values in the result vector are typically not caused by variability in Python’s speed, but by other processes interfering with your timing accuracy. So the min() of the result is probably the only number you should be interested in. ‘‘’‘

1.1.2.22.6 Example

```
>>> self = Timerit(num=10, verbose=0)
>>> self.call(math.factorial, 50)
>>> assert self.min() > 0
```

mean()
The mean of the best results of each trial.

Note: This is typically less informative than simply looking at the min

1.1.2.22.7 Example

```
>>> self = Timerit(num=10, verbose=0)
>>> self.call(math.factorial, 50)
>>> assert self.mean() > 0
```

std()
The standard deviation of the best results of each trial.

Note: As mentioned in the timeit source code, the standard deviation is not often useful. Typically the minimum value is most informative.

1.1.2.22.8 Example

```
>>> self = Timerit(num=10, verbose=1)
>>> self.call(math.factorial, 50)
>>> assert self.std() >= 0
```

ubelt.util_time.**timestamp**(*method='iso8601'*)
make an iso8601 timestamp

CommandLine: python -m ubelt.util_time timestamp

1.1.2.22.9 Example

```
>>> stamp = timestamp()
>>> print('stamp = {!r}'.format(stamp))
...-...-...T...
```

1.1.3 Module contents

CHAPTER 2

Indices and tables

- genindex
- modindex

Python Module Index

u

ubelt, 61
ubelt.meta, 7
ubelt.meta.docscrape_google, 3
ubelt.meta.dynamic_analysis, 6
ubelt.orderedset, 7
ubelt.progiter, 12
ubelt.util_arg, 15
ubelt.util_cache, 16
ubelt.util_cmd, 20
ubelt.util_colors, 21
ubelt.util_const, 22
ubelt.util_dict, 23
ubelt.util_download, 30
ubelt.util_format, 31
ubelt.util_func, 32
ubelt.util_hash, 33
ubelt.util_import, 34
ubelt.util_io, 37
ubelt.util_links, 40
ubelt.util_list, 41
ubelt.util_memoize, 48
ubelt.util_mixins, 50
ubelt.util_path, 50
ubelt.util_platform, 54
ubelt.util_str, 55
ubelt.util_time, 58

Index

A

add() (ubelt.orderedset.OrderedSet method), 8
allsame() (in module ubelt.util_list), 46
append() (ubelt.orderedset.OrderedSet method), 8
argflag() (in module ubelt.util_arg), 16
argmax() (in module ubelt.util_list), 47
argmin() (in module ubelt.util_list), 48
argsort() (in module ubelt.util_list), 47
argunique() (in module ubelt.util_list), 44
argval() (in module ubelt.util_arg), 15
augpath() (in module ubelt.util_path), 50
AutoDict (class in ubelt.util_dict), 23
AutoOrderedDict (class in ubelt.util_dict), 23
ave_secs (ubelt.util_time.Timerit attribute), 60

B

begin() (ubelt.progiter.ProgIter method), 14
boolmask() (in module ubelt.util_list), 45

C

Cacher (class in ubelt.util_cache), 16
call() (ubelt.util_time.Timerit method), 59
CaptureStdout (class in ubelt.util_str), 55
chunks (class in ubelt.util_list), 41
cleanup() (ubelt.util_path.TempDir method), 53
clear() (ubelt.util_cache.Cacher method), 18
cmd() (in module ubelt.util_cmd), 20
codeblock() (in module ubelt.util_str), 56
color_text() (in module ubelt.util_colors), 22
compress() (in module ubelt.util_list), 43
compressuser() (in module ubelt.util_path), 51
copy() (ubelt.orderedset.OrderedSet method), 10
cycle() (ubelt.util_list.chunks static method), 42

D

ddict (in module ubelt.util_dict), 23
delete() (in module ubelt.util_io), 39
dict_hist() (in module ubelt.util_dict), 25
dict_subset() (in module ubelt.util_dict), 26
dict_take() (in module ubelt.util_dict), 27
dict_union() (in module ubelt.util_dict), 27
difference() (ubelt.orderedset.OrderedSet method), 10
difference_update() (ubelt.orderedset.OrderedSet method), 11
discard() (ubelt.orderedset.OrderedSet method), 8
display_message() (ubelt.progiter.ProgIter method), 15
download() (in module ubelt.util_download), 30
dzip() (in module ubelt.util_dict), 24

E

editfile() (in module ubelt.util_platform), 55
end() (ubelt.progiter.ProgIter method), 14
ensure() (ubelt.util_cache.Cacher method), 19
ensure() (ubelt.util_path.TempDir method), 53
ensure_app_cache_dir() (in module ubelt.util_platform), 54
ensure_app_resource_dir() (in module ubelt.util_platform), 54
ensure_newline() (ubelt.progiter.ProgIter method), 14
ensure_unicode() (in module ubelt.util_str), 57
ensuredir() (in module ubelt.util_path), 52
existing_versions() (ubelt.util_cache.Cacher method), 18
exists() (ubelt.util_cache.Cacher method), 18
extend() (ubelt.orderedset.OrderedSet method), 9

F

find_duplicates() (in module ubelt.util_dict), 26
flatten() (in module ubelt.util_list), 43
format_message() (ubelt.progiter.ProgIter method), 14

