

---

# UBelt Documentation

*Release 0.7.1*

**Jon Crall**

**Mar 19, 2019**



<b>1</b>	<b>ubelt</b>	<b>3</b>
1.1	ubelt package . . . . .	3
1.1.1	Submodules . . . . .	3
1.1.1.1	ubelt.orderedset module . . . . .	3
1.1.1.2	ubelt.progiter module . . . . .	8
1.1.1.3	ubelt.util_arg module . . . . .	12
1.1.1.4	ubelt.util_cache module . . . . .	13
1.1.1.5	ubelt.util_cmd module . . . . .	19
1.1.1.6	ubelt.util_colors module . . . . .	21
1.1.1.7	ubelt.util_const module . . . . .	22
1.1.1.8	ubelt.util_dict module . . . . .	23
1.1.1.9	ubelt.util_download module . . . . .	31
1.1.1.10	ubelt.util_format module . . . . .	34
1.1.1.11	ubelt.util_func module . . . . .	37
1.1.1.12	ubelt.util_hash module . . . . .	37
1.1.1.13	ubelt.util_import module . . . . .	40
1.1.1.14	ubelt.util_io module . . . . .	44
1.1.1.15	ubelt.util_links module . . . . .	47
1.1.1.16	ubelt.util_list module . . . . .	49
1.1.1.17	ubelt.util_memoize module . . . . .	56
1.1.1.18	ubelt.util_mixins module . . . . .	60
1.1.1.19	ubelt.util_path module . . . . .	60
1.1.1.20	ubelt.util_platform module . . . . .	64
1.1.1.21	ubelt.util_str module . . . . .	70
1.1.1.22	ubelt.util_stream module . . . . .	72
1.1.1.23	ubelt.util_time module . . . . .	73
1.1.2	Module contents . . . . .	78
<b>2</b>	<b>Indices and tables</b>	<b>147</b>
	<b>Python Module Index</b>	<b>149</b>



UBelt is a “utility belt” of commonly needed utility and helper functions. It is a curated collection of utilities with functionality that falls into a mixture of the following categories:

- Timing
- Caching
- Hashing
- Command Line / Shell Interaction
- Cross-Platform Cache, Config, and Data Directories
- Symlinks
- Downloading Files
- Dictionary Histogram
- Find Duplicates
- Dictionary Manipulation
- AutoDict - Autovivification
- String-based imports
- Horizontal String Concatenation
- **Standalone modules.**
  - `progiter` for Loop Progress
  - `timerit` for Robust Timing and Benchmarking
  - `ordered-set` for ordered set collections

For more detailed high level documentation see the README on github: <https://github.com/Erotemic/ubelt>.

The ubelt API is organized by submodules containing related functionality. Each submodule contains top level overview documentation, and each function contains a docstring with at least one example. Please see specific submodule documentation for more details.

**AutogenInit:** `mkinit ubelt -w # todo: get sphinx to ignore this`



## 1.1 ubelt package

### 1.1.1 Submodules

#### 1.1.1.1 ubelt.orderedset module

This module exposes the *OrderedSet* class, which is a collection of unique items that maintains the order in which the items were added. An *OrderedSet* (or its alias *oset*) behaves very similarly to Python's builtin *set* object, the main difference being that an *OrderedSet* can efficiently lookup its items by index.

While *ubelt* exposes *OrderedSet*, the actual implementation lives in the standalone *ordered\_set* package. For more details on the implementation please reference <https://github.com/LuminosoInsight/ordered-set>.

#### Example

```
>>> import ubelt as ub
>>> ub.oset([1, 2, 3])
OrderedSet([1, 2, 3])
>>> (ub.oset([1, 2, 3]) - {2}) | {2}
OrderedSet([1, 3, 2])
>>> [ub.oset([1, 2, 3])[i] for i in [1, 0, 2]]
[2, 1, 3]
```

```
class ubelt.orderedset.OrderedSet (iterable=None)
    Bases: collections.abc.MutableSet, collections.abc.Sequence
```

An *OrderedSet* is a custom *MutableSet* that remembers its order, so that every entry has an index that can be looked up.

### Example

```
>>> OrderedSet([1, 1, 2, 3, 2])
OrderedSet([1, 2, 3])
```

#### **copy()**

Return a shallow copy of this object.

### Example

```
>>> this = OrderedSet([1, 2, 3])
>>> other = this.copy()
>>> this == other
True
>>> this is other
False
```

#### **add(key)**

Add *key* as an item to this *OrderedSet*, then return its index.

If *key* is already in the *OrderedSet*, return the index it already had.

### Example

```
>>> oset = OrderedSet()
>>> oset.append(3)
0
>>> print(oset)
OrderedSet([3])
```

#### **append(key)**

Add *key* as an item to this *OrderedSet*, then return its index.

If *key* is already in the *OrderedSet*, return the index it already had.

### Example

```
>>> oset = OrderedSet()
>>> oset.append(3)
0
>>> print(oset)
OrderedSet([3])
```

#### **update(sequence)**

Update the set with the given iterable sequence, then return the index of the last element inserted.

### Example

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

(continues on next page)



(continued from previous page)

```
>>> print(oiset)
OrderedSet([1, 2, 3, 5, 4])
```

**index** (*key*)

Get the index of a given entry, raising an `IndexError` if it's not present.

*key* can be an iterable of entries that is not a string, in which case this returns a list of indices.

**Example**

```
>>> oiset = OrderedSet([1, 2, 3])
>>> oiset.index(2)
1
```

**get\_loc** (*key*)

Get the index of a given entry, raising an `IndexError` if it's not present.

*key* can be an iterable of entries that is not a string, in which case this returns a list of indices.

**Example**

```
>>> oiset = OrderedSet([1, 2, 3])
>>> oiset.index(2)
1
```

**get\_indexer** (*key*)

Get the index of a given entry, raising an `IndexError` if it's not present.

*key* can be an iterable of entries that is not a string, in which case this returns a list of indices.

**Example**

```
>>> oiset = OrderedSet([1, 2, 3])
>>> oiset.index(2)
1
```

**pop** ()

Remove and return the last element from the set.

Raises `KeyError` if the set is empty.

**Example**

```
>>> oiset = OrderedSet([1, 2, 3])
>>> oiset.pop()
3
```

**discard** (*key*)

Remove an element. Do not raise an exception if absent.

The `MutableSet` mixin uses this to implement the `.remove()` method, which *does* raise an error when asked to remove a non-existent item.

### Example

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

#### **clear()**

Remove all items from this OrderedSet.

#### **union(\*sets)**

Combines all unique items. Each items order is defined by its first appearance.

### Example

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

### Example

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

#### **difference(\*sets)**

Returns all elements that are in this set but not the others.

### Example

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

**issubset** (*other*)

Report whether another set contains this set.

**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.

**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 OrderedSets as a new set. That is, the new set will contain all elements that are in exactly one of the sets.

Their order will be preserved, with elements from *self* preceding elements from *other*.

**Example**

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

**difference\_update** (*\*sets*)

Update this OrderedSet to remove items from one or more other sets.

**Example**

```
>>> this = OrderedSet([1, 2, 3])
>>> this.difference_update(OrderedSet([2, 4]))
>>> print(this)
OrderedSet([1, 3])
```

```
>>> this = OrderedSet([1, 2, 3, 4, 5])
>>> this.difference_update(OrderedSet([2, 4]), OrderedSet([1, 4, 6]))
>>> print(this)
OrderedSet([3, 5])
```

**intersection\_update** (*other*)

Update this OrderedSet to keep only items in another set, preserving their order in this set.

### Example

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

**symmetric\_difference\_update** (*other*)

Update this OrderedSet to remove items from another set, then add items from the other set that were not present in this set.

### Example

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

`ubelt.orderedset.aset`

alias of `ordered_set.OrderedSet`

### 1.1.1.2 ubelt.progiter module

ProgIter now exists in a standalone pip-installable module. The source code lives in its own github repo here: <https://github.com/Erotemic/progiter>

```
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,
                               chunksize=None, **kwargs)
```

Bases: `progiter.progiter._TQDMCompat`, `progiter.progiter._BackwardsCompat`

Prints progress as an iterator progresses

#### 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.
- **chunksize** (*int*) – indicates that each iteration processes a batch of this size. Iteration rate is displayed in terms of single-items.
- **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

---

**Note:** 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.

---

---

**Note:** ProgIter is an alternative to *tqdm*. 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 *ProgIter* API will change to become inter-compatible with *tqdm*.

---

**SeeAlso:** tqdm - <https://pypi.python.org/pypi/tqdm>

**Reference:** <http://datagenetics.com/blog/february12017/index.html>

## Example

```
>>> # doctest: +SKIP
>>> import progiter
>>> def is_prime(n):
...     return n >= 2 and not any(n % i == 0 for i in range(2, n))
>>> for n in progiter.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:

- [ ] extra is a bad name; come up with something better and rename
-

### Example

```
>>> import progiter
>>> prog = progiter.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 precedence over *inc*
- **inc** (*int*) – number of steps to increment (defaults to 1)

### Example

```
>>> import progiter
>>> n = 3
>>> prog = progiter.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()
```

### Example

```
>>> import progiter
>>> n = 3
>>> # can be used as a context manager in manual mode
>>> with progiter.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.

### Example

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

(continues on next page)

(continued from previous page)

```
>>> self.begin()
>>> self.step()
>>> print(repr(self.format_message()))
' 1/?... \n'
```

### Example

```
>>> self = ProgIter(chunksize=10, total=100, clearline=False,
>>>                  show_times=False, microseconds=True)
>>> # hack, microseconds=True for coverage, needs real test
>>> print(repr(self.format_message()))
' 0.00% of 10x100... \n'
>>> self.begin()
>>> self.update() # tqdm alternative to step
>>> print(repr(self.format_message()))
' 1.00% of 10x100... \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

### Example

```
>>> # Unsafe version may write your message on the wrong line
>>> import progiter
>>> prog = progiter.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 = progiter.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

### 1.1.1.3 ubelt.util\_arg module

Simple ways to interact with the commandline without defining a full blown CLI

`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** (*Optional[object]*) – value to return if not specified
- **argv** (*Optional[list]*) – overrides `sys.argv` if specified

#### Returns

**value** [the value specified after the key. If the key is specified multiple times, then the first value is returned.]

**Return type** `str`

---

#### Todo:

- [ ] Can we handle the case where the value is a list of long paths?
  - [ ] Should we default the first or last specified instance of the flag.
- 

#### Example

```
>>> 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
```

#### Example

```
>>> # Test fix for GH Issue #41
>>> import ubelt as ub
>>> argv = ['--path=/path/with/k=3']
>>> ub.argval('--path', argv=argv) == '/path/with/k=3'
```

`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** (*Optional[list]*) – overrides `sys.argv` if specified



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

**Return type** bool

### Example

```
>>> 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
```

#### 1.1.1.4 ubelt.util\_cache module

This module exposes *Cacher* and *CacheStamp* classes, which provide a simple API for on-disk caching.

The *Cacher* class is the simplest and most direct method of caching. In fact, it only requires four lines of boilerplate, which is the fewest number of lines which this author (Jon Crall) has ever been able to achieve while still being general and robust. These four lines achieve the following necessary and sufficient steps for general robust on-disk caching.

- Defining the cache dependencies
- Checking if the cache exists
- Loading the cache on a hit
- Executing the process and saving the result on a miss.

The following example illustrates these four points.

### Example

```
>>> import ubelt as ub
>>> # Defines a cache name and dependencies, note the use of `ub.hash_data`.
>>> cacher = ub.Cacher('name', cfgstr=ub.hash_data('dependencies')) #_
↪boilerplate:1
>>> # Calling tryload will return your data on a hit and None on a miss
>>> data = cacher.tryload() #_
↪boilerplate:2
>>> # Check if you need to recompute your data
>>> if data is None: #_
↪boilerplate:3
>>>     # Recompute your data (this is not boilerplate)
>>>     data = 'mydata'
>>>     # Cache the computation result (pickle is used by default)
>>>     cacher.save(data) #_
↪boilerplate:4
```

Surprisingly this uses just as many boilerplate lines as a decorator style cacher, but it is much more extensible. It is possible to use *ub.Cacher* in more sophisticated ways (e.g. metadata), but the simple in-line use is often easier and cleaner. The following example illustrates this:

## Example

```
>>> import ubelt as ub
>>> @ub.Cacher('name', cfgstr=ub.hash_data('dependencies')) # boilerplate:1
>>> def func(): # boilerplate:2
>>>     data = 'mydata'
>>>     return data # boilerplate:3
>>> data = func() # boilerplate:4
```

```
>>> cacher = ub.Cacher('name', cfgstr=ub.hash_data('dependencies')) # boilerplate:1
>>> data = cacher.tryload() # boilerplate:2
>>> if data is None: # boilerplate:3
>>>     data = 'mydata'
>>>     cacher.save(data) # boilerplate:4
```

While the above two are equivalent, the second version provides simpler tracebacks, explicit procedures, and makes it easier to use breakpoint debugging (because there is no closure scope).

While *Cacher* is used to store simple results of in-line code in a pickle format, the *CacheStamp* object is used to cache processes that produces an on-disk side effects other than the main return value. For instance, consider the following example:

## Example

```
>>> def compute_many_files(dpath):
...     for i in range(0):
...         fpath = '{}/file{}.txt'.format(dpath, i)
...         open(fpath).write('foo' + str(i))
>>> #
>>> import ubelt as ub
>>> dpath = ub.ensure_app_cache_dir('ubelt/demo/cache')
>>> # You must specify a directory, unlike in Cacher where it is optional
>>> self = ub.CacheStamp('name', dpath=dpath, cfgstr='dependencies')
>>> if self.expired():
>>>     compute_many_files(dpath)
>>>     # Instead of caching the whole processes, we just write a file
>>>     # that signals the process has been done.
>>>     self.renew()
>>> assert not self.expired()
```

```
class ubelt.util_cache.Cacher(fname, cfgstr=None, dpath=None, appname='ubelt', ext='.pkl',
                             meta=None, verbose=None, enabled=True, log=None,
                             hasher='sha1', 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 always 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** (*PathLike*) – 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*) – File extension (default = '.pkl')
- **meta** (*object*) – Metadata that is also saved with the *cfgstr*. This can be useful to indicate how the *cfgstr* was constructed.
- **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, ...)
- **hasher** (*str*) – Type of hashing algorithm to use if *cfgstr* needs to be condensed to less than 49 characters.
- **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

### 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
```

### 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')
```

(continues on next page)

(continued from previous page)

```
>>>    cacher.save(myvar)
>>> assert cacher.exists(), 'should now exist'
```

**VERBOSE** = 1

**FORCE\_DISABLE** = False

**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.

### 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.

### 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. Either raise, which re-raises the exception, or clear which deletes the cache and returns None.

**load** (*cfgstr=None*)

Loads the data

**Raises** *IOError* - if the data is unable to be loaded. This could be due to – a cache miss or because the cache is disabled.

### 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.

### 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

### Example

```
>>> from ubelt.util_cache import * # NOQA
>>> def func():
>>>     return 'expensive result'
```

(continues on next page)

(continued from previous page)

```
>>> 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()
```

```
class ubelt.util_cache.CacheStamp(fname, dpath, cfgstr=None, product=None, hasher='sha1',
                                verbose=None)
```

Bases: object

Quickly determine if a file-producing computation has been done.

Writes a file that marks that a procedure has been done by writing a “stamp” file to disk. Removing the stamp file will force recomputation. However, removing or changing the result of the computation may not trigger recomputation unless specific handling is done or the expected “product” of the computation is a file and registered with the stamper. If hasher is None, we only check if the product exists, and we ignore its hash, otherwise it checks that the hash of the product is the same.

#### Parameters

- **fname** (*str*) – Name of the stamp file
- **cfgstr** (*str*) – Configuration associated with the stamped computation. A common pattern is to call `ubelt.hash_data` on a dependency list.
- **dpath** (*PathLike*) – Where to store the cached stamp file
- **product** (*PathLike or Sequence[PathLike]*, *optional*) – Path or paths that we expect the computation to produce. If specified the hash of the paths are stored.
- **hasher** (*str*) – The type of hasher used to compute the file hash of product. If None, then we assume the file has not been corrupted or changed. Defaults to sha1.
- **verbose** (*bool*) – Passed to internal `ub.Cacher` object

#### Example

```
>>> import ubelt as ub
>>> from os.path import join
>>> # Stamp the computation of expensive-to-compute.txt
>>> dpath = ub.ensure_app_cache_dir('ubelt', 'test-cache-stamp')
>>> ub.delete(dpath)
>>> ub.ensuredir(dpath)
>>> product = join(dpath, 'expensive-to-compute.txt')
>>> self = CacheStamp('somedata', cfgstr='someconfig', dpath=dpath,
>>>                  product=product, hasher=None)
>>> self.hasher = None
>>> if self.expired():
>>>     ub.writeto(product, 'very expensive')
>>>     self.renew()
>>> assert not self.expired()
>>> # corrupting the output will not expire in non-robust mode
>>> ub.writeto(product, 'corrupted')
>>> assert not self.expired()
```

(continues on next page)

(continued from previous page)

```
>>> self.hasher = 'sha1'
>>> # but it will expire if we are in robust mode
>>> assert self.expired()
>>> # deleting the product will cause expiration in any mode
>>> self.hasher = None
>>> ub.delete(product)
>>> assert self.expired()
```

**expired** (cfgstr=None, product=None)

Check to see if a previously existing stamp is still valid and if the expected result of that computation still exists.

#### Parameters

- **cfgstr** (*str*, optional) – override the default cfgstr if specified
- **product** (*PathLike* or *Sequence[PathLike]*, optional) – override the default product if specified

**renew** (cfgstr=None, product=None)

Recertify that the product has been recomputed by writing a new certificate to disk.

### 1.1.1.5 ubelt.util\_cmd module

This module exposes the *ubelt.cmd* command, which provides a simple means for interacting with the commandline. While this does use *subprocess.Popen* under the hood, the key draw of *ubelt.cmd* is that you can capture stdout/stderr in your program while simultaneously printing it to the terminal in real time.

#### Example

```
>>> import ubelt as ub
>>> # Running with verbose=1 will write to stdout in real time
>>> info = ub.cmd('echo "write your command naturally"', verbose=1)
write your command naturally
>>> # Unless `detach=True`, `cmd` always returns an info dict.
>>> print('info = ' + ub.repr2(info))
info = {
  'command': 'echo "write your command naturally"',
  'cwd': None,
  'err': '',
  'out': 'write your command naturally\n',
  'proc': <subprocess.Popen object at ...>,
  'ret': 0,
}
```

**ubelt.util\_cmd.cmd** (command, shell=False, detach=False, verbose=0, tee=None, cwd=None, env=None, tee\_backend='auto', verbout=None, \*\*kwargs)

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 detach, return the process object and allow the process to run in the background (eventually we may return a Future object instead).

