Cawdrey

Release 0.5.1

Several useful custom dictionaries for Python

Dominic Davis-Foster

Apr 26, 2024

Contents

Chapter

ONE

Highlights

- *frozendict*: An immutable dictionary that cannot be changed after creation.
- *FrozenOrderedDict*: An immutable OrderedDict where the order of keys is preserved, but that cannot be changed after creation.
- *AlphaDict*: A *FrozenOrderedDict* where the keys are stored in alphabetical order.
- *bdict*: A dictionary where `key, value` pairs are stored both ways round.
- *Tally*: A subclass of `collections.Counter` with additional methods.
- *HeaderMapping*: A `collections.abc.MutableMapping` which supports duplicate, case-insentive keys.

This package also provides two base classes for creating your own custom dictionaries:

- *FrozenBase*: An Abstract Base Class for frozen dictionaries.
- *MutableBase*: An Abstract Base Class for mutable dictionaries.

Chapter

TWO

Other Dictionary Packages

If you're looking to unflatten a dictionary, such as to go from this:

```
{"foo.bar": "val"}
```

to this:

```
{"foo": {"bar": "val"}}
```

check out unflatten, flattery or morph to accomplish that.

indexed provides an OrderedDict where the values can be accessed by their index as well as by their keys.

There's also python-benedict, which provides a custom dictionary with **keylist/keypath** support, **I/O** shortcuts (`Base64`, `CSV`, `JSON`, `TOML`, `XML`, `YAML`, `pickle`, `query-string`) and many **utilities**.

Chapter

THREE

Installation

3.1 from PyPI

```
$ python3 -m pip install cawdrey --user
```

3.2 from Anaconda

First add the required channels

```
$ conda config --add channels https://conda.anaconda.org/conda-forge
$ conda config --add channels https://conda.anaconda.org/domdfcoding
```

Then install

```
$ conda install cawdrey
```

3.3 from GitHub

```
$ python3 -m pip install git+https://github.com/domdfcoding/cawdrey@master --user
```

Chapter

FOUR

AlphaDict

Provides *AlphaDict*, a frozen OrderedDict where the keys are stored alphabetically.

Classes:

AlphaDict([seq])	Initialize an immutable, alphabetised dictionary.

Functions:

alphabetical_dict(**kwargs)	Returns an OrderedDict with the keys sorted alphabetically.

class AlphaDict(*seq=None*, ***kwargs*)

Bases: *FrozenOrderedDict*[*~KT*, *~VT*]

Initialize an immutable, alphabetised dictionary.

The signature is the same as regular dictionaries.

- `AlphaDict()` -> new empty AlphaDict
- `AlphaDict(mapping)` -> new AlphaDict initialized from a mapping object's (key, value) pairs
- `AlphaDict(iterable)` -> new AlphaDict initialized as if via:

```
d = {}
for k, v in iterable:
    d[k] = v
```

- `AlphaDict(**kwargs)` -> new AlphaDict initialized with the `name=value` pairs in the keyword argument list.

  For example:

```
AlphaDict(one=1, two=2)
```

alphabetical_dict(***kwargs*)

Returns an OrderedDict with the keys sorted alphabetically.

Parameters **kwargs**

Return type OrderedDict[str, *~T*]

Chapter

FIVE

bdict

class bdict(*seq=None*, ***kwargs*)

Bases: UserDict

Returns a new dictionary initialized from an optional positional argument, and a possibly empty set of keyword arguments.

Each `key: value` pair is entered into the dictionary in both directions, so you can perform lookups with either the key or the value.

If no positional argument is given, an empty dictionary is created.

If a positional argument is given and it is a mapping object, a dictionary is created with the same key-value pairs as the mapping object. Otherwise, the positional argument must be an iterable object. Each item in the iterable must itself be an iterable with exactly two objects. The first object of each item becomes a key in the new dictionary, and the second object the corresponding value.

If keyword arguments are given, the keyword arguments and their values are added to the dictionary created from the positional argument.

If an attempt is made to add a key or value that already exists in the dictionary a `ValueError` will be raised.

Keys or values of `None`, `True` and `False` will be stored internally as `"_None"`, `"_True"` and `"_False"` respectively

Methods:

__contains__(key)	Return `key in self`.
__delitem__(key)	Delete `self[key]`.
__getitem__(key)	Return `self[key]`.
__setitem__(key, val)	Set `self[key]` to `value`.
clear()	Removes all items from the *bdict*.
get(k[, default])	Return the value for `k` if `k` is in the dictionary, else `default`.
items()	Returns a set-like object providing a view on the *bdict*'s items.
keys()	Returns a set-like object providing a view on the *bdict*'s keys.
values()	Returns an object providing a view on the *bdict*'s values.

__contains__(*key*)

Return `key in self`.

Parameters **key** (`object`)

Return type `bool`

__delitem__(*key*)

Delete `self[key]`.

Parameters **key** (*~KT*)

__getitem__(*key*)
Return `self[key]`.

Parameters **key** (*~KT*)

Return type *~VT*

__setitem__(*key*, *val*)
Set `self[key]` to `value`.

Parameters

- **key**
- **val**

clear()
Removes all items from the *bdict*.

get(*k*, *default=None*)
Return the value for `k` if `k` is in the dictionary, else `default`.

Parameters

- **k** – The key to return the value for.
- **default** – The value to return if `key` is not in the dictionary. Default `None`.