G

get_app_cache_dir() (in module ubelt.util_platform), 54
get_app_resource_dir() (in module ubelt.util_platform), 54
get_fpath() (ubelt.util_cache.Cacher method), 17
get_parent_frame() (in module ubelt.meta.dynamic_analysis), 6
get_stack_frame() (in module ubelt.meta.dynamic_analysis), 6

grabdata() (in module ubelt.util_download), 30
group_items() (in module ubelt.util_dict), 24

H

hash_data() (in module ubelt.util_hash), 33
hash_file() (in module ubelt.util_hash), 33
highlight_code() (in module ubelt.util_colors), 21
hzcat() (in module ubelt.util_str), 57

I

identity() (in module ubelt.util_func), 32
import_module_from_name() (in module ubelt.util_import), 36
import_module_from_path() (in module ubelt.util_import), 36
indent() (in module ubelt.util_str), 56
index() (ubelt.orderedset.OrderedSet method), 10
inject_method() (in module ubelt.util_func), 32
intersection() (ubelt.orderedset.OrderedSet method), 9
intersection_update() (ubelt.orderedset.OrderedSet method), 11
invert_dict() (in module ubelt.util_dict), 29
isdisjoint() (ubelt.orderedset.OrderedSet method), 8
issubset() (ubelt.orderedset.OrderedSet method), 10
issuperset() (ubelt.orderedset.OrderedSet method), 10
iter_window() (in module ubelt.util_list), 45
iterable() (in module ubelt.util_list), 42

L

load() (ubelt.util_cache.Cacher method), 18

M

map_keys() (in module ubelt.util_dict), 28
map_vals() (in module ubelt.util_dict), 28
mean() (ubelt.util_time.Timerit method), 60
memoize() (in module ubelt.util_memoize), 48
memoize_method (class in ubelt.util_memoize), 49
min() (ubelt.util_time.Timerit method), 60
modname_to_modpath() (in module ubelt.util_import), 35
modpath_to_modname() (in module ubelt.util_import), 35

N

NiceRepr (class in ubelt.util_mixins), 50
noborder() (ubelt.util_list.chunks static method), 42

O

odict (in module ubelt.util_dict), 23
OrderedSet (class in ubelt.orderedset), 7
oset (in module ubelt.orderedset), 12

P

parse_google_argblock() (in module ubelt.meta.docscrape_google), 5
parse_google_args() (in module ubelt.meta.docscrape_google), 3
parse_google_retblock() (in module ubelt.meta.docscrape_google), 4
parse_google_returns() (in module ubelt.meta.docscrape_google), 3
platform_cache_dir() (in module ubelt.util_platform), 54
platform_resource_dir() (in module ubelt.util_platform), 54
pop() (ubelt.orderedset.OrderedSet method), 8
ProgIter (class in ubelt.progiter), 12

R

readfrom() (in module ubelt.util_io), 38
replicate() (ubelt.util_list.chunks static method), 42
repr2() (in module ubelt.util_format), 31

S

save() (ubelt.util_cache.Cacher method), 19
set_extra() (ubelt.progiter.ProgIter method), 13
split_google_docblocks() (in module ubelt.meta.docscrape_google), 6
split_modpath() (in module ubelt.util_import), 34
start() (ubelt.util_path.TempDir method), 53
startfile() (in module ubelt.util_platform), 55
std() (ubelt.util_time.Timerit method), 60
step() (ubelt.progiter.ProgIter method), 14
symlink() (in module ubelt.util_links), 40
symmetric_difference() (ubelt.orderedset.OrderedSet method), 11
symmetric_difference_update() (ubelt.orderedset.OrderedSet method), 11

T

take() (in module ubelt.util_list), 43
TempDir (class in ubelt.util_path), 53
tic() (ubelt.util_time.Timer method), 59
Timer (class in ubelt.util_time), 58
Timerit (class in ubelt.util_time), 59
timestamp() (in module ubelt.util_time), 61
to_dict() (ubelt.util_dict.AutoDict method), 23
toc() (ubelt.util_time.Timer method), 59
touch() (in module ubelt.util_io), 38
truepath() (in module ubelt.util_path), 52
tryload() (ubelt.util_cache.Cacher method), 18

U

ubelt (module), 61
ubelt.meta (module), 7
ubelt.meta.docscrape_google (module), 3

ubelt.meta.dynamic_analysis (module), 6
ubelt.orderedset (module), 7
ubelt.progiter (module), 12
ubelt.util_arg (module), 15
ubelt.util_cache (module), 16
ubelt.util_cmd (module), 20
ubelt.util_colors (module), 21
ubelt.util_const (module), 22
ubelt.util_dict (module), 23
ubelt.util_download (module), 30
ubelt.util_format (module), 31
ubelt.util_func (module), 32
ubelt.util_hash (module), 33
ubelt.util_import (module), 34
ubelt.util_io (module), 37
ubelt.util_links (module), 40
ubelt.util_list (module), 41
ubelt.util_memoize (module), 48
ubelt.util_mixins (module), 50
ubelt.util_path (module), 50
ubelt.util_platform (module), 54
ubelt.util_str (module), 55
ubelt.util_time (module), 58
union() (ubelt.orderedset.OrderedSet method), 9
unique() (in module ubelt.util_list), 44
unique_flags() (in module ubelt.util_list), 45
update() (ubelt.orderedset.OrderedSet method), 9
userhome() (in module ubelt.util_path), 51

V

VERBOSE (ubelt.util_cache.Cacher attribute), 17

W

writeto() (in module ubelt.util_io), 37