#### Parameters

- **command** (*str or Sequence*) – bash-like command string or tuple of executable and args
- **shell** (*bool*) – if True, process is run in shell, defaults to False.
- **detach** (*bool*) – if True, process is detached and run in background, defaults to False.
- **verbose** (*int*) – verbosity mode. Can be 0, 1, 2, or 3. Defaults to 0.
- **tee** (*bool, optional*) – if True, simultaneously writes to stdout while capturing output from the command. If not specified, defaults to True if verbose > 0. If detach is True, then this argument is ignored.
- **cwd** (*PathLike, optional*) – path to run command
- **env** (*str, optional*) – environment passed to Popen
- **tee\_backend** (*str, optional*) – backend for tee output. Valid choices are: “auto”, “select” (POSIX only), and “thread”.
- **\*\*kwargs** – only used to support deprecated arguments

### Returns

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

**Return type** dict

### 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)”`

### 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-distort>  
 [3] <https://stackoverflow.com/questions/33560364/python-windows-parsing-command-lines-with-shlex>

### 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' + chr(10) + 'shell'
```

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), detach=True)
>>> info2 = ub.cmd('echo writing2 > ' + fpath2, shell=True, detach=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()
```

### 1.1.1.6 ubelt.util\_colors module

This module exposes simple functions to color your text and highlight your code using ANSI escape sequences. This works using the `Pygments` library, which is an optional requirement. Therefore, these functions only work properly if `Pygments` is installed, otherwise these functions will return the unmodified text and a warning will be printed.

**Requirements:** pip install pygments

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

Highlights a block of text using ANSI 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()))"`

### 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, underline, purple, green, teal, fuchsia

#### 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`

### Example

```
>>> text = 'raw text'
>>> import pytest
>>> import ubelt as ub
>>> if ub.modname_to_modpath('pygments'):
>>>     # Colors text only if pygments is installed
>>>     assert color_text(text, 'red') == '\x1b[31;01mraw text\x1b[39;49;00m'
>>>     assert color_text(text, None) == 'raw text'
>>> else:
>>>     # Otherwise text passes through unchanged
>>>     assert color_text(text, 'red') == 'raw text'
>>>     assert color_text(text, None) == 'raw text'
```

#### 1.1.1.7 ubelt.util\_const module

This module defines `ub.NoParam`. This is a robust sentinel 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)

### Example

```

>>> import ubelt as ub
>>> def func(a=ub.NoParam):
>>>     if a is ub.NoParam:
>>>         print('no param specified')
>>>     else:
>>>         print('a = {}'.format(a))
>>> func()
no param specified
>>> func(a=None)
a = None
>>> func(a=1)
a = 1
>>> # note: typically it is bad practice to use NoParam as an actual
>>> # (non-default) parameter. It goes against the spirit of the idea.
>>> func(a=ub.NoParam)
no param specified

```

### 1.1.1.8 ubelt.util\_dict module

Functions for working with dictionaries.

`ubelt.util_dict.odict`  
alias of `collections.OrderedDict`

`ubelt.util_dict.ddict`  
alias of `collections.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

### References

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

### 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`

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

**SeeAlso:** `ub.AutoDict` - the unordered version

### Example

```
>>> 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*, *cls=<class 'dict'>*)

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

#### Parameters

- **items1** (*Iterable*) – full sequence
- **items2** (*Iterable*) – can either be a sequence of one item or a sequence of equal length to *items1*
- **cls** (*Type[dict]*) – dictionary type to use. Defaults to `dict`, but could be ordered dict instead.

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

**Return type** `dict`

### 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` (*items*, *groupids*)

Groups a list of items by group id.

#### Parameters

- **items** (*Iterable*) – a list of items to group
- **groupids** (*Iterable or Callable*) – a corresponding list of item groupids or a function mapping an item to a groupid.

**Returns** groupid\_to\_items: maps a groupid to a list of items

**Return type** dict

**CommandLine:** python -m ubelt.util\_dict group\_items

### Example

```
>>> import ubelt as ub
>>> items = ['ham', 'jam', 'spam', 'eggs', 'cheese', 'banana']
>>> groupids = ['protein', 'fruit', 'protein', 'protein', 'dairy', 'fruit']
>>> groupid_to_items = ub.group_items(items, groupids)
>>> 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, counting the number of time each item appears in the input.

#### Parameters

- **item\_list** (*Iterable*) – hashable items (usually containing duplicates)
- **weight\_list** (*Iterable*) – corresponding weights for each item
- **ordered** (*bool*) – if True the result is ordered by frequency
- **labels** (*Iterable, optional*) – expected labels (default None) Allows this function to pre-initialize the histogram. 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

### 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}
```

### 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=[])
```

(continues on next page)

(continued from previous page)

```
>>> 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, key=None)`

Find all duplicate items in a list.

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

#### Parameters

- **items** (*Iterable*) – hashable items possibly containing duplicates
- **k** (*int*) – only return items that appear at least *k* times (default=2)
- **key** (*Callable, optional*) – Returns indices where `key(items[i])` maps to a particular value at least *k* times.

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

**Return type** dict

**CommandLine:** `python -m ubelt.util_dict find_duplicates`

#### 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]}
```

#### 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]}
```

### Example

```
>>> import ubelt as ub
>>> items = [10, 11, 12, 13, 14, 15, 16]
>>> duplicates = ub.find_duplicates(items, key=lambda x: x // 2)
>>> print(ub.repr2(duplicates, nl=0))
{5: [0, 1], 6: [2, 3], 7: [4, 5]}
```

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

Get a subset of a dictionary

#### Parameters

- **dict\_** (*Mapping*) – superset dictionary
- **keys** (*Iterable*) – keys to take from *dict\_*
- **default** (*object, optional*) – if specified uses default if keys are missing

**Returns** subset dictionary

**Return type** OrderedDict

**SeeAlso:** `dict_isect` - similar functionality, but will only take existing keys

### 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\_** (*Mapping*) – a dictionary to take from
- **keys** (*Iterable*) – the keys to take
- **default** (*object, optional*) – if specified uses default if keys are missing

**CommandLine:** `python -m ubelt.util_dict dict_take_gen`

### 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]
```

### 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 precedence.

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

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

**Return type** `Dict | OrderedDict`

**SeeAlso:**

**`collections.ChainMap`** - a standard python builtin data structure that provides a view that treats multiple dicts as a single dict. <https://docs.python.org/3/library/collections.html#chainmap-objects>

### 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.dict_isect(*args)`

Constructs a dictionary that contains keys common between all inputs. The returned values will only belong to the first dictionary.

**Parameters** `*args` – a sequence of dictionaries (or sets of keys)

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

**Return type** `Dict | OrderedDict`

### Notes

This function can be used as an alternative to `dict_subset` where any key not in the dictionary is ignored. See the following example:

```
>>> dict_isect({'a': 1, 'b': 2, 'c': 3}, ['a', 'c', 'd'])
{'a': 1, 'c': 3}
```



### Example

```
>>> dict_isect({'a': 1, 'b': 1}, {'b': 2, 'c': 2})
{'b': 1}
>>> dict_isect(odict([('a', 1), ('b', 2)]), odict([('c', 3)]))
OrderedDict()
>>> dict_isect()
{}
```

`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`

### 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`

### 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

**Return type** dict

#### 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`

#### Example

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

#### Example

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

## Example

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

### 1.1.1.9 ubelt.util\_download module

Helpers for downloading data

`ubelt.util_download.download(url, fpath=None, hash_prefix=None, hasher='sha512', chunk-size=8192, verbose=1)`

downloads a url to a fpath.

#### Parameters

- **url** (*str*) – The url to download.
- **fpath** (*PathLike* | *io.BytesIO* | *trio.IO*) – The path to download to. Defaults to basename of url and ubelt's application cache. If this is a *io.BytesIO* object then information is directly written to this object (note this prevents the use of temporary files).
- **hash\_prefix** (*None* or *str*) – If specified, download will retry / error if the file hash does not match this value. Defaults to *None*.
- **hasher** (*str* or *Hasher*) – If *hash\_prefix* is specified, this indicates the hashing algorithm to apply to the file. Defaults to *sha512*.
- **chunksize** (*int*) – Download chunksize. Defaults to  $2^{13}$ .
- **verbose** (*int*) – Verbosity level 0 or 1. Defaults to 1.

**Returns** fpath - file path string

**Return type** PathLike

#### Raises

- *URLError* - if there is problem downloading the url
- *RuntimeError* - if the hash does not match the *hash\_prefix*

## Notes

Original code taken from *pytorch* in *torch/utils/model\_zoo.py* and slightly modified.

## References

<http://blog.moleculea.com/2012/10/04/urlretrieve-progres-indicator/questions/15644964/python-progress-bar-and-downloads> <http://stackoverflow.com/questions/16694907/how-to-download-large-file-in-python-with-requests-py>

**CommandLine:** `python -m xdoctest ubelt.util_download download:1`

### Example

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

### Example

```
>>> # xdoctest: +REQUIRES(--network)
>>> import ubelt as ub
>>> import io
>>> url = 'http://i.imgur.com/rqwaDag.png'
>>> file = io.BytesIO()
>>> fpath = download(url, file)
>>> file.seek(0)
>>> data = file.read()
>>> assert ub.hash_data(data, hasher='sha1').startswith('f79ea24571')
```

### Example

```
>>> # xdoctest: +REQUIRES(--network)
>>> url = 'http://i.imgur.com/rqwaDag.png'
>>> fpath = download(url, hasher='sha1', hash_prefix=
↳ 'f79ea24571da6ddd2ba12e3d57b515249ecb8a35')
Downloading url='http://i.imgur.com/rqwaDag.png' to fpath=...rqwaDag.png
...
...1233/1233... rate=... Hz, eta=..., total=..., wall=...
```

### Example

```
>>> # xdoctest: +REQUIRES(--network)
>>> # test download from girder
>>> import pytest
>>> import ubelt as ub
>>> url = 'https://data.kitware.com/api/v1/item/5b4039308d777f2e6225994c/download'
>>> ub.download(url, hasher='sha512', hash_prefix='c98a46cb31205cf')
>>> with pytest.raises(RuntimeError):
>>>     ub.download(url, hasher='sha512', hash_prefix='BAD_HASH')
```

`ubelt.util_download.grabdata(url, fpath=None, dpath=None, fname=None, redo=False, verbose=1, appname=None, hash_prefix=None, hasher='sha512', **download_kw)`

Downloads a file, caches it, and returns its local path.

#### Parameters

- **url** (*str*) – url to the file to download
- **fpath** (*PathLike*) – The full path to download the file to. If unspecified, the arguments *dpath* and *fname* are used to determine this.

- **dpath** (*PathLike*) – 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*.
- **hash\_prefix** (*None or str*) – If specified, grabdata verifies that this matches the hash of the file, and then saves the hash in a adjacent file to certify that the download was successful. Defaults to None.
- **hasher** (*str or Hasher*) – If hash\_prefix is specified, this indicates the hashing algorithm to apply to the file. Defaults to sha512.
- **\*\*download\_kw** – additional kwargs to pass to *ub.download*

**Returns** *fpath* - file path string

**Return type** *PathLike*

### Example

```
>>> # xdoctest: +REQUIRES(--network)
>>> import ubelt as ub
>>> url = 'http://i.imgur.com/rqwaDag.png'
>>> fpath = ub.grabdata(url, fname='mario.png')
>>> result = basename(fpath)
>>> print(result)
mario.png
```

### Example

```
>>> # xdoctest: +REQUIRES(--network)
>>> import ubelt as ub
>>> fname = 'foo.bar'
>>> url = 'http://i.imgur.com/rqwaDag.png'
>>> prefix1 = '944389a39dfb8fa9'
>>> fpath = ub.grabdata(url, fname=fname, hash_prefix=prefix1)
>>> stamp_fpath = fpath + '.hash'
>>> assert open(stamp_fpath, 'r').read() == prefix1
>>> # Check that the download doesn't happen again
>>> fpath = ub.grabdata(url, fname=fname, hash_prefix=prefix1)
>>> # todo: check file timestamps have not changed
>>> #
>>> # Check redo works with hash
>>> fpath = ub.grabdata(url, fname=fname, hash_prefix=prefix1, redo=True)
>>> # todo: check file timestamps have changed
>>> #
>>> # Check that a redownload occurs when the stamp is changed
>>> open(stamp_fpath, 'w').write('corrupt-stamp')
>>> fpath = ub.grabdata(url, fname=fname, hash_prefix=prefix1)
```

(continues on next page)

(continued from previous page)

```

>>> assert open(stamp_fpath, 'r').read() == prefix1
>>> #
>>> # Check that a redownload occurs when the stamp is removed
>>> ub.delete(stamp_fpath)
>>> open(fpath, 'w').write('corrupt-data')
>>> assert not ub.hash_file(fpath, base='hex', hasher='sha512').
↳startswith(prefix1)
>>> fpath = ub.grabdata(url, fname=fname, hash_prefix=prefix1)
>>> assert ub.hash_file(fpath, base='hex', hasher='sha512').startswith(prefix1)
>>> #
>>> # Check that requesting new data causes redownload
>>> url2 = 'https://data.kitware.com/api/v1/item/5b4039308d777f2e6225994c/download
↳'
>>> prefix2 = 'c98a46cb31205cf'
>>> fpath = ub.grabdata(url2, fname=fname, hash_prefix=prefix2)
>>> assert open(stamp_fpath, 'r').read() == prefix2

```

### 1.1.1.10 ubelt.util\_format module

Defines the function `repr2`, which allows for a bit more customization than `repr` or `pprint`. See the docstring for more details.

`ubelt.util_format.repr2(data, **kwargs)`

Makes a pretty and easy-to-doctest 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.

#### Notes

This function has many keyword arguments that can be used to customize the final representation. For convenience some of the more frequently used kwargs have short aliases. See *Args* for more details.

#### Parameters

- **data** (*object*) – an arbitrary python object
- **\*\*kwargs** – see *the Kwargs* section

#### Kwargs:

**si, stritems, (bool):** dict/list items use str instead of repr

**strkeys, sk (bool):** dict keys use str instead of repr

**strvals, sv (bool):** dict values use str instead of repr

**nl, newlines (int | bool):** number of top level nestings to place a newline after. If true all items are followed by newlines regardless of nesting level. Defaults to 1 for lists and True for dicts.

**nobr, nobraces (bool, default=False):** if True, text will not contain outer braces for containers

**cbr, compact\_brace (bool, default=False):** if True, braces are compactified (i.e. they will not have newlines placed directly after them, think java / K&R / 1TBS)

**trailsep, trailing\_sep (bool):** if True, a separator is placed after the last item in a sequence. By default this is True if there are any `nl > 0`.

**explicit (bool, default=False):** changes dict representation from `{k1: v1, ...}` to `dict(k1=v1, ...)`.

**precision (int, default=None):** if specified floats are formatted with this precision

**kvsep (str, default=': '):** separator between keys and values

**itemsep (str, default=' '):** separator between items

**sort (bool):** if True, attempts to sort all unordered collections in the returned text. NOTE: currently if True this will sort lists, this may not be a correct thing to do, as such the behavior of this arg is subject to change.

**suppress\_small (bool):** passed to *numpy.array2string* for ndarrays

**max\_line\_width (int):** passed to *numpy.array2string* for ndarrays

**with\_dtype (bool):** only relevant to ndarrays. if True includes the dtype.

**Returns** outstr: output string

**Return type** str

## Notes

**There are also internal kwargs, which should not be used:** `_return_info` (bool): return information about child context `_root_info` (depth): information about parent context

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

## 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)
```

### 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)
>>> result = repr2(dict_, nl=-1, precision=2)
>>> print('---')
>>> print(result)
```

**class** `ubelt.util_format.FormatterExtensions`

Bases: `object`

Helper class for managing non-builtin (e.g. numpy) format types.

This module (`ubelt.util_format`) maintains a global set of basic extensions, but it is also possible to create a locally scoped set of extensions and explicitly pass it to `repr2`. The following example demonstrates this.

### Example

```
>>> import ubelt as ub
>>> class MyObject(object):
>>>     pass
>>> data = {'a': [1, 2.2222, MyObject()], 'b': MyObject()}
>>> # Create a custom set of extensions
>>> extensions = ub.FormatterExtensions()
>>> # Register a function to format your specific type
>>> @extensions.register(MyObject)
>>> def format_myobject(data, **kwargs):
>>>     return 'I can do anything here'
>>> # Repr2 will now respect the passed custom extensions
>>> # Note that the global extensions will still be respected
>>> # unless they are overloaded.
>>> print(ub.repr2(data, nl=-1, precision=1, extensions=extensions))
{
    'a': [1, 2.2, I can do anything here],
    'b': I can do anything here
}
>>> # Overload the formatter for float and int
>>> @extensions.register((float, int))
>>> def format_myobject(data, **kwargs):
>>>     return str((data + 10) // 2)
>>> print(ub.repr2(data, nl=-1, precision=1, extensions=extensions))
{
    'a': [5, 6.0, I can do anything here],
    'b': I can do anything here
}
```

**register** (*type*)

Registers a custom formatting function with `ub.repr2`



**lookup** (*data*)

Returns an appropriate function to format *data* if one has been registered.

#### 1.1.1.11 ubelt.util\_func module

Helpers for functional programming. Currently this module contains an identify function that simply returns its own inputs and a function to “inject” another function into a class instance as a method. This is useful for monkey patching.

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

The identity function. Simply returns its inputs.

**Parameters** *arg* (*object*) – some value

**Returns** *arg*: the same value

**Return type** *object*

#### 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

The main use case of this function is for monkey patching. While monkey patching is sometimes necessary it should generally be avoided. Thus, we simply remind the developer that there might be a better way.