Overloads

- *get*(**k**: *~KT*) -> `Optional`[*~VT*]
- *get*(**k**: *~KT*, **default**: `Union`[*~VT*, *~T*]) -> `Union`[*~VT*, *~T*]

items()
Returns a set-like object providing a view on the *bdict*'s items.

Return type `AbstractSet`[`Tuple`[*~KT*, *~VT*]]

keys()
Returns a set-like object providing a view on the *bdict*'s keys.

Return type `AbstractSet`[*~KT*]

values()
Returns an object providing a view on the *bdict*'s values.

Return type `ValuesView`[*~VT*]

Chapter

SIX

frozendict

6.1 About

frozendict is an immutable wrapper around dictionaries that implements the complete mapping interface. It can be used as a drop-in replacement for dictionaries where immutability is desired.

Of course this is Python, and you can still poke around the object's internals if you want.

The *frozendict* constructor mimics dict, and all of the expected interfaces (iter, len, repr, hash, getitem) are provided. Note that a *frozendict* does not guarantee the immutability of its values, so the utility of the hash method is restricted by usage.

The only difference is that the *copy()* method of *frozendict* takes variable keyword arguments, which will be present as key/value pairs in the new, immutable copy.

6.2 Usage

```
>>> from cawdrey import frozendict
>>>
>>> fd = frozendict({ 'hello': 'World' })
>>>
>>> print fd
<frozendict {'hello': 'World'}>
>>>
>>> print fd['hello']
'World'
>>>
>>> print fd.copy(another='key/value')
<frozendict {'hello': 'World', 'another': 'key/value'}>
>>>
```

In addition, *frozendict* supports the + and − operands. If you add a dict-like object, a new *frozendict* will be returned, equal to the old *frozendict* updated with the other object. Example:

```
>>> frozendict({"Sulla": "Marco", 2: 3}) + {"Sulla": "Marò", 4: 7}
<frozendict {'Sulla': 'Marò', 2: 3, 4: 7}>
>>>
```

You can also subtract an iterable from a *frozendict*. A new *frozendict* will be returned, without the keys that are in the iterable. Examples:

```
>>> frozendict({"Sulla": "Marco", 2: 3}) - {"Sulla": "Marò", 4: 7}
<frozendict {'Sulla': 'Marco', 2: 3}>
>>> frozendict({"Sulla": "Marco", 2: 3}) - [2, 4]
<frozendict {'Sulla': 'Marco'}>
>>>
```

Some other examples:

```
>>> from cawdrey import frozendict
>>> fd = frozendict({"Sulla": "Marco", "Hicks": "Bill"})
>>> print(fd)
<frozendict {'Sulla': 'Marco', 'Hicks': 'Bill'}>
>>> print(fd["Sulla"])
Marco
>>> fd["Bim"]
KeyError: 'Bim'
>>> len(fd)
2
>>> "Sulla" in fd
True
>>> "Sulla" not in fd
False
>>> "Bim" in fd
False
>>> hash(fd)
835910019049608535
>>> fd_unhashable = frozendict({1: []})
>>> hash(fd_unhashable)
TypeError: unhashable type: 'list'
>>> fd2 = frozendict({"Hicks": "Bill", "Sulla": "Marco"})
>>> print(fd2)
<frozendict {'Hicks': 'Bill', 'Sulla': 'Marco'}>
>>> fd2 is fd
False
>>> fd2 == fd
True
>>> frozendict()
<frozendict {}>
>>> frozendict(Sulla="Marco", Hicks="Bill")
<frozendict {'Sulla': 'Marco', 'Hicks': 'Bill'}>
>>> frozendict((("Sulla", "Marco"), ("Hicks", "Bill")))
<frozendict {'Sulla': 'Marco', 'Hicks': 'Bill'}>
>>> fd.get("Sulla")
'Marco'
>>> print(fd.get("God"))
None
>>> tuple(fd.keys())
('Sulla', 'Hicks')
>>> tuple(fd.values())
('Marco', 'Bill')
>>> tuple(fd.items())
(('Sulla', 'Marco'), ('Hicks', 'Bill'))
>>> iter(fd)
<dict_keyiterator object at 0x7feb75c49188>
>>> frozendict.fromkeys(["Marco", "Giulia"], "Sulla")
<frozendict {'Marco': 'Sulla', 'Giulia': 'Sulla'}>
>>> fd["Sulla"] = "Silla"
TypeError: 'frozendict' object does not support item assignment
>>> del fd["Sulla"]
TypeError: 'frozendict' object does not support item deletion
>>> fd.clear()
AttributeError: 'frozendict' object has no attribute 'clear'
>>> fd.pop("Sulla")
AttributeError: 'frozendict' object has no attribute 'pop'
```

(continues on next page)

(continued from previous page)

```
>>> fd.popitem()
AttributeError: 'frozendict' object has no attribute 'popitem'
>>> fd.setdefault("Sulla")
AttributeError: 'frozendict' object has no attribute 'setdefault'
>>> fd.update({"Bim": "James May"})
AttributeError: 'frozendict' object has no attribute 'update'
```

6.3 API Reference

class frozendict(**args*, ***kwargs*)

Bases: *FrozenBase*[*~KT*, *~VT*]

An immutable wrapper around dictionaries that implements the complete `collections.abc.Mapping` interface. It can be used as a drop-in replacement for dictionaries where immutability is desired.

Methods:

__add__(other, *args, **kwargs)	If you add a dict-like object, a new frozendict will be returned, equal to the old frozendict updated with the other object.
__and__(other, *args, **kwargs)	Returns a new `frozendict`, that is the intersection between `self` and `other`.
__sub__(other, *args, **kwargs)	The method will create a new *frozendict*, result of the subtraction by *other*.
copy(*args, **kwargs)	Return a copy of the dictionary.
sorted(*args[, by])	Return a new *frozendict*, with the element insertion sorted.

__add__(*other*, **args*, ***kwargs*)

If you add a dict-like object, a new frozendict will be returned, equal to the old frozendict updated with the other object.

__and__(*other*, **args*, ***kwargs*)

Returns a new `frozendict`, that is the intersection between `self` and `other`.

If `other` is a *dict*-like object, the intersection will contain only the *items* in common.

If `other` is another iterable, the intersection will contain the items of `self` which keys are in *other*.

Iterables of pairs are *not* managed differently. This is for consistency.

Beware! The final order is dictated by the order of *other*. This allows the coder to change the order of the original `frozendict`.

The last two behaviours breaks voluntarily the `dict.items()` API, for consistency and practical reasons.

__sub__(*other*, **args*, ***kwargs*)

The method will create a new *frozendict*, result of the subtraction by *other*.

If `other` is a `dict`-like, the result will have the items of the *frozendict* that are *not* in common with *other*.

If `other` is another type of iterable, the result will have the items of *frozendict* without the keys that are in `other`.

copy(**args*, ***kwargs*)
Return a copy of the dictionary.

Return type *~_D*

sorted(**args*, *by='keys'*, ***kwargs*)
Return a new *frozendict*, with the element insertion sorted. The signature is the same as the builtin *sorted* function, except for the additional parameter `by`, that is `'keys'` by default and can also be `'values'` and `'items'`. So the resulting *frozendict* can be sorted by keys, values or items.

If you want more complicated sorts read the documentation of *sorted*.

The the parameters passed to the `key` function are the keys of the `frozendict` if `by = "keys"`, and are the items otherwise.

Note: Sorting by keys and items achieves the same effect. The only difference is when you want to customize the sorting passing a custom `key` function. You *could* achieve the same result using `by = "values"`, since also sorting by values passes the items to the key function. But this is an implementation detail and you should not rely on it.

6.4 Copyright

Chapter

SEVEN

FrozenOrderedDict

7.1 About

FrozenOrderedDict is a immutable wrapper around an OrderedDict. It is similar to *frozendict*, and with regards to immutability it solves the same problems:

- Because dictionaries are mutable, they are not hashable and cannot be used in sets or as dictionary keys.
- Nasty bugs can and do occur when mutable data structures are passed around.

It can be initialized just like a dict or OrderedDict. Once instantiated, an instance of *FrozenOrderedDict* cannot be altered, since it does not implement the MutableMapping interface.

FrozenOrderedDict implements the Mapping interface, so can be used like a normal dictionary in most cases.

In order to modify the contents of a *FrozenOrderedDict*, a new instance must be created. The easiest way to do that is by calling the `.copy()` method. It will return a new instance of *FrozenOrderedDict* initialized using the following steps:

1. A copy of the wrapped OrderedDict instance will be created.
2. If any arguments or keyword arguments are passed to the `.copy()` method, they will be used to create another OrderedDict instance, which will then be used to update the copy made in step #1.
3. Finally, `self.__class__()` will be called, passing the copy as the only argument.

7.2 API Reference

class FrozenOrderedDict(**args*, ***kwargs*)

Bases: *FrozenBase*[*~KT*, *~VT*]

An immutable OrderedDict. It can be used as a drop-in replacement for dictionaries where immutability is desired.

Methods:

__contains__(key)	Return `key in self`.
__getitem__(key)	Return `self[key]`.
copy(*args, **kwargs)	Return a copy of the *FrozenOrderedDict*.
get(k[, default])	Return the value for `k` if `k` is in the dictionary, else `default`.
items()	Returns a set-like object providing a view on the *FrozenOrderedDict*'s items.
keys()	Returns a set-like object providing a view on the *FrozenOrderedDict*'s keys.
values()	Returns an object providing a view on the *FrozenOrderedDict*'s values.

__contains__(*key*)
Return `key in self`.

Parameters **key** (`object`)

Return type `bool`

__getitem__(*key*)
Return `self[key]`.

Parameters **key** (*~KT*)

Return type *~VT*

copy(**args*, ***kwargs*)
Return a copy of the *FrozenOrderedDict*.

Parameters

- **args**
- **kwargs**

get(*k*, *default=None*)
Return the value for `k` if `k` is in the dictionary, else `default`.

Parameters

- **k** – The key to return the value for.
- **default** – The value to return if `key` is not in the dictionary. Default `None`.

Overloads

- *get*(**k**: *~KT*) -> `Optional`[*~VT*]
- *get*(**k**: *~KT*, **default**: `Union`[*~VT*, *~T*]) -> `Union`[*~VT*, *~T*]

items()
Returns a set-like object providing a view on the *FrozenOrderedDict*'s items.

Return type `AbstractSet`[`Tuple`[*~KT*, *~VT*]]

keys()
Returns a set-like object providing a view on the *FrozenOrderedDict*'s keys.