**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.

#### 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'
```

#### 1.1.1.12 ubelt.util\_hash module

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

The hashes are deterministic across python versions and operating systems. This is verified by CI testing on Windows, Linux, Python with 2.7, 3.4, and greater, and on 32 and 64 bit versions.

**Use Case:** Problem: You have data that you want to hash. Assumptions: The data is in standard python scalars or ordered sequences:

e.g. tuple, list, odict, oset, int, str, etc...

Solution: `ub.hash_data`

### Example

```
>>> import ubelt as ub
>>> data = ub.odict(sorted({
>>>     'param1': True,
>>>     'param2': 0,
>>>     'param3': [None],
>>>     'param4': ('str', 4.2),
>>> }.items()))
>>> # hash_data can hash any ordered builtin object
>>> ub.hash_data(data, convert=False, hasher='sha512')
2ff39d0ecbf6ecc740ca7d...
```

**Use Case:**

**Problem:** You have a file you want to hash, but your system doesn't have a `shasum` executable (or you don't want to use `Popen`).

Solution: `ub.hash_file`

### Example

```
>>> import ubelt as ub
>>> from os.path import join
>>> fpath = ub.touch(join(ub.ensure_app_cache_dir('ubelt'), 'empty_file'))
>>> ub.hash_file(fpath, hasher='sha1')
da39a3ee5e6b4b0d3255bfe95601890afd80709
```

---

**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, base=NoParam, types=False, hashlen=NoParam, convert=False)`  
Get a unique hash depending on the state of the data.

#### Parameters

- **data** (*object*) – Any sort of loosely organized data
- **hasher** (*str or HASHER*) – Hash algorithm from hashlib, defaults to `sha512`.
- **base** (*str or List[str]*) – Shorthand key or a list of symbols. Valid keys are: 'abc', 'hex', and 'dec'. Defaults to 'hex'.
- **types** (*bool*) – If True data types are included in the hash, otherwise only the raw data is hashed. Defaults to False.

- **hashlen** (*int*) – Maximum number of symbols in the returned hash. If not specified, all are returned. DEPRECATED. Use slice syntax instead.
- **convert** (*bool, optional, default=True*) – if True, try and convert the data to json an the json is hashed instead. This can improve runtime in some instances, however the hash may differ from the case where convert=False.

## Notes

alphabet26 is a pretty nice base, I recommend it. However we default to hex because it is standard. This means the output of hashdata with base=sha1 will be the same as the output of *sha1sum*.

**Returns** text - hash string

**Return type** str

## Example

```
>>> import ubelt as ub
>>> print(ub.hash_data([1, 2, (3, '4')], convert=False))
60b758587f599663931057e6ebdf185a...
>>> print(ub.hash_data([1, 2, (3, '4')], base='abc', hasher='sha512')[:32])
hsrgqvfiuxvvhcdnypivhhthmrolkzej
```

`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** (*PathLike*) – 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, str*) – list of symbols or shorthand key. Valid keys are 'abc', 'hex', and 'dec'. Defaults to 'hex'.

## Notes

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

## References

<http://stackoverflow.com/questions/3431825/md5-checksum-of-a-file>  
[5001893/when-to-use-sha-1-vs-sha-2](http://stackoverflow.com/questions/5001893/when-to-use-sha-1-vs-sha-2)

<http://stackoverflow.com/questions/>

### 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='sha1', base='hex'))
8843d7f92416211de9ebb963ff4ce28125932878
```

### Example

```
>>> import ubelt as ub
>>> from os.path import join
>>> fpath = ub.touch(join(ub.ensure_app_cache_dir('ubelt'), 'empty_file'))
>>> # Test that the output is the same as shasum
>>> if ub.find_exe('shasum'):
>>>     want = ub.cmd(['shasum', fpath], verbose=2)['out'].split(' ')[0]
>>>     got = ub.hash_file(fpath, hasher='sha1')
>>>     print('want = {!r}'.format(want))
>>>     print('got = {!r}'.format(got))
>>>     assert want.endswith(got)
>>> # Do the same for sha512 sum and md5sum
>>> if ub.find_exe('sha512sum'):
>>>     want = ub.cmd(['sha512sum', fpath], verbose=2)['out'].split(' ')[0]
>>>     got = ub.hash_file(fpath, hasher='sha512')
>>>     print('want = {!r}'.format(want))
>>>     print('got = {!r}'.format(got))
>>>     assert want.endswith(got)
>>> if ub.find_exe('md5sum'):
>>>     want = ub.cmd(['md5sum', fpath], verbose=2)['out'].split(' ')[0]
>>>     got = ub.hash_file(fpath, hasher='md5')
>>>     print('want = {!r}'.format(want))
>>>     print('got = {!r}'.format(got))
>>>     assert want.endswith(got)
```

#### 1.1.1.13 ubelt.util\_import module

Expose functions to simplify importing from module names and paths.

**Partial Autogenerate:** import netharn as nh closer = nh.export.closer.Closer() from xdoctest import static\_analysis as static from xdoctest import utils

```
closer.add_dynamic(static.split_modpath) closer.add_dynamic(static.modpath_to_modname)
closer.add_dynamic(static.modname_to_modpath) closer.add_dynamic(utils.import_module_from_name)
closer.add_dynamic(utils.import_module_from_path)
```

```
closer.expand(['xdoctest']) text = closer.current_sourcecode() print(text)
```

```
text = text.replace(' from xdoctest.static_analysis import split_modpath, modpath_to_modnamen', '')
```

```
fpath = ub.expandpath('~/.code/ubelt/ubelt/_autogen_util_import.py') open(fpath, 'w').write(text + 'n')
```

ubelt.util\_import.**import\_module\_from\_path**(modpath, index=-1)

Imports a module via its path

**Parameters** modpath (*PathLike*) – path to the module on disk or within a zipfile.

**Returns** the imported module

**Return type** module

## References

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

## 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

This can import a module from within a zipfile. To do this modpath should specify the path to the zipfile and the path to the module within that zipfile separated by a colon or pathsep. E.g. `/path/to/archive.zip:mymodule.py`

**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.

## Example

```
>>> import xdoctest
>>> modpath = xdoctest.__file__
>>> module = import_module_from_path(modpath)
>>> assert module is xdoctest
```

## Example

```
>>> # Test importing a module from within a zipfile
>>> import zipfile
>>> from xdoctest import utils
>>> from os.path import join, expanduser
>>> dpath = expanduser('~/.cache/xdoctest')
>>> dpath = utils.ensuredir(dpath)
>>> #dpath = utils.TempDir().ensure()
>>> # Write to an external module named bar
>>> external_modpath = join(dpath, 'bar.py')
```

(continues on next page)

(continued from previous page)

```

>>> open(external_modpath, 'w').write('testvar = 1')
>>> internal = 'folder/bar.py'
>>> # Move the external bar module into a zipfile
>>> zippath = join(dpath, 'myzip.zip')
>>> with zipfile.ZipFile(zippath, 'w') as myzip:
>>>     myzip.write(external_modpath, internal)
>>> # Import the bar module from within the zipfile
>>> modpath = zippath + ':' + internal
>>> modpath = zippath + os.path.sep + internal
>>> module = import_module_from_path(modpath)
>>> assert module.__name__ == os.path.normpath('folder/bar')
>>> assert module.testvar == 1

```

**Doctest:**

```

>>> import pytest
>>> with pytest.raises(IOError):
>>>     import_module_from_path('does-not-exist')
>>> with pytest.raises(IOError):
>>>     import_module_from_path('does-not-exist.zip/')

```

`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

**Example**

```

>>> # test with modules that wont be imported in normal circumstances
>>> # todo write a test where we gaurentee this
>>> modname_list = [
>>>     '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.modname_to_modpath(modname, hide_init=True, hide_main=False, 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 xdoctest.static\_analysis modname\_to\_modpath:0 pytest  
/home/joncrall/code/xdoctest/xdoctest/static\_analysis.py::modname\_to\_modpath:0

### Example

```
>>> modname = 'xdoctest.__main__'
>>> modpath = modname_to_modpath(modname, hide_main=False)
>>> assert modpath.endswith('__main__.py')
>>> modname = 'xdoctest'
>>> modpath = modname_to_modpath(modname, hide_init=False)
>>> assert modpath.endswith('__init__.py')
>>> modpath = basename(modname_to_modpath('_ctypes'))
>>> assert 'ctypes' in modpath
```

```
ubelt.util_import.modpath_to_modname(modpath, hide_init=True, hide_main=False,
                                     check=True, relativeto=None)
```

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)
- **check** (*bool*) – if False, does not raise an error if modpath is a dir and does not contain an `__init__` file.
- **relativeto** (*str, optional*) – if specified, all checks are ignored and this is considered the path to the root module.

**Returns** modname

**Return type** str

**Raises** ValueError – if check is True and the path does not exist

**CommandLine:** xdoctest -m xdoctest.static\_analysis modpath\_to\_modname

### Example

```
>>> from xdoctest import static_analysis
>>> modpath = static_analysis.__file__.replace('.pyc', '.py')
>>> modpath = modpath.replace('.pyc', '.py')
>>> modname = modpath_to_modname(modpath)
>>> assert modname == 'xdoctest.static_analysis'
```

### Example

```
>>> import xdoctest
>>> assert modpath_to_modname(xdoctest.__file__.replace('.pyc', '.py')) ==
↳ 'xdoctest'
>>> assert modpath_to_modname(dirname(xdoctest.__file__.replace('.pyc', '.py')))_
↳ == 'xdoctest'
```

### Example

```
>>> modpath = modname_to_modpath('_ctypes')
>>> modname = modpath_to_modname(modpath)
>>> assert modname == '_ctypes'
```

`ubelt.util_import.split_modpath(modpath, check=True)`

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
- **check** (*bool*) – if False, does not raise an error if modpath is a directory and does not contain an `__init__.py` file.

**Returns** (directory, rel\_modpath)

**Return type** tuple

**Raises** `ValueError` – if modpath does not exist or is not a package

### Example

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

#### 1.1.1.14 ubelt.util\_io module

Functions for reading and writing files on disk.

`writeto` and `readfrom` wrap `open().write()` and `open().read()` and primarily 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. It also contains workarounds for win32 issues with *shutil*.

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

Writes (utf8) text to a file.

#### Parameters

- **fpath** (*PathLike*) – 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`

#### Example

```
>>> 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
```

#### Example

```
>>> 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** (*PathLike*) – 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** (*PathLike*) – 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).

**Returns** path to the file

**Return type** PathLike

**References**

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

**Example**

```
>>> 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 does nothing.

**Parameters**

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

**SeeAlso:**

**send2trash** - A cross-platform Python package for sending files to the trash instead of irreversibly deleting them. <https://github.com/hsoft/send2trash>

**Doctest:**

```
>>> import ubelt as ub
>>> base = ub.ensure_app_cache_dir('ubelt', 'delete_test')
>>> dpath1 = ub.ensure_dir(join(base, 'dir'))
>>> ub.ensure_dir(join(base, 'dir', 'subdir'))
>>> ub.touch(join(base, 'dir', 'to_remove1.txt'))
>>> fpath1 = join(base, 'dir', 'subdir', 'to_remove3.txt')
```

(continues on next page)

(continued from previous page)

```
>>> 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)))
```

**Doctest:**

```
>>> import ubelt as ub
>>> dpath = ub.ensure_app_cache_dir('ubelt', 'delete_test2')
>>> dpath1 = ub.ensure_dir(join(dpath, 'dir'))
>>> fpath1 = ub.touch(join(dpath1, 'to_remove.txt'))
>>> assert exists(fpath1)
>>> ub.delete(dpath)
>>> assert not exists(fpath1)
```

**1.1.1.15 ubelt.util\_links module**

Cross-platform logic for dealing with symlinks. Basic functionality should work on all operating systems including everyone's favorite pathological OS (note that there is an additional helper file for this case), but there are some corner cases depending on your version. Recent versions of Windows tend to work, but there certain system settings that cause issues. Obviously, any POSIX system work without difficulty.

**CommandLine:** xdoctest -m ubelt.util\_links \_\_doc\_\_:0

**Example**

```
>>> import ubelt as ub
>>> from os.path import normpath, join
>>> dpath = ub.ensure_app_cache_dir('ubelt', normpath('demo/symlink'))
>>> real_path = join(dpath, 'real_file.txt')
>>> link_path = join(dpath, 'link_file.txt')
>>> ub.touch(real_path)
>>> result = ub.symlink(real_path, link_path, overwrite=True, verbose=3)
>>> parts = result.split(os.path.sep)
>>> print(parts[-1])
link_file.txt
```

`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** (*PathLike*) – path to real file or directory
- **link\_path** (*PathLike*) – 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** PathLike

**CommandLine:** python -m ubelt.util\_links symlink:0

### 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]]
```

### 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)
```

### 1.1.1.16 ubelt.util\_list module

Utility functions for manipulating iterables, lists, and sequences.

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

Bases: `object`

Generates successive n-sized chunks from *items*.

If the last chunk has less than n elements, *bordermode* is used to determine fill values.

#### Parameters

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

---

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

---

#### References

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

**CommandLine:** `python -m ubelt.util_list chunks`

#### Example

```
>>> import ubelt as ub
>>> items = [1, 2, 3, 4, 5, 6, 7]
>>> genresult = ub.chunks(items, chunksize=3, bordermode='none')
>>> assert list(genresult) == [[1, 2, 3], [4, 5, 6], [7]]
>>> genresult = ub.chunks(items, chunksize=3, bordermode='cycle')
>>> assert list(genresult) == [[1, 2, 3], [4, 5, 6], [7, 1, 2]]
>>> genresult = ub.chunks(items, 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
>>> # Note: ub.chunks will not do the 2,1,1 split
>>> assert len(list(ub.chunks(range(4), nchunks=3))) == 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** (*items, chunksize*)

**static cycle** (*items, chunksize*)

**static replicate** (*items, 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

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

**Returns** subset of the list

**Return type** Iterable or scalar

**SeeAlso:** `ub.dict_subset`

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

**Returns** a subset of masked items

**Return type** `Iterable`

### 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)`

Transforms a nested iterable into a flat iterable.

This is simply an alias for `itertools.chain.from_iterable`

**Parameters** `nested_list` (*Iterable[Iterable]*) – list of lists

**Returns** flattened items

**Return type** `Iterable`

### 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** (*Iterable*) – list of items
- **key** (*Callable, 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`

### 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]
```

### 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** (*Sequence*) – indexable collection of items
- **key** (*Callable, optional*) – custom normalization function. If specified returns items where `key(item)` is unique.

**Yields** *int* – indices of the unique items

### 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** (*Sequence*) – indexable collection of items
- **key** (*Callable, optional*) – custom normalization function. If specified returns items where `key(item)` is unique.

**Returns** flags the items that are unique

**Return type** List[bool]

### Example

```
>>> import ubelt as ub
>>> items = [0, 2, 1, 1, 0, 9, 2]
>>> flags = unique_flags(items)
```

(continues on next page)



(continued from previous page)

```
>>> 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 inferred from *indices*

---

**Note:** In the future the arg *maxval* may change its name to *shape*

---

**Returns** mask: list of booleans. mask[idx] is True if idx in indices

**Return type** list

#### 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** (*Iterable*) – 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

#### 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)]
```

### 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)]
```

### 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)]
```

### 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** (*Iterable*) – items to determine if they are all the same
- **eq** (*Callable, optional*) – function to determine equality (default: `operator.eq`)

### 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 *numpy.argsort*, but it is written in pure python and works on both lists and dictionaries.

#### Parameters

- **indexable** (*Iterable or Mapping*) – indexable to sort by
- **key** (*Callable, 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

#### 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.

This is similar to *numpy.argmax*, but it is written in pure python and works on both lists and dictionaries.

#### Parameters

- **indexable** (*Iterable or Mapping*) – indexable to sort by
- **key** (*Callable, optional*) – customizes the ordering of the indexable

**CommandLine:** `python -m ubelt.util_list argmax`

#### 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
```

(continues on next page)

(continued from previous page)

```
>>> assert argmax({'a': 3, 'b': 2, 3: 100, 4: 4}) == 3
>>> assert argmax(iter(['a', 'c', 'b', 'z', 'f'])) == 3
```

`ubelt.util_list.argmaxin(indexable, key=None)`

Returns index / key of the item with the smallest value.

This is similar to `numpy.argmin`, but it is written in pure python and works on both lists and dictionaries.

#### Parameters

- **indexable** (*Iterable or Mapping*) – indexable to sort by
- **key** (*Callable, optional*) – customizes the ordering of the indexable

#### 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'
>>> assert argmin(iter(['a', 'c', 'A', 'z', 'f'])) == 2
```

`ubelt.util_list.peek(iterable)`

Look at the first item of an iterable. If the input is an iterator, then the next element is exhausted (i.e. a pop operation).

**Parameters** `iterable` (*List[T]*) – an iterable

#### Returns

**item:** the first item of ordered sequence, a popped item from an iterator, or an arbitrary item from an unordered collection.

**Return type** `T`

#### Example

```
>>> import ubelt as ub
>>> data = [0, 1, 2]
>>> ub.peek(data)
0
>>> iterator = iter(data)
>>> print(ub.peek(iterator))
0
>>> print(ub.peek(iterator))
1
>>> print(ub.peek(iterator))
2
>>> ub.peek(range(3))
0
```

### 1.1.1.17 ubelt.util\_memoize module

This module exposes decorators for in-memory caching of functional results.

## Example

```

>>> import ubelt as ub
>>> # Memoize a function, the args are hashed
>>> @ub.memoize
>>> def func(a, b):
>>>     return a + b
>>> #
>>> class MyClass:
>>>     # Memoize a class method, the args are hashed
>>>     @ub.memoize_method
>>>     def my_method(self, a, b):
>>>         return a + b
>>>     #
>>>     # Memoize a property: there can be no args,
>>>     @ub.memoize_property
>>>     @property
>>>     def my_property1(self):
>>>         return 4
>>>     #
>>>     # The property decorator is optional
>>>     def my_property2(self):
>>>         return 5
>>> #
>>> func(1, 2)
>>> func(1, 2)
>>> self = MyClass()
>>> self.my_method(1, 2)
>>> self.my_method(1, 2)
>>> self.my_property1
>>> self.my_property1
>>> self.my_property2
>>> self.my_property2

```

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

## References

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

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

**Returns** memoized wrapper

**Return type** `func`

**CommandLine:** `xdoctest -m ubelt.util_memoize memoize`

## Example

```

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

```

(continues on next page)

(continued from previous page)

```
>>>     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

## References

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

## 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
```

(continues on next page)

(continued from previous page)

```

>>> 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

```

`ubelt.util_memoize.memoize_property` (*fget*)

Return a property attribute for new-style classes that only calls its getter on the first access. The result is stored and on subsequent accesses is returned, preventing the need to call the getter any more.

This decorator can either be used by itself or by decorating another property. In either case the method will always become a property.

## Notes

implementation is a modified version of [1].

## References

..[1] <https://github.com/estebistec/python-memoized-property>

**CommandLine:** `xdoctest -m ubelt.util_memoize memoize_property`

## Example

```

>>> class C(object):
...     load_name_count = 0
...     @memoize_property
...     def name(self):
...         "name's docstring"
...         self.load_name_count += 1
...         return "the name"
...     @memoize_property
...     @property
...     def another_name(self):
...         "name's docstring"
...         self.load_name_count += 1
...         return "the name"
>>> c = C()
>>> c.load_name_count
0
>>> c.name
'the name'
>>> c.load_name_count
1
>>> c.name
'the name'
>>> c.load_name_count
1
>>> c.another_name

```

### 1.1.1.18 ubelt.util\_mixins module

This module defines the *NiceRepr* mixin class, which defines a `__repr__` and `__str__` method that only depend on a custom `__nice__` method, which you must define. This means you only have to overload one function instead of two. Furthermore, if the object defines a `__len__` method, then the `__nice__` method defaults to something sensible, otherwise it is treated as abstract and raises *NotImplementedError*.

**class** `ubelt.util_mixins.NiceRepr`

Bases: `object`

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

#### Example

```
>>> import ubelt as ub
>>> class Foo(ub.NiceRepr):
...     def __nice__(self):
...         return 'info'
>>> foo = Foo()
>>> assert str(foo) == '<Foo(info)>'
>>> assert repr(foo).startswith('<Foo(info) at ')
```

#### Example

```
>>> import ubelt as ub
>>> class Bar(ub.NiceRepr):
...     pass
>>> bar = Bar()
>>> import pytest
>>> with pytest.warns(None) as record:
>>>     assert 'object at' in str(bar)
>>>     assert 'object at' in repr(bar)
```

#### Example

```
>>> import ubelt as ub
>>> class Baz(ub.NiceRepr):
...     def __len__(self):
...         return 5
>>> baz = Baz()
>>> assert str(baz) == '<Baz(5)>'
```

### 1.1.1.19 ubelt.util\_path module

Functions for working with filesystem paths.

`ubelt.util_path.augpath(path, suffix="", prefix="", ext=None, base=None, multidot=False)`

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** (*PathLike*) – string representation of a path
- **suffix** (*str*) – placed between the basename and extension
- **prefix** (*str*) – placed in front of the basename
- **ext** (*str*) – if specified, replaces the extension
- **base** (*str*) – if specified, replaces the basename (without extension)
- **multidot** (*bool*) – if False, everything after the last dot in the basename is the extension. If True, everything after the first dot in the basename is the extension (Defaults to False).

**Returns** augmented path**Return type** PathLike**CommandLine:** python -m ubelt.util\_path augpath**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
```

**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'
>>> augpath('foo.tar.gz', ext='.zip', multidot=True)
foo.zip
>>> augpath('foo.tar.gz', ext='.zip', multidot=False)
foo.tar.zip
```

**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** PathLike

### 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** (*PathLike*) – 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** PathLike

**CommandLine:** `xdoctest -m ubelt.util_path compressuser`

### 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.expandpath(path)`

Wrapper around `expanduser` and `expandvars`.

Less aggressive than `truepath`. Only expands environs and tilde. Does not change relative paths to absolute paths.

**Parameters** **path** (*PathLike*) – string representation of a path

**Returns** expanded path

**Return type** PathLike

### Example

```
>>> import ubelt as ub
>>> assert normpath(ub.expandpath('~foo')) == join(ub.userhome(), 'foo')
>>> assert ub.expandpath('foo') == 'foo'
```

`ubelt.util_path.truepath(path, real=False)`

Normalizes a string representation of a path and does shell-like expansion.

#### Parameters

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

**Returns** normalized path

**Return type** PathLike

**Note:** This function is similar 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`

### 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** (*PathLike*) – 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** PathLike

### Notes

This function is not thread-safe in Python2

### 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.

### Example

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

### Example

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

**ensure()**

**cleanup()**

**start()**

#### 1.1.1.20 ubelt.util\_platform module

The goal of this module is to provide an idomatic cross-platform pattern of accessing platform dependant file systems. Standard application directory structure: cache, config, and other XDG standards [1].

### Notes

Table mapping the type of directory to the system default environment variable. Inspired by [2,3,4].

Linux | Win32 | Darwin

data | \$XDG\_DATA\_HOME | %APPDATA% | ~/Library/Application Support  
 config | \$XDG\_CONFIG\_HOME | %APPDATA%\config | ~/Library/Application Support/config  
 cache | \$XDG\_CACHE\_HOME | %LOCALAPPDATA%\Cache | ~/Library/Caches

**If an environment variable is not specified the defaults are:** APPDATA = ~/AppData/Roaming LOCALAPPDATA = ~/AppData/Local

XDG\_DATA\_HOME = ~/.local/share XDG\_CACHE\_HOME = ~/.cache XDG\_CONFIG\_HOME = ~/.config

### References

..[1] <https://specifications.freedesktop.org/basedir-spec/basedir-spec-latest.html> ..[2] <https://stackoverflow.com/questions/43853548/xdg-windows> ..[3] <https://stackoverflow.com/questions/11113974/cross-plat-path> ..[4] <https://github.com/harawata/appdirs#supported-directories>

`ubelt.util_platform.platform_data_dir()`

Returns path for user-specific data files

**Returns** path to the data dir used by the current operating system

**Return type** PathLike

```
ubelt.util_platform.platform_config_dir()
```

Returns a directory which should be writable for any application This should be used for persistent configuration files.

**Returns** path to the cahce dir used by the current operating system

**Return type** PathLike

```
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** PathLike

```
ubelt.util_platform.get_app_data_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 data directory for this application

**Return type** PathLike

**SeeAlso:** ensure\_app\_data\_dir

```
ubelt.util_platform.ensure_app_data_dir(appname, *args)
```

Calls *get\_app\_data\_dir* but ensures the directory exists.

**Parameters**

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

**SeeAlso:** get\_app\_data\_dir

## Example

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

```
ubelt.util_platform.get_app_config_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 config directory for this application

**Return type** PathLike

**SeeAlso:** ensure\_app\_config\_dir

`ubelt.util_platform.ensure_app_config_dir(appname, *args)`  
 Calls `get_app_config_dir` but ensures the directory exists.

**Parameters**

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

**SeeAlso:** `get_app_config_dir`

**Example**

```
>>> import ubelt as ub
>>> dpath = ub.ensure_app_config_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** `PathLike`

**SeeAlso:** `ensure_app_cache_dir`

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

**Parameters**

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

**SeeAlso:** `get_app_cache_dir`

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

## 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.find_exe(name, multi=False, path=None)`

Locate a command.

Search your local filesystem for an executable and return the first matching file with executable permission.

#### Parameters

- **name** (*str*) – globstr of matching filename
- **multi** (*bool*) – if True return all matches instead of just the first. Defaults to False.
- **path** (*str or Iterable[PathLike]*) – overrides the system PATH variable.

**Returns** returns matching executable(s).

**Return type** PathLike or List[PathLike] or None

**SeeAlso:** `shutil.which` - which is available in Python 3.3+.

## Notes

This is essentially the *which* UNIX command

## References

<https://stackoverflow.com/questions/377017/test-if-executable-exists-in-python/377028#377028>

<https://docs.python.org/dev/library/shutil.html#shutil.which>

## Example

```
>>> find_exe('ls')
>>> find_exe('ping')
>>> assert find_exe('which') == find_exe(find_exe('which'))
>>> find_exe('which', multi=True)
>>> find_exe('ping', multi=True)
>>> find_exe('cmake', multi=True)
>>> find_exe('nvcc', multi=True)
>>> find_exe('noexist', multi=True)
```

### Example

```
>>> assert not find_exe('noexist', multi=False)
>>> assert find_exe('ping', multi=False)
>>> assert not find_exe('noexist', multi=True)
>>> assert find_exe('ping', multi=True)
```

### Benchmark:

```
>>> # xdoctest: +IGNORE_WANT
>>> import ubelt as ub
>>> import shutil
>>> for timer in ub.Timerit(100, bestof=10, label='ub.find_exe'):
>>>     ub.find_exe('which')
>>> for timer in ub.Timerit(100, bestof=10, label='shutil.which'):
>>>     shutil.which('which')
Timed best=58.71  $\mu$ s, mean=59.64  $\pm$  0.96  $\mu$ s for ub.find_exe
Timed best=72.75  $\mu$ s, mean=73.07  $\pm$  0.22  $\mu$ s for shutil.which
```

`ubelt.util_platform.find_path(name, path=None, exact=False)`

Search for a file or directory on your local filesystem by name (file must be in a directory specified in a PATH environment variable)

#### Parameters

- **fname** (*PathLike or str*) – file name to match. If exact is False this may be a glob pattern
- **path** (*str or Iterable[PathLike]*) – list of directories to search either specified as an `os.pathsep` separated string or a list of directories. Defaults to environment PATH.
- **exact** (*bool*) – if True, only returns exact matches. Default False.

### Notes

For recursive behavior set `path=(d for d, _, _ in os.walk('.'))`, where `'.'` might be replaced by the root directory of interest.

### Example

```
>>> list(find_path('ping', exact=True))
>>> list(find_path('bin'))
>>> list(find_path('bin'))
>>> list(find_path('*cc*'))
>>> list(find_path('cmake*'))
```

### Example

```
>>> import ubelt as ub
>>> from os.path import dirname
>>> path = dirname(dirname(ub.util_platform.__file__))
>>> res = sorted(find_path('ubelt/util_*.py', path=path))
>>> assert len(res) >= 10
>>> res = sorted(find_path('ubelt/util_platform.py', path=path, exact=True))
```

(continues on next page)



(continued from previous page)

```
>>> print(res)
>>> assert len(res) == 1
```

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

DEPRICATED: This has been ported to xdev, please use that version.

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** (*PathLike*) – 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)
```

`ubelt.util_platform.platform_resource_dir()`

Alias for `platform_cache_dir`

DEPRICATED in favor of `platform_config_dir` / `platform_data_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** `PathLike`

`ubelt.util_platform.get_app_resource_dir(appname, *args)`

Returns a writable directory for an application This should be used for persistent configuration files.

DEPRICATED in favor of `get_app_config_dir` / `get_app_data_dir`

#### 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** `PathLike`

**SeeAlso:** `ensure_app_resource_dir`

`ubelt.util_platform.ensure_app_resource_dir(appname, *args)`

Calls `get_app_resource_dir` but ensures the directory exists.

DEPRICATED in favor of `ensure_app_config_dir` / `ensure_app_data_dir`

#### Parameters

- **appname** (*str*) – the name of the application

- **\*args** – any other subdirectories may be specified

**SeeAlso:** `get_app_resource_dir`

### 1.1.1.21 `ubelt.util_str` module

Functions for working with text strings.

`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`

#### Example

```
>>> from ubelt.util_str import * # NOQA
>>> NL = chr(10) # newline character
>>> text = 'Lorem ipsum' + NL + 'dolor sit amet'
>>> prefix = '    '
>>> result = indent(text, prefix)
>>> assert all(t.startswith(prefix) for t in result.split(NL))
```

`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`

#### 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'
...         '''
>>>     )
```

(continues on next page)

(continued from previous page)

```
>>> # 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

Concatenates 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[str])` – strings to concatenate
- `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
>>> import unicodedata
>>> 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)

#### References

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

## Example

```
>>> from ubelt.util_str import *
>>> import codecs # NOQA
>>> assert ensure_unicode('my ünïcôdé strıng') == 'my ünïcôdé strıng'
>>> assert ensure_unicode('text1') == 'text1'
>>> assert ensure_unicode('text1'.encode('utf8')) == 'text1'
>>> assert ensure_unicode('ı»ıtext1'.encode('utf8')) == 'ı»ıtext1'
>>> assert (codecs.BOM_UTF8 + 'textı»ı'.encode('utf8')).decode('utf8')
```

### 1.1.1.22 ubelt.util\_stream module

Functions for capturing and redirecting IO streams.

**class** `ubelt.util_stream.TeeStringIO` (*redirect=None*)

Bases: `_io.StringIO`

An IO object that writes to itself and another IO stream.

**Variables** `redirect` (*io.IOBase*) – The other stream to write to.

## Example

```
>>> redirect = io.StringIO()
>>> self = TeeStringIO(redirect)
```

**isatty** ()

Returns true if the redirect is a terminal.

## Notes

Needed for IPython.embed to work properly when this class is used to override stdout / stderr.

**encoding**

Gets the encoding of the *redirect* IO object

**Doctest:**

```
>>> redirect = io.StringIO()
>>> assert TeeStringIO(redirect).encoding is None
>>> assert TeeStringIO(None).encoding is None
>>> assert TeeStringIO(sys.stdout).encoding is sys.stdout.encoding
>>> redirect = io.TextIOWrapper(io.StringIO())
>>> assert TeeStringIO(redirect).encoding is redirect.encoding
```

**write** (*msg*)

Write to this and the redirected stream

**flush** ()

Flush to this and the redirected stream

**class** `ubelt.util_stream.CaptureStream`

Bases: `object`

Generic class for capturing streaming output from stdout or stderr

**class** `ubelt.util_stream.CaptureStdout` (*supress=True, enabled=True*)

Bases: `ubelt.util_stream.CaptureStream`

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

#### Parameters

- **supress** (*bool, default=True*) – if True, stdout is not printed while captured
- **enabled** (*bool, default=True*) – does nothing if this is False

#### Example

```
>>> self = CaptureStdout(supress=True)
>>> print('dont capture the table flip (°° ')
>>> with self:
...     text = 'capture the heart '
...     print(text)
>>> print('dont capture look of disapproval _')
>>> assert isinstance(self.text, six.text_type)
>>> assert self.text == text + '\n', 'failed capture text'
```

#### Example

```
>>> self = CaptureStdout(supress=False)
>>> with self:
...     print('I am captured and printed in stdout')
>>> assert self.text.strip() == 'I am captured and printed in stdout'
```

#### Example

```
>>> self = CaptureStdout(supress=True, enabled=False)
>>> with self:
...     print('dont capture')
>>> assert self.text is None
```

**log\_part()**

Log what has been captured so far

**start()**

**stop()**

**Doctest:**

```
>>> CaptureStdout(enabled=False).stop()
>>> CaptureStdout(enabled=True).stop()
```

**close()**

### 1.1.1.23 ubelt.util\_time module

Timerit now exists in a standalone pip-installable module. The source code lives in its own github repo here: <https://github.com/Erotemic/timerit>

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

**Parameters** `method` (*str*) – type of timestamp

### Example

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

**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, defaults to True

#### Variables

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

**CommandLine:** `python -m timerit.core Timer`

### Example

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

### Example

```
>>> # Create and start the timer using the tic/toc interface
>>> from timerit import Timer
>>> 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=1, label=None, bestof=3, unit=None, 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 timerit.core Timerit:0

#### Example

```
>>> from timerit import Timerit
>>> import math
>>> num = 15
>>> t1 = Timerit(num, label='factorial', verbose=1)
>>> 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)
Timed best=..., mean=... for factorial
>>> # <you can now access Timerit attributes>
>>> assert t1.total_time > 0
>>> assert t1.n_loops == t1.num
>>> assert t1.n_loops == num
```

#### Example

```
>>> # xdoc: +IGNORE_WANT
>>> import math
>>> from timerit import Timerit
>>> num = 10
>>> # If the timer object is unused, time will still be recorded,
>>> # but with less precision.
>>> for _ in Timerit(num, 'concise', verbose=2):
>>>     math.factorial(10000)
Timed concise for: 10 loops, best of 3
    time per loop: best=4.954 ms, mean=4.972 ± 0.018 ms
>>> # Using the timer object results in the most precise timings
>>> for timer in Timerit(num, 'precise', verbose=3):
>>>     with timer: math.factorial(10000)
```

(continues on next page)

(continued from previous page)

```
Timing precise for: 15 loops, best of 3
Timed precise for: 15 loops, best of 3
    time per loop: best=2.474 ms, mean=2.54 ± 0.046 ms
```

**reset** (*label=None*)

clears all measurements, allowing the object to be reused

**Parameters** *label* (*str, optional*) – optionally change the label

### Example

```
>>> from timerit import Timerit
>>> import math
>>> ti = Timerit(num=10, unit='us', verbose=True)
>>> _ = ti.reset(label='10!').call(math.factorial, 10)
Timed best=...s, mean=...s for 10!
>>> _ = ti.reset(label='20!').call(math.factorial, 20)
Timed best=...s, mean=...s for 20!
>>> _ = ti.reset().call(math.factorial, 20)
Timed best=...s, mean=...s for 20!
```

**call** (*func, \*args, \*\*kwargs*)

Alternative way to time a simple function call using condensed syntax.

**Returns**

Use *min*, or *mean* to get a scalar. Use *print* to output a report to stdout.

**Return type** self (*Timerit*)

### Example

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

**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.”

**Returns** minimum measured seconds over all trials

**Return type** float

### Example



```
>>> import math
>>> 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.

**Returns** mean of measured seconds

**Return type** float

---

**Note:** This is typically less informative than simply looking at the min. It is recommended to use min as the expectation value rather than mean in most cases.

---

**Example**

```
>>> import math
>>> 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.

**Returns** standard deviation of measured seconds

**Return type** float

---

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

---

**Example**

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

**report(verbose=1)**

Creates a human readable report

**Parameters** **verbose** (*int*) – verbosity level. Either 1, 2, or 3.

**Returns** the report

**Return type** str

**SeeAlso:** Timerit.print

### Example

```
>>> import math
>>> ti = Timerit(num=1).call(math.factorial, 5)
>>> print(ti.report(verbose=1))
Timed best=...s, mean=...s
```

**print** (*verbose=1*)

Prints human readable report using the print function

**Parameters** **verbose** (*int*) – verbosity level

**SeeAlso:** Timerit.report

### Example

```
>>> import math
>>> Timerit(num=10).call(math.factorial, 50).print(verbose=1)
Timed best=...s, mean=...s
>>> Timerit(num=10).call(math.factorial, 50).print(verbose=2)
Timed for: 10 loops, best of 3
    time per loop: best=...s, mean=...s
>>> Timerit(num=10).call(math.factorial, 50).print(verbose=3)
Timed for: 10 loops, best of 3
    body took: ...
    time per loop: best=...s, mean=...s
```

## 1.1.2 Module contents

UBelt is a “utility belt” of commonly needed utility and helper functions. It is a curated collection of utilities with functionality that falls into a mixture of the following categories:

- Timing
- Caching
- Hashing
- Command Line / Shell Interaction
- Cross-Platform Cache, Config, and Data Directories
- Symlinks
- Downloading Files
- Dictionary Histogram
- Find Duplicates
- Dictionary Manipulation
- AutoDict - Autovivification
- String-based imports
- Horizontal String Concatenation
- **Standalone modules.**

- `progiter` for Loop Progress
- `timerit` for Robust Timing and Benchmarking
- `ordered-set` for ordered set collections

For more detailed high level documentation see the README on github: <https://github.com/Erotemic/ubelt>.

The ubelt API is organized by submodules containing related functionality. Each submodule contains top level overview documentation, and each function contains a docstring with at least one example. Please see specific submodule documentation for more details.

**AutogenInit:** `mkinit ubelt -w` # todo: get sphinx to ignore this

**class** `ubelt.AutoDict`

Bases: `dict`

An infinitely nested default dict of dicts.

Implementation of Perl's autovivification feature.

**SeeAlso:** `ub.AutoOrderedDict` - the ordered version

## References

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

## 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`

## 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.AutoOrderedDict`

Bases: `collections.OrderedDict`, `ubelt.util_dict.AutoDict`

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

**SeeAlso:** `ub.AutoDict` - the unordered version

## Example

```
>>> 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]
```

**class** `ubelt.CacheStamp` (*fname, dpath, cfgstr=None, product=None, hasher='sha1', verbose=None*)  
 Bases: `object`

Quickly determine if a file-producing computation has been done.

Writes a file that marks that a procedure has been done by writing a “stamp” file to disk. Removing the stamp file will force recomputation. However, removing or changing the result of the computation may not trigger recomputation unless specific handling is done or the expected “product” of the computation is a file and registered with the stamper. If `hasher` is `None`, we only check if the product exists, and we ignore its hash, otherwise it checks that the hash of the product is the same.

### Parameters

- **fname** (*str*) – Name of the stamp file
- **cfgstr** (*str*) – Configuration associated with the stamped computation. A common pattern is to call `ubelt.hash_data` on a dependency list.
- **dpath** (*PathLike*) – Where to store the cached stamp file
- **product** (*PathLike or Sequence[PathLike], optional*) – Path or paths that we expect the computation to produce. If specified the hash of the paths are stored.
- **hasher** (*str*) – The type of hasher used to compute the file hash of product. If `None`, then we assume the file has not been corrupted or changed. Defaults to `sha1`.
- **verbose** (*bool*) – Passed to internal `ub.Cacher` object

## Example

```
>>> import ubelt as ub
>>> from os.path import join
>>> # Stamp the computation of expensive-to-compute.txt
>>> dpath = ub.ensure_app_cache_dir('ubelt', 'test-cache-stamp')
>>> ub.delete(dpath)
>>> ub.ensuredir(dpath)
>>> product = join(dpath, 'expensive-to-compute.txt')
>>> self = CacheStamp('somedata', cfgstr='someconfig', dpath=dpath,
>>>                    product=product, hasher=None)
>>> self.hasher = None
>>> if self.expired():
>>>     ub.writeto(product, 'very expensive')
>>>     self.renew()
>>> assert not self.expired()
>>> # corrupting the output will not expire in non-robust mode
>>> ub.writeto(product, 'corrupted')
>>> assert not self.expired()
>>> self.hasher = 'sha1'
>>> # but it will expire if we are in robust mode
```

(continues on next page)

(continued from previous page)

```
>>> assert self.expired()
>>> # deleting the product will cause expiration in any mode
>>> self.hasher = None
>>> ub.delete(product)
>>> assert self.expired()
```

**expired** (*cfgstr=None, product=None*)

Check to see if a previously existing stamp is still valid and if the expected result of that computation still exists.

#### Parameters

- **cfgstr** (*str, optional*) – override the default *cfgstr* if specified
- **product** (*PathLike or Sequence[PathLike], optional*) – override the default product if specified

**renew** (*cfgstr=None, product=None*)

Recertify that the product has been recomputed by writing a new certificate to disk.

**class** `ubelt.Cacher` (*fname, cfgstr=None, dpath=None, appname='ubelt', ext='.pkl', meta=None, verbose=None, enabled=True, log=None, hasher='sha1', 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 always 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** (*PathLike*) – 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*) – File extension (default = '.pkl')
- **meta** (*object*) – Metadata that is also saved with the *cfgstr*. This can be useful to indicate how the *cfgstr* was constructed.
- **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`, ...)
- **hasher** (*str*) – Type of hashing algorithm to use if *cfgstr* needs to be condensed to less than 49 characters.
- **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`

### 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
```

### 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**

**FORCE\_DISABLE = False**

**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.

### 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.

**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. Either raise, which re-raises the exception, or clear which deletes the cache and returns None.