Return type `AbstractSet`[*~KT*]

values()
Returns an object providing a view on the *FrozenOrderedDict*'s values.

Return type `ValuesView`[*~VT*]

7.3 Copyright

Chapter

EIGHT

HeaderMapping

collections.abc.MutableMapping which supports duplicate, case-insentive keys.

New in version 0.4.0.

Classes:

HeaderMapping()	Provides a MutableMapping interface to a list of headers, such as those used in an email message.

class HeaderMapping

Bases: MutableMapping[str, ~*VT*]

Provides a MutableMapping interface to a list of headers, such as those used in an email message.

See also: email.message.Message and email.message.EmailMessage

MutableMapping interface, which assumes there is exactly one occurrence of the header per mapping. Some headers do in fact appear multiple times, and for those headers you must use the *get_all()* method to obtain all values for that key.

Methods:

__contains__(name)	Returns whether name is in the *HeaderMapping*.
__delitem__(name)	Delete all occurrences of a header, if present.
__getitem__(name)	Get a header value.
__iter__()	Returns an iterator over the keys in the *HeaderMapping*.
__len__()	Return the total number of keys, including duplicates.
__repr__()	Return a string representation of the *HeaderMapping*.
__setitem__(name, val)	Set the value of a header.
get(k[, default])	Get a header value.
get_all(k[, default])	Return a list of all the values for the named field.
items()	Get all the message's header fields and values.
keys()	Return a list of all the message's header field names.
values()	Return a list of all the message's header values.

__contains__(*name*)

Returns whether name is in the *HeaderMapping*.

Parameters **name** (object)

Return type bool

__delitem__(*name*)
Delete all occurrences of a header, if present.

Does not raise an exception if the header is missing.

Parameters **name** (str)

__getitem__(*name*)
Get a header value.

Note: If the header appears multiple times, exactly which occurrence gets returned is undefined. Use the *get_all()* method to get all values matching a header field name.

Parameters **name** (str)

Return type *~VT*

__iter__()
Returns an iterator over the keys in the *HeaderMapping*.

Return type Iterator[str]

__len__()
Return the total number of keys, including duplicates.

Return type int

__repr__()
Return a string representation of the *HeaderMapping*.

New in version 0.4.1.

Return type str

__setitem__(*name*, *val*)
Set the value of a header.

Parameters

- **name** (str)
- **val** (*~VT*)

get(*k*, *default=None*)
Get a header value.

Like *__getitem__()*, but returns default instead of None when the field is missing.

Parameters

- **k** (str)
- **default** – Default None.

Overloads

- *get*(**k**: str) -> Optional[*~VT*]

- *get*(**k**: str, **default**: Union[*~VT*, *~T*]) -> Union[*~VT*, *~T*]

get_all(*k*, *default=None*)
Return a list of all the values for the named field.

These will be sorted in the order they appeared in the original message, and may contain duplicates. Any fields deleted and re-inserted are always appended to the header list.

If no such fields exist, `default` is returned.

Parameters

- **k** (str)
- **default** – Default None.

Overloads

- *get_all*(**k**: str) -> Optional[List[*~VT*]]
- *get_all*(**k**: str, **default**: Union[*~VT*, *~T*]) -> Union[List[*~VT*], *~T*]

items()
Get all the message's header fields and values.

These will be sorted in the order they appeared in the original message, or were added to the message, and may contain duplicates. Any fields deleted and re-inserted are always appended to the header list.

Return type List[Tuple[str, *~VT*]]

keys()
Return a list of all the message's header field names.

These will be sorted in the order they appeared in the original message, or were added to the message, and may contain duplicates. Any fields deleted and re-inserted are always appended to the header list.

Return type List[str]

values()
Return a list of all the message's header values.

These will be sorted in the order they appeared in the original message, or were added to the message, and may contain duplicates. Any fields deleted and re-inserted are always appended to the header list.

Return type List[*~VT*]

Chapter

NINE

NonelessDict

9.1 About

NonelessDict is a wrapper around dict that will check if a value is None/empty/False, and not add the key in that case.

The class has a method *set_with_strict_none_check()* that can be used to set a value and check only for None values.

NonelessOrderedDict is based on *NonelessDict* and OrderedDict, so the order of key insertion is preserved.

9.2 API Reference

Classes:

NonelessDict(*args, **kwargs)	A wrapper around dict that will check if a value is None/empty/False, and not add the key in that case.
NonelessOrderedDict(*args, **kwargs)	A wrapper around OrderedDict that will check if a value is None/empty/False, and not add the key in that case.

Data:

_ND	Invariant TypeVar bound to *cawdrey.nonelessdict.NonelessDict*.
_NOD	Invariant TypeVar bound to *cawdrey.nonelessdict.NonelessOrderedDict*.

class NonelessDict(**args*, ***kwargs*)

Bases: *MutableBase*[*~KT*, *~VT*]

A wrapper around dict that will check if a value is None/empty/False, and not add the key in that case.

Use the *set_with_strict_none_check()* method to check only for None.

Methods:

__setitem__(key, value)	Set self[key] to value.
copy(**add_or_replace)	Return a copy of the dictionary.
set_with_strict_none_check(key, value)	Set key in the dictionary to value, but skipping None values.

__setitem__(*key*, *value*)
Set `self[key]` to `value`.

copy(**add_or_replace*)
Return a copy of the dictionary.