**load(cfgstr=None)**

Loads the data

**Raises IOError** - if the data is unable to be loaded. This could be due to – a cache miss or because the cache is disabled.

**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.

### 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

### 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()
```

**class** `ubelt.CaptureStdout` (*supress=True, enabled=True*)

Bases: `ubelt.util_stream.CaptureStream`

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

#### Parameters

- **supress** (*bool, default=True*) – if True, stdout is not printed while captured
- **enabled** (*bool, default=True*) – does nothing if this is False

### Example

```
>>> self = CaptureStdout(supress=True)
>>> print('dont capture the table flip (°° ')
>>> with self:
...     text = 'capture the heart '
```

(continues on next page)



(continued from previous page)

```
...     print(text)
>>> print('dont capture look of disapproval _')
>>> assert isinstance(self.text, six.text_type)
>>> assert self.text == text + '\n', 'failed capture text'
```

### Example

```
>>> self = CaptureStdout(suppress=False)
>>> with self:
...     print('I am captured and printed in stdout')
>>> assert self.text.strip() == 'I am captured and printed in stdout'
```

### Example

```
>>> self = CaptureStdout(suppress=True, enabled=False)
>>> with self:
...     print('dont capture')
>>> assert self.text is None
```

**log\_part()**

Log what has been captured so far

**start()****stop()****Doctest:**

```
>>> CaptureStdout(enabled=False).stop()
>>> CaptureStdout(enabled=True).stop()
```

**close()****class ubelt.CaptureStream**

Bases: object

Generic class for capturing streaming output from stdout or stderr

**class ubelt.FormatterExtensions**

Bases: object

Helper class for managing non-builtin (e.g. numpy) format types.

This module (*ubelt.util\_format*) maintains a global set of basic extensions, but it is also possible to create a locally scoped set of extensions and explicitly pass it to repr2. The following example demonstrates this.

### Example

```
>>> import ubelt as ub
>>> class MyObject(object):
>>>     pass
>>> data = {'a': [1, 2.2222, MyObject()], 'b': MyObject()}
>>> # Create a custom set of extensions
>>> extensions = ub.FormatterExtensions()
```

(continues on next page)

(continued from previous page)

```
>>> # Register a function to format your specific type
>>> @extensions.register(MyObject)
>>> def format_myobject(data, **kwargs):
>>>     return 'I can do anything here'
>>> # Repr2 will now respect the passed custom extensions
>>> # Note that the global extensions will still be respected
>>> # unless they are overloaded.
>>> print(ub.repr2(data, nl=-1, precision=1, extensions=extensions))
{
    'a': [1, 2.2, I can do anything here],
    'b': I can do anything here
}
>>> # Overload the formatter for float and int
>>> @extensions.register((float, int))
>>> def format_myobject(data, **kwargs):
>>>     return str((data + 10) // 2)
>>> print(ub.repr2(data, nl=-1, precision=1, extensions=extensions))
{
    'a': [5, 6.0, I can do anything here],
    'b': I can do anything here
}
```

**register** (*type*)

Registers a custom formatting function with `ub.repr2`

**lookup** (*data*)

Returns an appropriate function to format *data* if one has been registered.

**class** `ubelt.NiceRepr`

Bases: `object`

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

## Example

```
>>> import ubelt as ub
>>> class Foo(ub.NiceRepr):
...     def __nice__(self):
...         return 'info'
>>> foo = Foo()
>>> assert str(foo) == '<Foo(info)>'
>>> assert repr(foo).startswith('<Foo(info) at ')

```

## Example

```
>>> import ubelt as ub
>>> class Bar(ub.NiceRepr):
...     pass
>>> bar = Bar()
>>> import pytest
>>> with pytest.warns(None) as record:
>>>     assert 'object at' in str(bar)
>>>     assert 'object at' in repr(bar)

```

### Example

```
>>> import ubelt as ub
>>> class Baz(ub.NiceRepr):
...     def __len__(self):
...         return 5
>>> baz = Baz()
>>> assert str(baz) == '<Baz(5)>'
```

**class** `ubelt.OrderedSet` (*iterable=None*)

Bases: `collections.abc.MutableSet`, `collections.abc.Sequence`

An `OrderedSet` is a custom `MutableSet` that remembers its order, so that every entry has an index that can be looked up.

### Example

```
>>> OrderedSet([1, 1, 2, 3, 2])
OrderedSet([1, 2, 3])
```

**copy()**

Return a shallow copy of this object.

### Example

```
>>> this = OrderedSet([1, 2, 3])
>>> other = this.copy()
>>> this == other
True
>>> this is other
False
```

**add** (*key*)

Add *key* as an item to this `OrderedSet`, then return its index.

If *key* is already in the `OrderedSet`, return the index it already had.

### Example

```
>>> oset = OrderedSet()
>>> oset.append(3)
0
>>> print(oset)
OrderedSet([3])
```

**append** (*key*)

Add *key* as an item to this `OrderedSet`, then return its index.

If *key* is already in the `OrderedSet`, return the index it already had.

### Example

```
>>> oset = OrderedSet()
>>> oset.append(3)
0
>>> print(oset)
OrderedSet([3])
```

### **update** (*sequence*)

Update the set with the given iterable sequence, then return the index of the last element inserted.

### Example

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

### **index** (*key*)

Get the index of a given entry, raising an `IndexError` if it's not present.

*key* can be an iterable of entries that is not a string, in which case this returns a list of indices.

### Example

```
>>> oset = OrderedSet([1, 2, 3])
>>> oset.index(2)
1
```

### **get\_loc** (*key*)

Get the index of a given entry, raising an `IndexError` if it's not present.

*key* can be an iterable of entries that is not a string, in which case this returns a list of indices.

### Example

```
>>> oset = OrderedSet([1, 2, 3])
>>> oset.index(2)
1
```

### **get\_indexer** (*key*)

Get the index of a given entry, raising an `IndexError` if it's not present.

*key* can be an iterable of entries that is not a string, in which case this returns a list of indices.

### Example

```
>>> oset = OrderedSet([1, 2, 3])
>>> oset.index(2)
1
```

**pop()**

Remove and return the last element from the set.

Raises `KeyError` if the set is empty.

**Example**

```
>>> oset = OrderedSet([1, 2, 3])
>>> oset.pop()
3
```

**discard(key)**

Remove an element. Do not raise an exception if absent.

The `MutableSet` mixin uses this to implement the `.remove()` method, which *does* raise an error when asked to remove a non-existent item.

**Example**

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

**clear()**

Remove all items from this `OrderedSet`.

**union(\*sets)**

Combines all unique items. Each items order is defined by its first appearance.

**Example**

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

**Example**

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

(continues on next page)

(continued from previous page)

```
OrderedSet([2])
>>> oset.intersection()
OrderedSet([1, 2, 3])
```

**difference** (\*sets)

Returns all elements that are in this set but not the others.

**Example**

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

**issubset** (other)

Report whether another set contains this set.

**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.

**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 OrderedSets as a new set. That is, the new set will contain all elements that are in exactly one of the sets.

Their order will be preserved, with elements from *self* preceding elements from *other*.

### Example

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

#### **difference\_update** (\*sets)

Update this OrderedSet to remove items from one or more other sets.

### Example

```
>>> this = OrderedSet([1, 2, 3])
>>> this.difference_update(OrderedSet([2, 4]))
>>> print(this)
OrderedSet([1, 3])
```

```
>>> this = OrderedSet([1, 2, 3, 4, 5])
>>> this.difference_update(OrderedSet([2, 4]), OrderedSet([1, 4, 6]))
>>> print(this)
OrderedSet([3, 5])
```

#### **intersection\_update** (other)

Update this OrderedSet to keep only items in another set, preserving their order in this set.

### Example

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

#### **symmetric\_difference\_update** (other)

Update this OrderedSet to remove items from another set, then add items from the other set that were not present in this set.

### Example

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

```
class ubelt.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, en-
                      abled=True, verbose=None, stream=None, chunksize=None, **kwargs)
Bases: progiter.progiter._TQDMCompat, progiter.progiter._BackwardsCompat
```

Prints progress as an iterator progresses

#### 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.
- **chunksize**(*int*) – indicates that each iteration processes a batch of this size. Iteration rate is displayed in terms of single-items.
- **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

---

**Note:** 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.

---



---

**Note:** ProgIter is an alternative to *tqdm*. 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 *ProgIter* API will change to become inter-compatible with *tqdm*.

---

**SeeAlso:** *tqdm* - <https://pypi.python.org/pypi/tqdm>

**Reference:** <http://datagenetics.com/blog/february12017/index.html>

## Example

```
>>> # doctest: +SKIP
>>> import progiter
>>> def is_prime(n):
...     return n >= 2 and not any(n % i == 0 for i in range(2, n))
>>> for n in progiter.ProgIter(range(100), verbose=1):
>>>     # do some work
```

(continues on next page)



(continued from previous page)

```
>>> 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:**

- [ ] *extra* is a bad name; come up with something better and rename

**Example**

```
>>> import progiter
>>> prog = progiter.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 precedence over *inc*
- **inc** (*int*) – number of steps to increment (defaults to 1)

**Example**

```
>>> import progiter
>>> n = 3
>>> prog = progiter.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()
```

**Example**

```
>>> import progiter
>>> n = 3
>>> # can be used as a context manager in manual mode
>>> with progiter.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.

### 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'
```

### Example

```
>>> self = ProgIter(chunksize=10, total=100, clearline=False,
>>>                  show_times=False, microseconds=True)
>>> # hack, microseconds=True for coverage, needs real test
>>> print(repr(self.format_message()))
' 0.00% of 10x100... \n'
>>> self.begin()
>>> self.update() # tqdm alternative to step
>>> print(repr(self.format_message()))
' 1.00% of 10x100... \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

### Example

```
>>> # Unsafe version may write your message on the wrong line
>>> import progiter
>>> prog = progiter.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 = progiter.ProgIter(range(4), show_times=False, verbose=1)
>>> for n in prog:
...     prog.ensure_newline()
...     print('safe message')
0/4...
safe message
```

(continues on next page)

(continued from previous page)

```
1/4...
safe message
safe message
safe message
4/4...
```

**display\_message()**

Writes current progress to the output stream

**class** `ubelt.TeeStringIO` (*redirect=None*)

Bases: `_io.StringIO`

An IO object that writes to itself and another IO stream.

**Variables** `redirect` (*io.IOWrapper*) – The other stream to write to.

### Example

```
>>> redirect = io.StringIO()
>>> self = TeeStringIO(redirect)
```

**isatty()**

Returns true if the redirect is a terminal.

### Notes

Needed for IPython.embed to work properly when this class is used to override stdout / stderr.

**encoding**

Gets the encoding of the *redirect* IO object

**Doctest:**

```
>>> redirect = io.StringIO()
>>> assert TeeStringIO(redirect).encoding is None
>>> assert TeeStringIO(None).encoding is None
>>> assert TeeStringIO(sys.stdout).encoding is sys.stdout.encoding
>>> redirect = io.TextIOWrapper(io.StringIO())
>>> assert TeeStringIO(redirect).encoding is redirect.encoding
```

**write(msg)**

Write to this and the redirected stream

**flush()**

Flush to this and the redirected stream

**class** `ubelt.TempDir`

Bases: `object`

Context for creating and cleaning up temporary directories.

### Example

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

### Example

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

**ensure()**

**cleanup()**

**start()**

**class** ubelt.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, defaults to True

#### Variables

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

**CommandLine:** python -m timerit.core Timer

### Example

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

## Example

```
>>> # Create and start the timer using the tic/toc interface
>>> from timerit import Timer
>>> 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.Timerit (num=1, label=None, bestof=3, unit=None, 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 timerit.core Timerit:0

## Example

```
>>> from timerit import Timerit
>>> import math
>>> num = 15
>>> t1 = Timerit(num, label='factorial', verbose=1)
>>> 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)
Timed best=..., mean=... for factorial
>>> # <you can now access Timerit attributes>
>>> assert t1.total_time > 0
>>> assert t1.n_loops == t1.num
>>> assert t1.n_loops == num
```

## Example

```
>>> # xdoc: +IGNORE_WANT
>>> import math
>>> from timerit import Timerit
>>> num = 10
>>> # If the timer object is unused, time will still be recorded,
>>> # but with less precision.
>>> for _ in Timerit(num, 'concise', verbose=2):
>>>     math.factorial(10000)
Timed concise for: 10 loops, best of 3
    time per loop: best=4.954 ms, mean=4.972 ± 0.018 ms
>>> # Using the timer object results in the most precise timings
>>> for timer in Timerit(num, 'precise', verbose=3):
>>>     with timer: math.factorial(10000)
Timing precise for: 15 loops, best of 3
Timed precise for: 15 loops, best of 3
    time per loop: best=2.474 ms, mean=2.54 ± 0.046 ms
```

**reset** (*label=None*)

clears all measurements, allowing the object to be reused

**Parameters** *label* (*str; optional*) – optionally change the label

### Example

```
>>> from timerit import Timerit
>>> import math
>>> ti = Timerit(num=10, unit='us', verbose=True)
>>> _ = ti.reset(label='10!').call(math.factorial, 10)
Timed best=...s, mean=...s for 10!
>>> _ = ti.reset(label='20!').call(math.factorial, 20)
Timed best=...s, mean=...s for 20!
>>> _ = ti.reset().call(math.factorial, 20)
Timed best=...s, mean=...s for 20!
```

**call** (*func, \*args, \*\*kwargs*)

Alternative way to time a simple function call using condensed syntax.

### Returns

Use *min*, or *mean* to get a scalar. Use *print* to output a report to stdout.

**Return type** *self* (*Timerit*)

### Example

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

**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. ““

**Returns** minimum measured seconds over all trials

**Return type** float

### Example

```
>>> import math
>>> 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.

**Returns** mean of measured seconds

**Return type** float

---

**Note:** This is typically less informative than simply looking at the min. It is recommended to use min as the expectation value rather than mean in most cases.

---

### Example

```
>>> import math
>>> 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.

**Returns** standard deviation of measured seconds

**Return type** float

---

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

---

### Example

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

#### `report(verbose=1)`

Creates a human readable report

**Parameters** `verbose` (*int*) – verbosity level. Either 1, 2, or 3.

**Returns** the report

**Return type** str

**SeeAlso:** Timerit.print

### Example

```
>>> import math
>>> ti = Timerit(num=1).call(math.factorial, 5)
>>> print(ti.report(verbose=1))
Timed best=...s, mean=...s
```

**print** (*verbose=1*)

Prints human readable report using the print function

**Parameters** *verbose* (*int*) – verbosity level

**SeeAlso:** Timerit.report

### Example

```
>>> import math
>>> Timerit(num=10).call(math.factorial, 50).print(verbose=1)
Timed best=...s, mean=...s
>>> Timerit(num=10).call(math.factorial, 50).print(verbose=2)
Timed for: 10 loops, best of 3
    time per loop: best=...s, mean=...s
>>> Timerit(num=10).call(math.factorial, 50).print(verbose=3)
Timed for: 10 loops, best of 3
    body took: ...
    time per loop: best=...s, mean=...s
```

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

Determine if all items in a sequence are the same

**Parameters**

- **iterable** (*Iterable*) – items to determine if they are all the same
- **eq** (*Callable*, *optional*) – function to determine equality (default: operator.eq)

### 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))
```

(continues on next page)



(continued from previous page)

```
False
>>> allsame(range(10), lambda a, b: True)
True
```

`ubelt.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** (*Optional[list]*) – overrides `sys.argv` if specified

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

**Return type** `bool`

#### Example

```
>>> 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
```

`ubelt.argmax(indexable, key=None)`

Returns index / key of the item with the largest value.

This is similar to `numpy.argmax`, but it is written in pure python and works on both lists and dictionaries.

#### Parameters

- **indexable** (*Iterable or Mapping*) – indexable to sort by
- **key** (*Callable, optional*) – customizes the ordering of the indexable

**CommandLine:** `python -m ubelt.util_list argmax`

#### 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
>>> assert argmax(iter(['a', 'c', 'b', 'z', 'f'])) == 3
```

`ubelt.argmin(indexable, key=None)`

Returns index / key of the item with the smallest value.

This is similar to `numpy.argmin`, but it is written in pure python and works on both lists and dictionaries.

#### Parameters

- **indexable** (*Iterable or Mapping*) – indexable to sort by
- **key** (*Callable, optional*) – customizes the ordering of the indexable

### 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'
>>> assert argmin(iter(['a', 'c', 'A', 'z', 'f'])) == 2
```

`ubelt.argmaxsort` (*indexable*, *key=None*, *reverse=False*)  
Returns the indices that would sort a indexable object.

This is similar to `numpy.argsort`, but it is written in pure python and works on both lists and dictionaries.

#### Parameters

- **indexable** (*Iterable or Mapping*) – indexable to sort by
- **key** (*Callable, 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

### 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.argunique` (*items*, *key=None*)  
Returns indices corresponding to the first instance of each unique item.

#### Parameters

- **items** (*Sequence*) – indexable collection of items
- **key** (*Callable, optional*) – custom normalization function. If specified returns items where `key(item)` is unique.

**Yields** *int* – indices of the unique items

### 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.argval(key, default=None, 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** (*Optional[object]*) – value to return if not specified
- **argv** (*Optional[list]*) – overrides `sys.argv` if specified

#### Returns

**value** [the value specified after the key. If the key is specified multiple times, then the first value is returned.]

**Return type** `str`

#### Todo:

- [ ] Can we handle the case where the value is a list of long paths?
- [ ] Should we default the first or last specified instance of the flag.

### Example

```
>>> 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
```

### Example

```
>>> # Test fix for GH Issue #41
>>> import ubelt as ub
>>> argv = ['--path=/path/with/k=3']
>>> ub.argval('--path', argv=argv) == '/path/with/k=3'
```

`ubelt.augpath(path, suffix="", prefix="", ext=None, base=None, multidot=False)`

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** (*PathLike*) – string representation of a path
- **suffix** (*str*) – placed between the basename and extension
- **prefix** (*str*) – placed in front of the basename
- **ext** (*str*) – if specified, replaces the extension
- **base** (*str*) – if specified, replaces the basename (without extension)
- **multidot** (*bool*) – if False, everything after the last dot in the basename is the extension. If True, everything after the first dot in the basename is the extension (Defaults to False).

**Returns** augmented path

**Return type** PathLike

**CommandLine:** `python -m ubelt.util_path augpath`

### 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
```

### 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'
>>> augpath('foo.tar.gz', ext='.zip', multidot=True)
foo.zip
>>> augpath('foo.tar.gz', ext='.zip', multidot=False)
foo.tar.zip
```

`ubelt.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 inferred from *indices*

---

**Note:** In the future the arg *maxval* may change its name to *shape*

---

**Returns** mask: list of booleans. mask[idx] is True if idx in indices

**Return type** list

### 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]
```

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

Bases: object

Generates successive n-sized chunks from *items*.

If the last chunk has less than n elements, *bordermode* is used to determine fill values.

#### Parameters

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

---

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

---

### References

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

**CommandLine:** `python -m ubelt.util_list chunks`

### Example

```
>>> import ubelt as ub
>>> items = [1, 2, 3, 4, 5, 6, 7]
>>> genresult = ub.chunks(items, chunksize=3, bordermode='none')
>>> assert list(genresult) == [[1, 2, 3], [4, 5, 6], [7]]
>>> genresult = ub.chunks(items, chunksize=3, bordermode='cycle')
>>> assert list(genresult) == [[1, 2, 3], [4, 5, 6], [7, 1, 2]]
>>> genresult = ub.chunks(items, 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
>>> # Note: ub.chunks will not do the 2,1,1 split
>>> assert len(list(ub.chunks(range(4), nchunks=3))) == 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** (*items, chunksize*)

**static cycle** (*items, chunksize*)

**static replicate** (*items, chunksize*)

`ubelt.cmd` (*command, shell=False, detach=False, verbose=0, tee=None, cwd=None, env=None, tee\_backend='auto', verbout=None, \*\*kwargs*)

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 detach, return the process object and allow the process to run in the background (eventually we may return a Future object instead).

**Parameters**

- **command** (*str or Sequence*) – bash-like command string or tuple of executable and args
- **shell** (*bool*) – if True, process is run in shell, defaults to False.
- **detach** (*bool*) – if True, process is detached and run in background, defaults to False.
- **verbose** (*int*) – verbosity mode. Can be 0, 1, 2, or 3. Defaults to 0.
- **tee** (*bool, optional*) – if True, simultaneously writes to stdout while capturing output from the command. If not specified, defaults to True if `verbose > 0`. If `detach` is True, then this argument is ignored.
- **cwd** (*PathLike, optional*) – path to run command
- **env** (*str, optional*) – environment passed to Popen
- **tee\_backend** (*str, optional*) – backend for tee output. Valid choices are: “auto”, “select” (POSIX only), and “thread”.
- **\*\*kwargs** – only used to support deprecated arguments

**Returns**

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

**Return type** dict

**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)”

**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>

**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' + chr(10) + 'shell'
```

**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), detach=True)
>>> info2 = ub.cmd('echo writing2 > ' + fpath2, shell=True, detach=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()
```

`ubelt.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`

### 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.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, fuchsia

**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

**Example**

```
>>> text = 'raw text'
>>> import pytest
>>> import ubelt as ub
>>> if ub.modname_to_modpath('pygments'):
>>>     # Colors text only if pygments is installed
>>>     assert color_text(text, 'red') == '\x1b[31;01mraw text\x1b[39;49;00m'
>>>     assert color_text(text, None) == 'raw text'
>>> else:
>>>     # Otherwise text passes through unchanged
>>>     assert color_text(text, 'red') == 'raw text'
>>>     assert color_text(text, None) == 'raw text'
```

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

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

**Parameters**

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

**Returns** a subset of masked items

**Return type** Iterable

**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.compressuser` (*path, home='~'*)

Inverse of `os.path.expanduser`

**Parameters**

- **path** (*PathLike*) – 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** PathLike

**CommandLine:** `xdoctest -m ubelt.util_path compressuser`

### 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.ddict`

alias of `collections.defaultdict`

`ubelt.delete(path, verbose=False)`

Removes a file or recursively removes a directory. If a path does not exist, then this does nothing.

#### Parameters

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

#### SeeAlso:

**send2trash** - A cross-platform Python package for sending files to the trash instead of irreversibly deleting them. <https://github.com/hsoft/send2trash>

#### Doctest:

```
>>> import ubelt as ub
>>> 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)))
```

#### Doctest:

```
>>> import ubelt as ub
>>> dpath = ub.ensure_app_cache_dir('ubelt', 'delete_test2')
>>> dpath1 = ub.ensuredir(join(dpath, 'dir'))
>>> fpath1 = ub.touch(join(dpath1, 'to_remove.txt'))
>>> assert exists(fpath1)
>>> ub.delete(dpath)
>>> assert not exists(fpath1)
```

`ubelt.dict_hist` (*item\_list*, *weight\_list=None*, *ordered=False*, *labels=None*)

Builds a histogram of items, counting the number of time each item appears in the input.

#### Parameters

- **item\_list** (*Iterable*) – hashable items (usually containing duplicates)
- **weight\_list** (*Iterable*) – corresponding weights for each item
- **ordered** (*bool*) – if True the result is ordered by frequency
- **labels** (*Iterable, optional*) – expected labels (default None) Allows this function to pre-initialize the histogram. 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

#### 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}
```

#### 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.dict_isect` (\*args)

Constructs a dictionary that contains keys common between all inputs. The returned values will only belong to the first dictionary.

**Parameters** \*args – a sequence of dictionaries (or sets of keys)

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

**Return type** Dict | OrderedDict

## Notes

This function can be used as an alternative to `dict_subset` where any key not in the dictionary is ignored. See the following example:

```
>>> dict_isect({'a': 1, 'b': 2, 'c': 3}, ['a', 'c', 'd'])
{'a': 1, 'c': 3}
```

## Example

```
>>> dict_isect({'a': 1, 'b': 1}, {'b': 2, 'c': 2})
{'b': 1}
>>> dict_isect(odict([('a', 1), ('b', 2)]), odict([('c', 3)]))
OrderedDict()
>>> dict_isect()
{}
```

`ubelt.dict_subset(dict_, keys, default=NoParam)`

Get a subset of a dictionary

### Parameters

- **dict\_** (*Mapping*) – superset dictionary
- **keys** (*Iterable*) – keys to take from `dict_`
- **default** (*object, optional*) – if specified uses default if keys are missing

**Returns** subset dictionary

**Return type** OrderedDict

**SeeAlso:** `dict_isect` - similar functionality, but will only take existing keys

## 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.dict_take(dict_, keys, default=NoParam)`

Generates values from a dictionary

### Parameters

- **dict\_** (*Mapping*) – a dictionary to take from
- **keys** (*Iterable*) – the keys to take
- **default** (*object, optional*) – if specified uses default if keys are missing

**CommandLine:** `python -m ubelt.util_dict dict_take_gen`

### 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]
```

### 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.dict_union(*args)`

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

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

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

**Return type** `Dict | OrderedDict`

**SeeAlso:**

**`collections.ChainMap`** - a standard python builtin data structure that provides a view that treats multiple dicts as a single dict. <https://docs.python.org/3/library/collections.html#chainmap-objects>

### 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.download(url, fpath=None, hash_prefix=None, hasher='sha512', chunksize=8192, verbose=1)`

downloads a url to a fpath.

**Parameters**

- **url** (*str*) – The url to download.
- **fpath** (*PathLike | io.BytesIO | tringIO*) – The path to download to. Defaults to basename of url and ubelt's application cache. If this is a `io.BytesIO` object then information is directly written to this object (note this prevents the use of temporary files).

- **hash\_prefix** (*None or str*) – If specified, download will retry / error if the file hash does not match this value. Defaults to None.
- **hasher** (*str or Hasher*) – If hash\_prefix is specified, this indicates the hashing algorithm to apply to the file. Defaults to sha512.
- **chunksize** (*int*) – Download chunksize. Defaults to  $2^{13}$ .
- **verbose** (*int*) – Verbosity level 0 or 1. Defaults to 1.

**Returns** fpath - file path string

**Return type** PathLike

**Raises**

- `URLError` - if there is problem downloading the url
- `RuntimeError` - if the hash does not match the hash\_prefix

## Notes

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

## References

<http://blog.moleculea.com/2012/10/04/urlretrieve-progres-indicator/questions/15644964/python-progress-bar-and-downloads> <http://stackoverflow.com/questions/16694907/how-to-download-large-file-in-python-with-requests-py>

**CommandLine:** `python -m xdoctest ubelt.util_download download:1`

## Example

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

## Example

```
>>> # xdoctest: +REQUIRES(--network)
>>> import ubelt as ub
>>> import io
>>> url = 'http://i.imgur.com/rqwaDag.png'
>>> file = io.BytesIO()
>>> fpath = download(url, file)
>>> file.seek(0)
>>> data = file.read()
>>> assert ub.hash_data(data, hasher='sha1').startswith('f79ea24571')
```

### Example

```
>>> # xdoctest: +REQUIRES(--network)
>>> url = 'http://i.imgur.com/rqwaDag.png'
>>> fpath = download(url, hasher='sha1', hash_prefix=
↳ 'f79ea24571da6ddd2ba12e3d57b515249ecb8a35')
Downloading url='http://i.imgur.com/rqwaDag.png' to fpath=...rqwaDag.png
...
...1233/1233... rate=... Hz, eta=..., total=..., wall=...
```

### Example

```
>>> # xdoctest: +REQUIRES(--network)
>>> # test download from girder
>>> import pytest
>>> import ubelt as ub
>>> url = 'https://data.kitware.com/api/v1/item/5b4039308d777f2e6225994c/download'
>>> ub.download(url, hasher='sha512', hash_prefix='c98a46cb31205cf')
>>> with pytest.raises(RuntimeError):
>>>     ub.download(url, hasher='sha512', hash_prefix='BAD_HASH')
```

`ubelt.dzip(items1, items2, cls=<class 'dict'>)`

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

#### Parameters

- **items1** (*Iterable*) – full sequence
- **items2** (*Iterable*) – can either be a sequence of one item or a sequence of equal length to `items1`
- **cls** (*Type[dict]*) – dictionary type to use. Defaults to `dict`, but could be `ordered dict` instead.

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

**Return type** `dict`

### 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.editfile(fpath, verbose=True)`

DEPRICATED: This has been ported to `xdev`, please use that version.

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** (*PathLike*) – 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)
```

`ubelt.ensure_app_cache_dir(appname, *args)`

Calls `get_app_cache_dir` but ensures the directory exists.

**Parameters**

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

**SeeAlso:** `get_app_cache_dir`

**Example**

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

`ubelt.ensure_app_config_dir(appname, *args)`

Calls `get_app_config_dir` but ensures the directory exists.

**Parameters**

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

**SeeAlso:** `get_app_config_dir`

**Example**

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

`ubelt.ensure_app_data_dir(appname, *args)`

Calls `get_app_data_dir` but ensures the directory exists.

**Parameters**

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

**SeeAlso:** `get_app_data_dir`



## Example

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

`ubelt.ensure_app_resource_dir(appname, *args)`

Calls `get_app_resource_dir` but ensures the directory exists.

DEPRECATED in favor of `ensure_app_config_dir` / `ensure_app_data_dir`

### Parameters

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

**SeeAlso:** `get_app_resource_dir`

`ubelt.ensure_unicode(text)`

Casts bytes into utf8 (mostly for python2 compatibility)

## References

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

## Example

```
>>> from ubelt.util_str import *
>>> import codecs # NOQA
>>> assert ensure_unicode('my unicôdé string') == 'my unicôdé string'
>>> assert ensure_unicode('text1') == 'text1'
>>> assert ensure_unicode('text1'.encode('utf8')) == 'text1'
>>> assert ensure_unicode('i»¿text1'.encode('utf8')) == 'i»¿text1'
>>> assert (codecs.BOM_UTF8 + 'text»¿'.encode('utf8')).decode('utf8')
```

`ubelt.ensure_dir(dpath, mode=1023, verbose=None)`

Ensures that directory will exist. Creates new dir with sticky bits by default

### Parameters

- **dpath** (*PathLike*) – 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** PathLike

## Notes

This function is not thread-safe in Python2

### 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)
```

`ubelt.expandpath(path)`

Wrapper around `expanduser` and `expandvars`.

Less aggressive than `truepath`. Only expands environs and tilde. Does not change relative paths to absolute paths.

**Parameters** `path` (*PathLike*) – string representation of a path

**Returns** expanded path

**Return type** `PathLike`

### Example

```
>>> import ubelt as ub
>>> assert normpath(ub.expandpath('~foo')) == join(ub.userhome(), 'foo')
>>> assert ub.expandpath('foo') == 'foo'
```

`ubelt.find_duplicates(items, k=2, key=None)`

Find all duplicate items in a list.

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

**Parameters**

- **items** (*Iterable*) – hashable items possibly containing duplicates
- **k** (*int*) – only return items that appear at least *k* times (default=2)
- **key** (*Callable, optional*) – Returns indices where `key(items[i])` maps to a particular value at least *k* times.

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

**Return type** `dict`

**CommandLine:** `python -m ubelt.util_dict find_duplicates`

### 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,))
```

(continues on next page)

(continued from previous page)

```
>>> 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]}
```

### 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]}
```

### Example

```
>>> import ubelt as ub
>>> items = [10, 11, 12, 13, 14, 15, 16]
>>> duplicates = ub.find_duplicates(items, key=lambda x: x // 2)
>>> print(ub.repr2(duplicates, nl=0))
{5: [0, 1], 6: [2, 3], 7: [4, 5]}
```

`ubelt.find_exe(name, multi=False, path=None)`

Locate a command.

Search your local filesystem for an executable and return the first matching file with executable permission.

#### Parameters

- **name** (*str*) – globstr of matching filename
- **multi** (*bool*) – if True return all matches instead of just the first. Defaults to False.
- **path** (*str or Iterable[PathLike]*) – overrides the system PATH variable.

**Returns** returns matching executable(s).

**Return type** PathLike or List[PathLike] or None

**SeeAlso:** `shutil.which` - which is available in Python 3.3+.

### Notes

This is essentially the *which* UNIX command

### References

<https://stackoverflow.com/questions/377017/test-if-executable-exists-in-python/377028#377028>  
<https://docs.python.org/dev/library/shutil.html#shutil.which>

<https://docs.python.org/dev/library/shutil.html#shutil.which>

### Example

```
>>> find_exe('ls')
>>> find_exe('ping')
>>> assert find_exe('which') == find_exe(find_exe('which'))
>>> find_exe('which', multi=True)
>>> find_exe('ping', multi=True)
>>> find_exe('cmake', multi=True)
>>> find_exe('nvcc', multi=True)
>>> find_exe('noexist', multi=True)
```

### Example

```
>>> assert not find_exe('noexist', multi=False)
>>> assert find_exe('ping', multi=False)
>>> assert not find_exe('noexist', multi=True)
>>> assert find_exe('ping', multi=True)
```

### Benchmark:

```
>>> # xdoctest: +IGNORE_WANT
>>> import ubelt as ub
>>> import shutil
>>> for timer in ub.Timerit(100, bestof=10, label='ub.find_exe'):
>>>     ub.find_exe('which')
>>> for timer in ub.Timerit(100, bestof=10, label='shutil.which'):
>>>     shutil.which('which')
Timed best=58.71  $\mu$ s, mean=59.64  $\pm$  0.96  $\mu$ s for ub.find_exe
Timed best=72.75  $\mu$ s, mean=73.07  $\pm$  0.22  $\mu$ s for shutil.which
```

`ubelt.find_path(name, path=None, exact=False)`

Search for a file or directory on your local filesystem by name (file must be in a directory specified in a PATH environment variable)

#### Parameters

- **fname** (*PathLike* or *str*) – file name to match. If `exact` is `False` this may be a glob pattern
- **path** (*str* or *Iterable[PathLike]*) – list of directories to search either specified as an `os.pathsep` separated string or a list of directories. Defaults to environment PATH.
- **exact** (*bool*) – if `True`, only returns exact matches. Default `False`.

### Notes

For recursive behavior set `path=(d for d, _, _ in os.walk('.'))`, where `'.'` might be replaced by the root directory of interest.

### Example

```
>>> list(find_path('ping', exact=True))
>>> list(find_path('bin'))
>>> list(find_path('bin'))
```

(continues on next page)

(continued from previous page)

```
>>> list(find_path('*cc*'))
>>> list(find_path('cmake*'))
```

### Example

```
>>> import ubelt as ub
>>> from os.path import dirname
>>> path = dirname(dirname(ub.util_platform.__file__))
>>> res = sorted(find_path('ubelt/util_*.py', path=path))
>>> assert len(res) >= 10
>>> res = sorted(find_path('ubelt/util_platform.py', path=path, exact=True))
>>> print(res)
>>> assert len(res) == 1
```

`ubelt.flatten(nested_list)`

Transforms a nested iterable into a flat iterable.

This is simply an alias for `itertools.chain.from_iterable`

**Parameters** `nested_list` (*Iterable[Iterable]*) – list of lists

**Returns** flattened items

**Return type** `Iterable`

### Example

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

`ubelt.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** `PathLike`

**SeeAlso:** `ensure_app_cache_dir`

`ubelt.get_app_config_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 config directory for this application

**Return type** PathLike

**SeeAlso:** `ensure_app_config_dir`

`ubelt.get_app_data_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 data directory for this application

**Return type** PathLike

**SeeAlso:** `ensure_app_data_dir`

`ubelt.get_app_resource_dir(appname, *args)`

Returns a writable directory for an application This should be used for persistent configuration files.

DEPRICATED in favor of `get_app_config_dir` / `get_app_data_dir`

**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** PathLike

**SeeAlso:** `ensure_app_resource_dir`

`ubelt.grabdata(url, fpath=None, dpath=None, fname=None, redo=False, verbose=1, appname=None, hash_prefix=None, hasher='sha512', **download_kw)`

Downloads a file, caches it, and returns its local path.

**Parameters**

- **url** (*str*) – url to the file to download
- **fpath** (*PathLike*) – The full path to download the file to. If unspecified, the arguments *dpath* and *fname* are used to determine this.
- **dpath** (*PathLike*) – 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*.
- **hash\_prefix** (*None or str*) – If specified, `grabdata` verifies that this matches the hash of the file, and then saves the hash in a adjacent file to certify that the download was successful. Defaults to *None*.