Return type *~_ND*

set_with_strict_none_check(*key*, *value*)
Set `key` in the dictionary to `value`, but skipping `None` values.

Parameters

- **key** (*~KT*)
- **value** (`Optional`[*~VT*])

class NonelessOrderedDict(**args*, ***kwargs*)
Bases: *MutableBase*[*~KT*, *~VT*]

A wrapper around OrderedDict that will check if a value is None/empty/False, and not add the key in that case. Use the set_with_strict_none_check function to check only for None

Methods:

__setitem__(key, value)	Set `self[key]` to `value`.
copy(*args, **kwargs)	Return a copy of the dictionary.
set_with_strict_none_check(key, value)	Set `key` in the dictionary to `value`, but skipping `None` values.

__setitem__(*key*, *value*)
Set `self[key]` to `value`.

copy(**args*, ***kwargs*)
Return a copy of the dictionary.

Return type *~_NOD*

set_with_strict_none_check(*key*, *value*)
Set `key` in the dictionary to `value`, but skipping `None` values.

Parameters

- **key** (*~KT*)
- **value** (`Optional`[*~VT*])

_ND = TypeVar(_ND, bound=NonelessDict)
Type: `TypeVar`

Invariant `TypeVar` bound to *cawdrey.nonelessdict.NonelessDict*.

_NOD = TypeVar(_NOD, bound=NonelessOrderedDict)
Type: `TypeVar`

Invariant `TypeVar` bound to *cawdrey.nonelessdict.NonelessOrderedDict*.

9.3 Copyright

Based on https://github.com/slezica/python-frozendict and
https://github.com/jerr0328/python-helpfuldicts .

Chapter

TEN

Tally

Subclass of `collections.Counter` with additional methods.

New in version 0.3.0.

Data:

_F	Invariant `TypeVar` constrained to `float`, `int` and `numbers.Real`.

Classes:

SupportsMostCommon	`typing.Protocol` for classes which support a `collections.Counter`-like `collections.Counter.most_common()` method.
Tally([iterable])	Subclass of `collections.Counter` with additional methods.
Percentage	Provides a dictionary interface, but with `collections.Counter`'s `collections.Counter.most_common()` method.

_F = TypeVar(_F, float, int, Real)

Type: `TypeVar`

Invariant `TypeVar` constrained to `float`, `int` and `numbers.Real`.

protocol SupportsMostCommon

Bases: `Protocol`[*~KT*]

`typing.Protocol` for classes which support a `collections.Counter`-like `collections.Counter.most_common()` method.

This protocol is runtime checkable.

Classes that implement this protocol must have the following methods / attributes:

items()

Returns an iterator over the mapping's items (as `(key, value)` pairs).

Return type `Iterable`[`Tuple`[*~KT*, `float`]]

most_common(*n=None*)

List the `n` most common elements and their counts from the most common to the least. If `n` is `None` then list all element counts.

```
>>> Counter('abracadabra').most_common(3)
[('a', 5), ('b', 2), ('r', 2)]
```

Parameters **n** (`Optional`[`int`]) – Default `None`.

Return type `Union`[`List`[`Tuple`[*~KT*, `float`]], `List`[`Tuple`[*~KT*, `int`]]]

__non_callable_proto_members__ = {}
Type: set

class Tally(*iterable=None*, */*, ***kwds*)
Bases: Counter[~KT]

Subclass of collections.Counter with additional methods.

New in version 0.3.0.

Methods:

as_percentage()	Returns the *Tally* as a collections.OrderedDict comprising the count for each element as a percentage of the sum of all elements.
get_percentage(item[, default])	Returns the count for item, as a percentage of the sum of all elements.
most_common([n])	List the n most common elements and their counts from the most common to the least.

Attributes:

total	Returns the total count for all elements.

as_percentage()
Returns the *Tally* as a collections.OrderedDict comprising the count for each element as a percentage of the sum of all elements.

Important: The sum of the dictionary's values may not add up to exactly 1.0 due to limitations of floating-point numbers.

Return type *Percentage*[~KT]

property total
Returns the total count for all elements.

Return type int

get_percentage(*item*, *default=None*)
Returns the count for item, as a percentage of the sum of all elements.

Parameters

- **item** (*~KT*)
- **default** (Optional[*~_F*]) – A default percentage (as a float) to return if item is not in the dictionary. Default None.

Return type Union[None, *~_F*, float]

Overloads

- *get_percentage*(**item**: *~KT*) -> Optional[float]
- *get_percentage*(**item**: *~KT*, **default**: *~_F*) -> Union[*~_F*, float]

most_common(*n=None*)

List the n most common elements and their counts from the most common to the least. If n is None then list all element counts.

```
>>> Tally('abracadabra').most_common(3)
[('a', 5), ('b', 2), ('r', 2)]
```

Parameters **n** (Optional[int]) – Default None.

Return type List[Tuple[*~KT*, int]]

class Percentage

Bases: Dict[*~KT*, float]

Provides a dictionary interface, but with collections.Counter's collections.Counter.most_common() method.

Represents the return type of *cawdrey.tally.Tally.as_percentage()*.

Methods:

most_common([n])	List the n most common elements and their counts from the most common to the least.

most_common(*n=None*)

List the n most common elements and their counts from the most common to the least. If n is None then list all element counts.

```
>>> Tally('abracadabra').as_percentage().most_common(3)
[('a', 0.4545454545454545), ('b', 0.18181818181818182), ('r', 0.
↪18181818181818182)]
```

Parameters **n** (Optional[int]) – Default None.

Return type List[Tuple[*~KT*, float]]

Chapter

ELEVEN

Base Classes

11.1 About

`FrozenBase` is the base class for *`frozendict`* and *`FrozenOrderedDict`*. If you wish to construct your own frozen dictionary classes, you may inherit from this class.

11.2 API Reference

Classes:

`DictWrapper`()	Abstract Mixin class for classes that wrap a dict object or similar.
`FrozenBase`(*args, **kwargs)	Abstract Base Class for Frozen dictionaries.
`MutableBase`(*args, **kwargs)	Abstract Base Class for mutable dictionaries.

Data:

`KT`	Invariant `TypeVar`.
`T`	Invariant `TypeVar`.
`VT`	Invariant `TypeVar`.
`_D`	Invariant `TypeVar` bound to *`cawdrey.base.DictWrapper`*.

class DictWrapper

Bases: `Mapping`[*`~KT`*, *`~VT`*]

Abstract Mixin class for classes that wrap a dict object or similar.

Methods:

`__contains__`(key)	Return `key in self`.
`__getitem__`(key)	Return `self[key]`.
`__iter__`()	Iterates over the dictionary's keys.
`__len__`()	Returns the number of keys in the dictionary.
`__repr__`()	Return a string representation of the *`DictWrapper`*.
`copy`(*args, **kwargs)	Return a copy of the dictionary.
`get`(k[, default])	Return the value for `k` if `k` is in the dictionary, else `default`.
`items`()	Returns a set-like object providing a view on the dictionary's items.
`keys`()	Returns a set-like object providing a view on the dictionary's keys.
`values`()	Returns an object providing a view on the *`bdict`*'s values.

__contains__(*key*)
Return `key in self`.

Parameters **key** (`object`)

Return type `bool`

__getitem__(*key*)
Return `self[key]`.

Parameters **key** (*~KT*)

Return type *~VT*

__iter__()
Iterates over the dictionary's keys.

Return type `Iterator`[*~KT*]

__len__()
Returns the number of keys in the dictionary.

Return type `int`

__repr__()
Return a string representation of the *DictWrapper*.

Return type `str`

abstract copy(**args*, ***kwargs*)
Return a copy of the dictionary.

Return type *~_D*

get(*k*, *default=None*)
Return the value for `k` if `k` is in the dictionary, else `default`.

Parameters

- **k** – The key to return the value for.
- **default** – The value to return if `key` is not in the dictionary. Default `None`.

Overloads

- *get*(**k**: *~KT*) -> `Optional`[*~VT*]
- *get*(**k**: *~KT*, **default**: `Union`[*~VT*, *~T*]) -> `Union`[*~VT*, *~T*]

items()
Returns a set-like object providing a view on the dictionary's items.

Return type `AbstractSet`[`Tuple`[*~KT*, *~VT*]]

keys()
Returns a set-like object providing a view on the dictionary's keys.

Return type `AbstractSet`[*~KT*]

values()
Returns an object providing a view on the *bdict*'s values.

Return type ValuesView[*~VT*]

class FrozenBase(**args*, ***kwargs*)
Bases: *DictWrapper*[*~KT*, *~VT*]

Abstract Base Class for Frozen dictionaries.

Used by *frozendict* and *FrozenOrderedDict*.

Custom subclasses must implement at a minimum __init__, copy, fromkeys.

Methods:

fromkeys(iterable[, value])	Create a new dictionary with keys from iterable and values set to value.

classmethod fromkeys(*iterable*, *value=None*)
Create a new dictionary with keys from iterable and values set to value.

Return type *FrozenBase*[*~KT*, *~VT*]

Overloads

- *fromkeys*(**iterable**) -> *FrozenBase*[*~KT*, Any]
- *fromkeys*(**iterable**, **value**: *~VT*) -> *FrozenBase*[*~KT*, *~VT*]

KT = TypeVar(KT)
Type: TypeVar

Invariant TypeVar.

typing.TypeVar used for annotating key types in mappings.

class MutableBase(**args*, ***kwargs*)
Bases: *DictWrapper*[*~KT*, *~VT*], MutableMapping[*~KT*, *~VT*]

Abstract Base Class for mutable dictionaries.

Used by *NonelessDict* and *NonelessOrderedDict*.

Custom subclasses must implement at a minimum __init__, copy, fromkeys.

Methods:

__delitem__(key)	Delete self[key].
__setitem__(key, value)	Set self[key] to value.
fromkeys(iterable[, value])	Create a new dictionary with keys from iterable and values set to value.

__delitem__(*key*)
Delete self[key].

__setitem__(*key*, *value*)

Set `self[key]` to `value`.

classmethod fromkeys(*iterable*, *value=None*)
Create a new dictionary with keys from iterable and values set to value.

Return type *MutableBase*[*~KT*, *~VT*]

Overloads

- *fromkeys*(**iterable**) -> *MutableBase*[*~KT*, Any]
- *fromkeys*(**iterable**, **value**: *~VT*) -> *MutableBase*[*~KT*, *~VT*]

T = TypeVar(T)
Type: TypeVar

Invariant TypeVar.

VT = TypeVar(VT)
Type: TypeVar

Invariant TypeVar.

typing.TypeVar used for annotating value types in mappings.