- **hasher** (*str or Hasher*) – If `hash_prefix` is specified, this indicates the hashing algorithm to apply to the file. Defaults to `sha512`.
- **\*\*download\_kw** – additional kwargs to pass to `ub.download`

**Returns** `fpath` - file path string

**Return type** `PathLike`

### Example

```
>>> # xdoctest: +REQUIRES(--network)
>>> import ubelt as ub
>>> url = 'http://i.imgur.com/rqwaDag.png'
>>> fpath = ub.grabdata(url, fname='mario.png')
>>> result = basename(fpath)
>>> print(result)
mario.png
```

### Example

```
>>> # xdoctest: +REQUIRES(--network)
>>> import ubelt as ub
>>> fname = 'foo.bar'
>>> url = 'http://i.imgur.com/rqwaDag.png'
>>> prefix1 = '944389a39dfb8fa9'
>>> fpath = ub.grabdata(url, fname=fname, hash_prefix=prefix1)
>>> stamp_fpath = fpath + '.hash'
>>> assert open(stamp_fpath, 'r').read() == prefix1
>>> # Check that the download doesn't happen again
>>> fpath = ub.grabdata(url, fname=fname, hash_prefix=prefix1)
>>> # todo: check file timestamps have not changed
>>> #
>>> # Check redo works with hash
>>> fpath = ub.grabdata(url, fname=fname, hash_prefix=prefix1, redo=True)
>>> # todo: check file timestamps have changed
>>> #
>>> # Check that a redownload occurs when the stamp is changed
>>> open(stamp_fpath, 'w').write('corrupt-stamp')
>>> fpath = ub.grabdata(url, fname=fname, hash_prefix=prefix1)
>>> assert open(stamp_fpath, 'r').read() == prefix1
>>> #
>>> # Check that a redownload occurs when the stamp is removed
>>> ub.delete(stamp_fpath)
>>> open(fpath, 'w').write('corrupt-data')
>>> assert not ub.hash_file(fpath, base='hex', hasher='sha512').
↳startswith(prefix1)
>>> fpath = ub.grabdata(url, fname=fname, hash_prefix=prefix1)
>>> assert ub.hash_file(fpath, base='hex', hasher='sha512').startswith(prefix1)
>>> #
>>> # Check that requesting new data causes redownload
>>> url2 = 'https://data.kitware.com/api/v1/item/5b4039308d777f2e6225994c/download
↳'
>>> prefix2 = 'c98a46cb31205cf'
>>> fpath = ub.grabdata(url2, fname=fname, hash_prefix=prefix2)
>>> assert open(stamp_fpath, 'r').read() == prefix2
```

`ubelt.group_items(items, groupids)`

Groups a list of items by group id.

**Parameters**

- **items** (*Iterable*) – a list of items to group
- **groupids** (*Iterable or Callable*) – a corresponding list of item groupids or a function mapping an item to a groupid.

**Returns** groupid\_to\_items: maps a groupid to a list of items

**Return type** dict

**CommandLine:** `python -m ubelt.util_dict group_items`

**Example**

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

`ubelt.hash_data(data, hasher=NoParam, base=NoParam, types=False, hashlen=NoParam, convert=False)`

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

**Parameters**

- **data** (*object*) – Any sort of loosely organized data
- **hasher** (*str or HASHER*) – Hash algorithm from hashlib, defaults to `sha512`.
- **base** (*str or List[str]*) – Shorthand key or a list of symbols. Valid keys are: ‘abc’, ‘hex’, and ‘dec’. Defaults to ‘hex’.
- **types** (*bool*) – If True data types are included in the hash, otherwise only the raw data is hashed. Defaults to False.
- **hashlen** (*int*) – Maximum number of symbols in the returned hash. If not specified, all are returned. DEPRECATED. Use slice syntax instead.
- **convert** (*bool, optional, default=True*) – if True, try and convert the data to json an the json is hashed instead. This can improve runtime in some instances, however the hash may differ from the case where `convert=False`.

**Notes**

alphabet26 is a pretty nice base, I recommend it. However we default to hex because it is standard. This means the output of `hashdata` with `base=sha1` will be the same as the output of `sha1sum`.

**Returns** text - hash string

**Return type** str



## Example

```
>>> import ubelt as ub
>>> print(ub.hash_data([1, 2, (3, '4')], convert=False))
60b758587f599663931057e6ebdf185a...
>>> print(ub.hash_data([1, 2, (3, '4')], base='abc', hasher='sha512')[:32])
hsrgqvfiuxvvhcdnypivhhthmrolkzej
```

`ubelt.hash_file(fpath, blocksize=65536, stride=1, hasher=NoParam, hashlen=NoParam, base=NoParam)`  
 Hashes the data in a file on disk.

### Parameters

- **fpath** (*PathLike*) – 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, str*) – list of symbols or shorthand key. Valid keys are ‘abc’, ‘hex’, and ‘dec’. Defaults to ‘hex’.

## Notes

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

## References

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

## 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='sha1', base='hex'))
8843d7f92416211de9ebb963ff4ce28125932878
```

## Example

```
>>> import ubelt as ub
>>> from os.path import join
>>> fpath = ub.touch(join(ub.ensure_app_cache_dir('ubelt'), 'empty_file'))
>>> # Test that the output is the same as shasum
>>> if ub.find_exe('shasum'):
```

(continues on next page)

(continued from previous page)

```
>>> want = ub.cmd(['sha1sum', fpath], verbose=2)['out'].split(' ')[0]
>>> got = ub.hash_file(fpath, hasher='sha1')
>>> print('want = {!r}'.format(want))
>>> print('got = {!r}'.format(got))
>>> assert want.endswith(got)
>>> # Do the same for sha512 sum and md5sum
>>> if ub.find_exe('sha512sum'):
>>>     want = ub.cmd(['sha512sum', fpath], verbose=2)['out'].split(' ')[0]
>>>     got = ub.hash_file(fpath, hasher='sha512')
>>>     print('want = {!r}'.format(want))
>>>     print('got = {!r}'.format(got))
>>>     assert want.endswith(got)
>>> if ub.find_exe('md5sum'):
>>>     want = ub.cmd(['md5sum', fpath], verbose=2)['out'].split(' ')[0]
>>>     got = ub.hash_file(fpath, hasher='md5')
>>>     print('want = {!r}'.format(want))
>>>     print('got = {!r}'.format(got))
>>>     assert want.endswith(got)
```

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

Highlights a block of text using ANSI 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()))"`

### Example

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

`ubelt.hzcat(args, sep="")`

Horizontally concatenates strings preserving indentation

Concatenates 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[str]*) – strings to concatenate
- **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
>>> import unicodedata
>>> 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.identity(arg)`

The identity function. Simply returns its inputs.

**Parameters** `arg (object)` – some value

**Returns** `arg`: the same value

**Return type** `object`

**Example**

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

`ubelt.import_module_from_name(modname)`

Imports a module from its string name (`__name__`)

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

**Returns** `module`

**Return type** `module`

**Example**

```
>>> # test with modules that wont be imported in normal circumstances
>>> # todo write a test where we gaurentee this
>>> modname_list = [
>>>     '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.import_module_from_path(modpath, index=-1)
```

Imports a module via its path

**Parameters** `modpath` (*PathLike*) – path to the module on disk or within a zipfile.

**Returns** the imported module

**Return type** module

## References

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

## 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

This can import a module from within a zipfile. To do this `modpath` should specify the path to the zipfile and the path to the module within that zipfile separated by a colon or `pathsep`. E.g. `/path/to/archive.zip:mymodule.py`

**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.

## Example

```
>>> import xdoctest
>>> modpath = xdoctest.__file__
>>> module = import_module_from_path(modpath)
>>> assert module is xdoctest
```

## Example

```
>>> # Test importing a module from within a zipfile
>>> import zipfile
>>> from xdoctest import utils
>>> from os.path import join, expanduser
>>> dpath = expanduser('~/.cache/xdoctest')
```

(continues on next page)

(continued from previous page)

```

>>> dpath = utils.ensuredir(dpath)
>>> #dpath = utils.TempDir().ensure()
>>> # Write to an external module named bar
>>> external_modpath = join(dpath, 'bar.py')
>>> open(external_modpath, 'w').write('testvar = 1')
>>> internal = 'folder/bar.py'
>>> # Move the external bar module into a zipfile
>>> zippath = join(dpath, 'myzip.zip')
>>> with zipfile.ZipFile(zippath, 'w') as myzip:
>>>     myzip.write(external_modpath, internal)
>>> # Import the bar module from within the zipfile
>>> modpath = zippath + ':' + internal
>>> modpath = zippath + os.path.sep + internal
>>> module = import_module_from_path(modpath)
>>> assert module.__name__ == os.path.normpath('folder/bar')
>>> assert module.testvar == 1

```

**Doctest:**

```

>>> import pytest
>>> with pytest.raises(IOError):
>>>     import_module_from_path('does-not-exist')
>>> with pytest.raises(IOError):
>>>     import_module_from_path('does-not-exist.zip/')

```

`ubelt.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`

**Example**

```

>>> from ubelt.util_str import * # NOQA
>>> NL = chr(10) # newline character
>>> text = 'Lorem ipsum' + NL + 'dolor sit amet'
>>> prefix = ' '
>>> result = indent(text, prefix)
>>> assert all(t.startswith(prefix) for t in result.split(NL))

```

`ubelt.inject_method(self, func, name=None)`

Injects a function into an object instance as a bound method

The main use case of this function is for monkey patching. While monkey patching is sometimes necessary it should generally be avoided. Thus, we simply remind the developer that there might be a better way.

**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.

### 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'
```

`ubelt.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

**Return type** dict

### 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`

### Example

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

### Example

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

### Example

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

`ubelt.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** (*Iterable*) – 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

### 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)]
```

### 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)]
```

### 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)
```

(continues on next page)

(continued from previous page)

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

### 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.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

### 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.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`

### Example

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

(continues on next page)



(continued from previous page)

```

>>> 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.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`

### 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.memoize(func)`

memoization decorator that respects args and kwargs

### References

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

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

**Returns** memoized wrapper

**Return type** func

**CommandLine:** `xdoctest -m ubelt.util_memoize memoize`

### 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.memoize_method(func)`

Bases: `object`

memoization decorator for a method that respects args and kwargs

### References

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

### 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'
```

(continues on next page)

(continued from previous page)

```

>>> 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

```

`ubelt.memoize_property` (*fget*)

Return a property attribute for new-style classes that only calls its getter on the first access. The result is stored and on subsequent accesses is returned, preventing the need to call the getter any more.

This decorator can either be used by itself or by decorating another property. In either case the method will always become a property.

## Notes

implementation is a modified version of [1].

## References

..[1] <https://github.com/estebistec/python-memoized-property>

**CommandLine:** `xdoctest -m ubelt.util_memoize memoize_property`

## Example

```

>>> class C(object):
...     load_name_count = 0
...     @memoize_property
...     def name(self):
...         "name's docstring"
...         self.load_name_count += 1
...         return "the name"
...     @memoize_property
...     @property
...     def another_name(self):
...         "name's docstring"
...         self.load_name_count += 1
...         return "the name"
>>> c = C()
>>> c.load_name_count
0
>>> c.name
'the name'
>>> c.load_name_count
1

```

(continues on next page)

(continued from previous page)

```
>>> c.name
'the name'
>>> c.load_name_count
1
>>> c.another_name
```

ubelt.**modname\_to\_modpath** (*modname*, *hide\_init=True*, *hide\_main=False*, *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 xdoctest.static\_analysis modname\_to\_modpath:0 pytest  
/home/joncrall/code/xdoctest/xdoctest/static\_analysis.py::modname\_to\_modpath:0

#### Example

```
>>> modname = 'xdoctest.__main__'
>>> modpath = modname_to_modpath(modname, hide_main=False)
>>> assert modpath.endswith('__main__.py')
>>> modname = 'xdoctest'
>>> modpath = modname_to_modpath(modname, hide_init=False)
>>> assert modpath.endswith('__init__.py')
>>> modpath = basename(modname_to_modpath('_ctypes'))
>>> assert 'ctypes' in modpath
```

ubelt.**modpath\_to\_modname** (*modpath*, *hide\_init=True*, *hide\_main=False*, *check=True*, *relativeto=None*)

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)

- **check** (*bool*) – if False, does not raise an error if modpath is a dir and does not contain an `__init__` file.
- **relativeto** (*str, optional*) – if specified, all checks are ignored and this is considered the path to the root module.

**Returns** modname

**Return type** str

**Raises** ValueError – if check is True and the path does not exist

**CommandLine:** `xdoctest -m xdoctest.static_analysis modpath_to_modname`

### Example

```
>>> from xdoctest import static_analysis
>>> modpath = static_analysis.__file__.replace('.pyc', '.py')
>>> modpath = modpath.replace('.pyc', '.py')
>>> modname = modpath_to_modname(modpath)
>>> assert modname == 'xdoctest.static_analysis'
```

### Example

```
>>> import xdoctest
>>> assert modpath_to_modname(xdoctest.__file__.replace('.pyc', '.py')) ==
↳ 'xdoctest'
>>> assert modpath_to_modname(dirname(xdoctest.__file__.replace('.pyc', '.py'))).
↳ == 'xdoctest'
```

### Example

```
>>> modpath = modname_to_modpath('_ctypes')
>>> modname = modpath_to_modname(modpath)
>>> assert modname == '_ctypes'
```

`ubelt.odict`  
alias of `collections.OrderedDict`

`ubelt.oset`  
alias of `ordered_set.OrderedSet`

`ubelt.peek` (*iterable*)  
Look at the first item of an iterable. If the input is an iterator, then the next element is exhausted (i.e. a pop operation).

**Parameters** *iterable* (*List[T]*) – an iterable

**Returns**

**item:** the first item of ordered sequence, a popped item from an iterator, or an arbitrary item from an unordered collection.

**Return type** T

## Example

```
>>> import ubelt as ub
>>> data = [0, 1, 2]
>>> ub.peek(data)
0
>>> iterator = iter(data)
>>> print(ub.peek(iterator))
0
>>> print(ub.peek(iterator))
1
>>> print(ub.peek(iterator))
2
>>> ub.peek(range(3))
0
```

`ubelt.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** PathLike

`ubelt.platform_config_dir()`

Returns a directory which should be writable for any application This should be used for persistent configuration files.

**Returns** path to the cahce dir used by the current operating system

**Return type** PathLike

`ubelt.platform_data_dir()`

Returns path for user-specific data files

**Returns** path to the data dir used by the current operating system

**Return type** PathLike

`ubelt.platform_resource_dir()`

Alias for platform\_cache\_dir

DEPRICATED in favor of platform\_config\_dir / platform\_data\_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** PathLike

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

Reads (utf8) text from a file.

### Parameters

- **fpath** (*PathLike*) – 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.repr2(data, **kwargs)`

Makes a pretty and easy-to-doctest 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.

## Notes

This function has many keyword arguments that can be used to customize the final representation. For convenience some of the more frequently used kwargs have short aliases. See *Args* for more details.

### Parameters

- **data** (*object*) – an arbitrary python object
- **\*\*kwargs** – see *the Kwargs* section

### Kwargs:

**si, stritems, (bool):** dict/list items use str instead of repr

**strkeys, sk (bool):** dict keys use str instead of repr

**strvals, sv (bool):** dict values use str instead of repr

**nl, newlines (int | bool):** number of top level nestings to place a newline after. If true all items are followed by newlines regardless of nesting level. Defaults to 1 for lists and True for dicts.

**nobr, nobraces (bool, default=False):** if True, text will not contain outer braces for containers

**cbr, compact\_brace (bool, default=False):** if True, braces are compactified (i.e. they will not have newlines placed directly after them, think java / K&R / 1TBS)

**trailsep, trailing\_sep (bool):** if True, a separator is placed after the last item in a sequence. By default this is True if there are any `nl > 0`.

**explicit (bool, default=False):** changes dict representation from `{kl: vl, ...}` to `dict(kl=vl, ...)`.

**precision (int, default=None):** if specified floats are formatted with this precision

**kvsep (str, default=': '):** separator between keys and values

**itemsep (str, default=' '):** separator between items

**sort (bool):** if True, attempts to sort all unordered collections in the returned text. NOTE: currently if True this will sort lists, this may not be a correct thing to do, as such the behavior of this arg is subject to change.

**suppress\_small (bool):** passed to `numpy.array2string` for ndarrays

**max\_line\_width (int):** passed to `numpy.array2string` for ndarrays

**with\_dtype (bool):** only relevant to ndarrays. if True includes the dtype.

**Returns** `outstr`: output string

**Return type** `str`

## Notes

**There are also internal kwargs, which should not be used:** `_return_info` (bool): return information about child context `_root_info` (depth): information about parent context

**CommandLine:** python -m ubelt.util\_format repr2:0 python -m ubelt.util\_format repr2:1

### 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)
```

### Example

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

`ubelt.split_modpath(modpath, check=True)`

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
- **check** (*bool*) – if False, does not raise an error if modpath is a directory and does not contain an `__init__.py` file.

**Returns** (directory, rel\_modpath)

**Return type** tuple

**Raises** `ValueError` – if modpath does not exist or is not a package

### Example

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

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

### 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.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** (*PathLike*) – path to real file or directory
- **link\_path** (*PathLike*) – 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** PathLike

**CommandLine:** python -m ubelt.util\_links symlink:0

### 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]]
```

### 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)
```

`ubelt.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** (*Sequence*) – an indexable object to select items from
- **indices** (*Iterable*) – sequence of indexing objects

**Returns** subset of the list

**Return type** Iterable or scalar

**SeeAlso:** `ub.dict_subset`

### Example

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

`ubelt.timestamp(method='iso8601')`  
make an iso8601 timestamp

**Parameters** `method` (*str*) – type of timestamp

### Example

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

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

Works like the touch unix utility

#### Parameters

- **fpath** (*PathLike*) – 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).

**Returns** path to the file

**Return type** PathLike

### References

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

### Example

```
>>> 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.truepath(path, real=False)`

Normalizes a string representation of a path and does shell-like expansion.

#### Parameters

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

**Returns** normalized path

**Return type** PathLike

---

**Note:** This function is similar 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`

### 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.unique(items, key=None)`

Generates unique items in the order they appear.

#### Parameters

- **items** (*Iterable*) – list of items
- **key** (*Callable, 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`

### 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]
```

### 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.unique_flags(items, key=None)`

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

#### Parameters

- **items** (*Sequence*) – indexable collection of items
- **key** (*Callable, optional*) – custom normalization function. If specified returns items where `key(item)` is unique.

**Returns** flags the items that are unique

**Return type** List[bool]

### 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.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** PathLike

### Example

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

`ubelt.writeto` (*fpath*, *to\_write*, *aslines=False*, *verbose=None*)  
Writes (utf8) text to a file.