_D = TypeVar(_D, bound=DictWrapper)
Type: TypeVar

Invariant TypeVar bound to *cawdrey.base.DictWrapper*.

Chapter

TWELVE

Functions

General utility functions.

Functions:

search_dict(dictionary, regex)	Return the subset of the dictionary whose keys match the regex.

search_dict (*dictionary*, *regex*)

Return the subset of the dictionary whose keys match the regex.

Parameters

- **dictionary** (Mapping[str, Any])
- **regex** (Union[str, Pattern])

Return type Dict[str, Any]

Chapter

THIRTEEN

Contributing

`Cawdrey` uses tox to automate testing and packaging, and pre-commit to maintain code quality.

Install `pre-commit` with `pip` and install the git hook:

```
$ python -m pip install pre-commit
$ pre-commit install
```

13.1 Coding style

formate is used for code formatting.

It can be run manually via `pre-commit`:

```
$ pre-commit run formate -a
```

Or, to run the complete autoformatting suite:

```
$ pre-commit run -a
```

13.2 Automated tests

Tests are run with `tox` and `pytest`. To run tests for a specific Python version, such as Python 3.6:

```
$ tox -e py36
```

To run tests for all Python versions, simply run:

```
$ tox
```

13.3 Type Annotations

Type annotations are checked using `mypy`. Run `mypy` using `tox`:

```
$ tox -e mypy
```

13.4 Build documentation locally

The documentation is powered by Sphinx. A local copy of the documentation can be built with `tox`:

```
$ tox -e docs
```

Chapter

FOURTEEN

Downloading source code

The `Cawdrey` source code is available on GitHub, and can be accessed from the following URL: https://github.com/domdfcoding/cawdrey

If you have `git` installed, you can clone the repository with the following command:

```
$ git clone https://github.com/domdfcoding/cawdrey
```

```
Cloning into 'cawdrey'...
remote: Enumerating objects: 47, done.
remote: Counting objects: 100% (47/47), done.
remote: Compressing objects: 100% (41/41), done.
remote: Total 173 (delta 16), reused 17 (delta 6), pack-reused 126
Receiving objects: 100% (173/173), 126.56 KiB | 678.00 KiB/s, done.
Resolving deltas: 100% (66/66), done.
```

Alternatively, the code can be downloaded in a 'zip' file by clicking:
Clone or download –> Download Zip

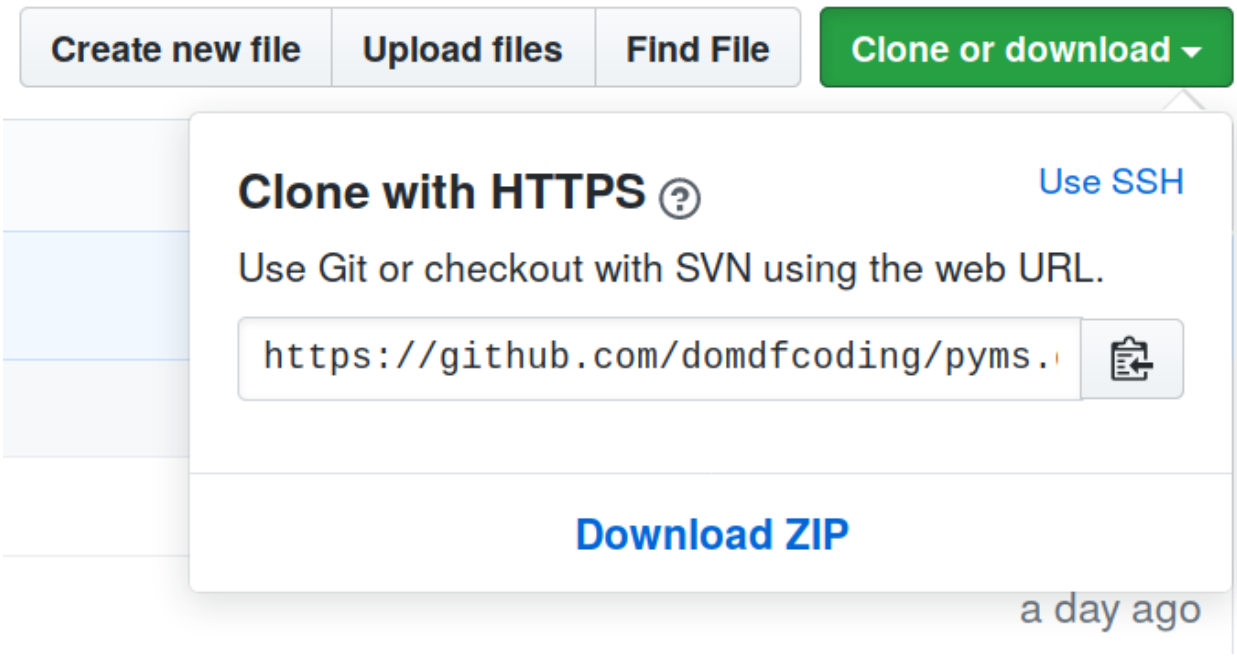

Fig. 1: Downloading a 'zip' file of the source code

14.1 Building from source

The recommended way to build `Cawdrey` is to use tox:

```
$ tox -e build
```

The source and wheel distributions will be in the directory `dist`.