#### Parameters

- **fpath** (*PathLike*) – 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`

#### Example

```
>>> 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
```

#### Example

```
>>> 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
```

## CHAPTER 2

---

### Indices and tables

---

- `genindex`
- `modindex`





### u

- [ubelt](#), 78
- [ubelt.\\_\\_init\\_\\_](#), 1
- [ubelt.orderedset](#), 3
- [ubelt.progiter](#), 8
- [ubelt.util\\_arg](#), 12
- [ubelt.util\\_cache](#), 13
- [ubelt.util\\_cmd](#), 19
- [ubelt.util\\_colors](#), 21
- [ubelt.util\\_const](#), 22
- [ubelt.util\\_dict](#), 23
- [ubelt.util\\_download](#), 31
- [ubelt.util\\_format](#), 34
- [ubelt.util\\_func](#), 37
- [ubelt.util\\_hash](#), 37
- [ubelt.util\\_import](#), 40
- [ubelt.util\\_io](#), 44
- [ubelt.util\\_links](#), 47
- [ubelt.util\\_list](#), 49
- [ubelt.util\\_memoize](#), 56
- [ubelt.util\\_mixins](#), 60
- [ubelt.util\\_path](#), 60
- [ubelt.util\\_platform](#), 64
- [ubelt.util\\_str](#), 70
- [ubelt.util\\_stream](#), 72
- [ubelt.util\\_time](#), 73



## A

add() (*ubelt.OrderedSet method*), 87  
 add() (*ubelt.orderedset.OrderedSet method*), 4  
 allsame() (*in module ubelt*), 100  
 allsame() (*in module ubelt.util\_list*), 54  
 append() (*ubelt.OrderedSet method*), 87  
 append() (*ubelt.orderedset.OrderedSet method*), 4  
 argflag() (*in module ubelt*), 101  
 argflag() (*in module ubelt.util\_arg*), 12  
 argmax() (*in module ubelt*), 101  
 argmax() (*in module ubelt.util\_list*), 55  
 argmin() (*in module ubelt*), 101  
 argmin() (*in module ubelt.util\_list*), 56  
 argsort() (*in module ubelt*), 102  
 argsort() (*in module ubelt.util\_list*), 54  
 argunique() (*in module ubelt*), 102  
 argunique() (*in module ubelt.util\_list*), 52  
 argval() (*in module ubelt*), 103  
 argval() (*in module ubelt.util\_arg*), 12  
 augpath() (*in module ubelt*), 103  
 augpath() (*in module ubelt.util\_path*), 60  
 AutoDict (*class in ubelt*), 79  
 AutoDict (*class in ubelt.util\_dict*), 23  
 AutoOrderedDict (*class in ubelt*), 79  
 AutoOrderedDict (*class in ubelt.util\_dict*), 24

## B

begin() (*ubelt.ProgIter method*), 93  
 begin() (*ubelt.progiter.ProgIter method*), 10  
 boolmask() (*in module ubelt*), 104  
 boolmask() (*in module ubelt.util\_list*), 53

## C

Cacher (*class in ubelt*), 81  
 Cacher (*class in ubelt.util\_cache*), 14  
 CacheStamp (*class in ubelt*), 80  
 CacheStamp (*class in ubelt.util\_cache*), 18  
 call() (*ubelt.Timerit method*), 98  
 call() (*ubelt.util\_time.Timerit method*), 76

CaptureStdout (*class in ubelt*), 84  
 CaptureStdout (*class in ubelt.util\_stream*), 72  
 CaptureStream (*class in ubelt*), 85  
 CaptureStream (*class in ubelt.util\_stream*), 72  
 chunks (*class in ubelt*), 105  
 chunks (*class in ubelt.util\_list*), 49  
 cleanup() (*ubelt.TempDir method*), 96  
 cleanup() (*ubelt.util\_path.TempDir method*), 64  
 clear() (*ubelt.Cacher method*), 83  
 clear() (*ubelt.OrderedSet method*), 89  
 clear() (*ubelt.orderedset.OrderedSet method*), 6  
 clear() (*ubelt.util\_cache.Cacher method*), 16  
 close() (*ubelt.CaptureStdout method*), 85  
 close() (*ubelt.util\_stream.CaptureStdout method*), 73  
 cmd() (*in module ubelt*), 106  
 cmd() (*in module ubelt.util\_cmd*), 19  
 codeblock() (*in module ubelt*), 108  
 codeblock() (*in module ubelt.util\_str*), 70  
 color\_text() (*in module ubelt*), 108  
 color\_text() (*in module ubelt.util\_colors*), 22  
 compress() (*in module ubelt*), 109  
 compress() (*in module ubelt.util\_list*), 51  
 compressuser() (*in module ubelt*), 109  
 compressuser() (*in module ubelt.util\_path*), 62  
 copy() (*ubelt.OrderedSet method*), 87  
 copy() (*ubelt.orderedset.OrderedSet method*), 4  
 cycle() (*ubelt.chunks static method*), 106  
 cycle() (*ubelt.util\_list.chunks static method*), 50

## D

ddict (*in module ubelt*), 110  
 ddict (*in module ubelt.util\_dict*), 23  
 delete() (*in module ubelt*), 110  
 delete() (*in module ubelt.util\_io*), 46  
 dict\_hist() (*in module ubelt*), 111  
 dict\_hist() (*in module ubelt.util\_dict*), 25  
 dict\_isect() (*in module ubelt*), 111  
 dict\_isect() (*in module ubelt.util\_dict*), 28  
 dict\_subset() (*in module ubelt*), 112  
 dict\_subset() (*in module ubelt.util\_dict*), 27

dict\_take() (in module ubelt), 112  
dict\_take() (in module ubelt.util\_dict), 27  
dict\_union() (in module ubelt), 113  
dict\_union() (in module ubelt.util\_dict), 28  
difference() (ubelt.OrderedSet method), 90  
difference() (ubelt.orderedset.OrderedSet method), 6  
difference\_update() (ubelt.OrderedSet method), 91  
difference\_update() (ubelt.orderedset.OrderedSet method), 7  
discard() (ubelt.OrderedSet method), 89  
discard() (ubelt.orderedset.OrderedSet method), 5  
display\_message() (ubelt.ProgIter method), 95  
display\_message() (ubelt.progiter.ProgIter method), 11  
download() (in module ubelt), 113  
download() (in module ubelt.util\_download), 31  
dzip() (in module ubelt), 115  
dzip() (in module ubelt.util\_dict), 24

## E

editfile() (in module ubelt), 115  
editfile() (in module ubelt.util\_platform), 69  
encoding (ubelt.TeeStringIO attribute), 95  
encoding (ubelt.util\_stream.TeeStringIO attribute), 72  
end() (ubelt.ProgIter method), 94  
end() (ubelt.progiter.ProgIter method), 10  
ensure() (ubelt.Cacher method), 84  
ensure() (ubelt.TempDir method), 96  
ensure() (ubelt.util\_cache.Cacher method), 17  
ensure() (ubelt.util\_path.TempDir method), 64  
ensure\_app\_cache\_dir() (in module ubelt), 116  
ensure\_app\_cache\_dir() (in module ubelt.util\_platform), 66  
ensure\_app\_config\_dir() (in module ubelt), 116  
ensure\_app\_config\_dir() (in module ubelt.util\_platform), 65  
ensure\_app\_data\_dir() (in module ubelt), 116  
ensure\_app\_data\_dir() (in module ubelt.util\_platform), 65  
ensure\_app\_resource\_dir() (in module ubelt), 117  
ensure\_app\_resource\_dir() (in module ubelt.util\_platform), 69  
ensure\_newline() (ubelt.ProgIter method), 94  
ensure\_newline() (ubelt.progiter.ProgIter method), 11  
ensure\_unicode() (in module ubelt), 117  
ensure\_unicode() (in module ubelt.util\_str), 71  
ensuredir() (in module ubelt), 117  
ensuredir() (in module ubelt.util\_path), 63  
existing\_versions() (ubelt.Cacher method), 82

existing\_versions() (ubelt.util\_cache.Cacher method), 16  
exists() (ubelt.Cacher method), 82  
exists() (ubelt.util\_cache.Cacher method), 16  
expandpath() (in module ubelt), 118  
expandpath() (in module ubelt.util\_path), 62  
expired() (ubelt.CacheStamp method), 81  
expired() (ubelt.util\_cache.CacheStamp method), 19

## F

find\_duplicates() (in module ubelt), 118  
find\_duplicates() (in module ubelt.util\_dict), 26  
find\_exe() (in module ubelt), 119  
find\_exe() (in module ubelt.util\_platform), 67  
find\_path() (in module ubelt), 120  
find\_path() (in module ubelt.util\_platform), 68  
flatten() (in module ubelt), 121  
flatten() (in module ubelt.util\_list), 51  
flush() (ubelt.TeeStringIO method), 95  
flush() (ubelt.util\_stream.TeeStringIO method), 72  
FORCE\_DISABLE (ubelt.Cacher attribute), 82  
FORCE\_DISABLE (ubelt.util\_cache.Cacher attribute), 16  
format\_message() (ubelt.ProgIter method), 94  
format\_message() (ubelt.progiter.ProgIter method), 10  
FormatterExtensions (class in ubelt), 85  
FormatterExtensions (class in ubelt.util\_format), 36

## G

get\_app\_cache\_dir() (in module ubelt), 121  
get\_app\_cache\_dir() (in module ubelt.util\_platform), 66  
get\_app\_config\_dir() (in module ubelt), 121  
get\_app\_config\_dir() (in module ubelt.util\_platform), 65  
get\_app\_data\_dir() (in module ubelt), 122  
get\_app\_data\_dir() (in module ubelt.util\_platform), 65  
get\_app\_resource\_dir() (in module ubelt), 122  
get\_app\_resource\_dir() (in module ubelt.util\_platform), 69  
get\_fpath() (ubelt.Cacher method), 82  
get\_fpath() (ubelt.util\_cache.Cacher method), 16  
get\_indexer() (ubelt.OrderedSet method), 88  
get\_indexer() (ubelt.orderedset.OrderedSet method), 5  
get\_loc() (ubelt.OrderedSet method), 88  
get\_loc() (ubelt.orderedset.OrderedSet method), 5  
grabdata() (in module ubelt), 122  
grabdata() (in module ubelt.util\_download), 32  
group\_items() (in module ubelt), 123  
group\_items() (in module ubelt.util\_dict), 24

## H

hash\_data() (in module ubelt), 124  
 hash\_data() (in module ubelt.util\_hash), 38  
 hash\_file() (in module ubelt), 125  
 hash\_file() (in module ubelt.util\_hash), 39  
 highlight\_code() (in module ubelt), 126  
 highlight\_code() (in module ubelt.util\_colors), 21  
 hzcat() (in module ubelt), 126  
 hzcat() (in module ubelt.util\_str), 71

## I

identity() (in module ubelt), 127  
 identity() (in module ubelt.util\_func), 37  
 import\_module\_from\_name() (in module ubelt), 127  
 import\_module\_from\_name() (in module ubelt.util\_import), 42  
 import\_module\_from\_path() (in module ubelt), 127  
 import\_module\_from\_path() (in module ubelt.util\_import), 40  
 indent() (in module ubelt), 129  
 indent() (in module ubelt.util\_str), 70  
 index() (ubelt.OrderedSet method), 88  
 index() (ubelt.orderedset.OrderedSet method), 5  
 inject\_method() (in module ubelt), 129  
 inject\_method() (in module ubelt.util\_func), 37  
 intersection() (ubelt.OrderedSet method), 89  
 intersection() (ubelt.orderedset.OrderedSet method), 6  
 intersection\_update() (ubelt.OrderedSet method), 91  
 intersection\_update() (ubelt.orderedset.OrderedSet method), 7  
 invert\_dict() (in module ubelt), 130  
 invert\_dict() (in module ubelt.util\_dict), 30  
 isatty() (ubelt.TeeStringIO method), 95  
 isatty() (ubelt.util\_stream.TeeStringIO method), 72  
 issubset() (ubelt.OrderedSet method), 90  
 issubset() (ubelt.orderedset.OrderedSet method), 6  
 issuperset() (ubelt.OrderedSet method), 90  
 issuperset() (ubelt.orderedset.OrderedSet method), 7  
 iter\_window() (in module ubelt), 131  
 iter\_window() (in module ubelt.util\_list), 53  
 iterable() (in module ubelt), 132  
 iterable() (in module ubelt.util\_list), 50

## L

load() (ubelt.Cacher method), 83  
 load() (ubelt.util\_cache.Cacher method), 17  
 log\_part() (ubelt.CaptureStdout method), 85  
 log\_part() (ubelt.util\_stream.CaptureStdout method), 73

lookup() (ubelt.FormatterExtensions method), 86  
 lookup() (ubelt.util\_format.FormatterExtensions method), 36

## M

map\_keys() (in module ubelt), 132  
 map\_keys() (in module ubelt.util\_dict), 29  
 map\_vals() (in module ubelt), 133  
 map\_vals() (in module ubelt.util\_dict), 29  
 mean() (ubelt.Timerit method), 99  
 mean() (ubelt.util\_time.Timerit method), 77  
 memoize() (in module ubelt), 133  
 memoize() (in module ubelt.util\_memoize), 57  
 memoize\_method(class in ubelt), 134  
 memoize\_method(class in ubelt.util\_memoize), 58  
 memoize\_property() (in module ubelt), 135  
 memoize\_property() (in module ubelt.util\_memoize), 59  
 min() (ubelt.Timerit method), 98  
 min() (ubelt.util\_time.Timerit method), 76  
 modname\_to\_modpath() (in module ubelt), 136  
 modname\_to\_modpath() (in module ubelt.util\_import), 42  
 modpath\_to\_modname() (in module ubelt), 136  
 modpath\_to\_modname() (in module ubelt.util\_import), 43

## N

NiceRepr(class in ubelt), 86  
 NiceRepr(class in ubelt.util\_mixins), 60  
 noborder() (ubelt.chunks static method), 106  
 noborder() (ubelt.util\_list.chunks static method), 50

## O

odict(in module ubelt), 137  
 odict(in module ubelt.util\_dict), 23  
 OrderedSet(class in ubelt), 87  
 OrderedSet(class in ubelt.orderedset), 3  
 oset(in module ubelt), 137  
 oset(in module ubelt.orderedset), 8

## P

peek() (in module ubelt), 137  
 peek() (in module ubelt.util\_list), 56  
 platform\_cache\_dir() (in module ubelt), 138  
 platform\_cache\_dir() (in module ubelt.util\_platform), 65  
 platform\_config\_dir() (in module ubelt), 138  
 platform\_config\_dir() (in module ubelt.util\_platform), 64  
 platform\_data\_dir() (in module ubelt), 138  
 platform\_data\_dir() (in module ubelt.util\_platform), 64

platform\_resource\_dir() (in module *ubelt*), 138  
platform\_resource\_dir() (in module *ubelt.util\_platform*), 69  
pop() (*ubelt.OrderedSet* method), 88  
pop() (*ubelt.orderedset.OrderedSet* method), 5  
print() (*ubelt.Timerit* method), 100  
print() (*ubelt.util\_time.Timerit* method), 78  
ProgIter (class in *ubelt*), 91  
ProgIter (class in *ubelt.progiter*), 8

## R

readfrom() (in module *ubelt*), 138  
readfrom() (in module *ubelt.util\_io*), 45  
register() (*ubelt.FormatterExtensions* method), 86  
register() (*ubelt.util\_format.FormatterExtensions* method), 36  
renew() (*ubelt.CacheStamp* method), 81  
renew() (*ubelt.util\_cache.CacheStamp* method), 19  
replicate() (*ubelt.chunks* static method), 106  
replicate() (*ubelt.util\_list.chunks* static method), 50  
report() (*ubelt.Timerit* method), 99  
report() (*ubelt.util\_time.Timerit* method), 77  
repr2() (in module *ubelt*), 139  
repr2() (in module *ubelt.util\_format*), 34  
reset() (*ubelt.Timerit* method), 98  
reset() (*ubelt.util\_time.Timerit* method), 76

## S

save() (*ubelt.Cacher* method), 83  
save() (*ubelt.util\_cache.Cacher* method), 17  
set\_extra() (*ubelt.ProgIter* method), 93  
set\_extra() (*ubelt.progiter.ProgIter* method), 9  
split\_modpath() (in module *ubelt*), 140  
split\_modpath() (in module *ubelt.util\_import*), 44  
start() (*ubelt.CaptureStdout* method), 85  
start() (*ubelt.TempDir* method), 96  
start() (*ubelt.util\_path.TempDir* method), 64  
start() (*ubelt.util\_stream.CaptureStdout* method), 73  
startfile() (in module *ubelt*), 141  
startfile() (in module *ubelt.util\_platform*), 66  
std() (*ubelt.Timerit* method), 99  
std() (*ubelt.util\_time.Timerit* method), 77  
step() (*ubelt.ProgIter* method), 93  
step() (*ubelt.progiter.ProgIter* method), 10  
stop() (*ubelt.CaptureStdout* method), 85  
stop() (*ubelt.util\_stream.CaptureStdout* method), 73  
symlink() (in module *ubelt*), 141  
symlink() (in module *ubelt.util\_links*), 47  
symmetric\_difference() (*ubelt.OrderedSet* method), 90  
symmetric\_difference() (*ubelt.orderedset.OrderedSet* method), 7  
symmetric\_difference\_update() (*ubelt.OrderedSet* method), 91

symmetric\_difference\_update() (*ubelt.orderedset.OrderedSet* method), 8

## T

take() (in module *ubelt*), 142  
take() (in module *ubelt.util\_list*), 50  
TeeStringIO (class in *ubelt*), 95  
TeeStringIO (class in *ubelt.util\_stream*), 72  
TempDir (class in *ubelt*), 95  
TempDir (class in *ubelt.util\_path*), 63  
tic() (*ubelt.Timer* method), 97  
tic() (*ubelt.util\_time.Timer* method), 74  
Timer (class in *ubelt*), 96  
Timer (class in *ubelt.util\_time*), 74  
Timerit (class in *ubelt*), 97  
Timerit (class in *ubelt.util\_time*), 75  
timestamp() (in module *ubelt*), 143  
timestamp() (in module *ubelt.util\_time*), 73  
to\_dict() (*ubelt.AutoDict* method), 79  
to\_dict() (*ubelt.util\_dict.AutoDict* method), 23  
toc() (*ubelt.Timer* method), 97  
toc() (*ubelt.util\_time.Timer* method), 75  
touch() (in module *ubelt*), 143  
touch() (in module *ubelt.util\_io*), 46  
truepath() (in module *ubelt*), 144  
truepath() (in module *ubelt.util\_path*), 62  
tryload() (*ubelt.Cacher* method), 83  
tryload() (*ubelt.util\_cache.Cacher* method), 16

## U

ubelt (module), 78  
ubelt.\_\_init\_\_ (module), 1  
ubelt.orderedset (module), 3  
ubelt.progiter (module), 8  
ubelt.util\_arg (module), 12  
ubelt.util\_cache (module), 13  
ubelt.util\_cmd (module), 19  
ubelt.util\_colors (module), 21  
ubelt.util\_const (module), 22  
ubelt.util\_dict (module), 23  
ubelt.util\_download (module), 31  
ubelt.util\_format (module), 34  
ubelt.util\_func (module), 37  
ubelt.util\_hash (module), 37  
ubelt.util\_import (module), 40  
ubelt.util\_io (module), 44  
ubelt.util\_links (module), 47  
ubelt.util\_list (module), 49  
ubelt.util\_memoize (module), 56  
ubelt.util\_mixins (module), 60  
ubelt.util\_path (module), 60  
ubelt.util\_platform (module), 64  
ubelt.util\_str (module), 70  
ubelt.util\_stream (module), 72

`ubelt.util_time` (*module*), 73  
`union()` (*ubelt.OrderedSet method*), 89  
`union()` (*ubelt.orderedset.OrderedSet method*), 6  
`unique()` (*in module ubelt*), 144  
`unique()` (*in module ubelt.util\_list*), 51  
`unique_flags()` (*in module ubelt*), 145  
`unique_flags()` (*in module ubelt.util\_list*), 52  
`update()` (*ubelt.OrderedSet method*), 88  
`update()` (*ubelt.orderedset.OrderedSet method*), 4  
`userhome()` (*in module ubelt*), 145  
`userhome()` (*in module ubelt.util\_path*), 61

## V

`VERBOSE` (*ubelt.Cacher attribute*), 82  
`VERBOSE` (*ubelt.util\_cache.Cacher attribute*), 16

## W

`write()` (*ubelt.TeeStringIO method*), 95  
`write()` (*ubelt.util\_stream.TeeStringIO method*), 72  
`writeto()` (*in module ubelt*), 145  
`writeto()` (*in module ubelt.util\_io*), 45