If you wish, you may also use pep517.build or another **PEP 517**-compatible build tool.

Chapter

FIFTEEN

License

`Cawdrey` is licensed under the GNU Lesser General Public License v3.0

Permissions of this copyleft license are conditioned on making available complete source code of licensed works and modifications under the same license or the GNU GPLv3. Copyright and license notices must be preserved. Contributors provide an express grant of patent rights. However, a larger work using the licensed work through interfaces provided by the licensed work may be distributed under different terms and without source code for the larger work.

Permissions

- Commercial use – The licensed material and derivatives may be used for commercial purposes.
- Modification – The licensed material may be modified.
- Distribution – The licensed material may be distributed.
- Patent use – This license provides an express grant of patent rights from contributors.
- Private use – The licensed material may be used and modified in private.

Conditions

- License and copyright notice – A copy of the license and copyright notice must be included with the licensed material.
- Disclose source – Source code must be made available when the licensed material is distributed.
- State changes – Changes made to the licensed material must be documented.
- Same license (library) – Modifications must be released under the same license when distributing the licensed material. In some cases a similar or related license may be used, or this condition may not apply to works that use the licensed material as a library.

Limitations

- Liability – This license includes a limitation of liability.
- Warranty – This license explicitly states that it does NOT provide any warranty.

See more information on choosealicense.com ⇒

```
                   GNU LESSER GENERAL PUBLIC LICENSE
                       Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. <https://fsf.org/>
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.
```

(continues on next page)

(continued from previous page)

```
  This version of the GNU Lesser General Public License incorporates
the terms and conditions of version 3 of the GNU General Public
License, supplemented by the additional permissions listed below.

  0. Additional Definitions.

  As used herein, "this License" refers to version 3 of the GNU Lesser
General Public License, and the "GNU GPL" refers to version 3 of the GNU
General Public License.

  "The Library" refers to a covered work governed by this License,
other than an Application or a Combined Work as defined below.

  An "Application" is any work that makes use of an interface provided
by the Library, but which is not otherwise based on the Library.
Defining a subclass of a class defined by the Library is deemed a mode
of using an interface provided by the Library.

  A "Combined Work" is a work produced by combining or linking an
Application with the Library.  The particular version of the Library
with which the Combined Work was made is also called the "Linked
Version".

  The "Minimal Corresponding Source" for a Combined Work means the
Corresponding Source for the Combined Work, excluding any source code
for portions of the Combined Work that, considered in isolation, are
based on the Application, and not on the Linked Version.

  The "Corresponding Application Code" for a Combined Work means the
object code and/or source code for the Application, including any data
and utility programs needed for reproducing the Combined Work from the
Application, but excluding the System Libraries of the Combined Work.

  1. Exception to Section 3 of the GNU GPL.

  You may convey a covered work under sections 3 and 4 of this License
without being bound by section 3 of the GNU GPL.

  2. Conveying Modified Versions.

  If you modify a copy of the Library, and, in your modifications, a
facility refers to a function or data to be supplied by an Application
that uses the facility (other than as an argument passed when the
facility is invoked), then you may convey a copy of the modified
version:

   a) under this License, provided that you make a good faith effort to
   ensure that, in the event an Application does not supply the
   function or data, the facility still operates, and performs
   whatever part of its purpose remains meaningful, or

   b) under the GNU GPL, with none of the additional permissions of
   this License applicable to that copy.

  3. Object Code Incorporating Material from Library Header Files.
```

(continues on next page)

(continued from previous page)

```
  The object code form of an Application may incorporate material from
a header file that is part of the Library.  You may convey such object
code under terms of your choice, provided that, if the incorporated
material is not limited to numerical parameters, data structure
layouts and accessors, or small macros, inline functions and templates
(ten or fewer lines in length), you do both of the following:

   a) Give prominent notice with each copy of the object code that the
   Library is used in it and that the Library and its use are
   covered by this License.

   b) Accompany the object code with a copy of the GNU GPL and this license
   document.

  4. Combined Works.

  You may convey a Combined Work under terms of your choice that,
taken together, effectively do not restrict modification of the
portions of the Library contained in the Combined Work and reverse
engineering for debugging such modifications, if you also do each of
the following:

   a) Give prominent notice with each copy of the Combined Work that
   the Library is used in it and that the Library and its use are
   covered by this License.

   b) Accompany the Combined Work with a copy of the GNU GPL and this license
   document.

   c) For a Combined Work that displays copyright notices during
   execution, include the copyright notice for the Library among
   these notices, as well as a reference directing the user to the
   copies of the GNU GPL and this license document.

   d) Do one of the following:

       0) Convey the Minimal Corresponding Source under the terms of this
       License, and the Corresponding Application Code in a form
       suitable for, and under terms that permit, the user to
       recombine or relink the Application with a modified version of
       the Linked Version to produce a modified Combined Work, in the
       manner specified by section 6 of the GNU GPL for conveying
       Corresponding Source.

       1) Use a suitable shared library mechanism for linking with the
       Library.  A suitable mechanism is one that (a) uses at run time
       a copy of the Library already present on the user's computer
       system, and (b) will operate properly with a modified version
       of the Library that is interface-compatible with the Linked
       Version.

   e) Provide Installation Information, but only if you would otherwise
   be required to provide such information under section 6 of the
   GNU GPL, and only to the extent that such information is
   necessary to install and execute a modified version of the
   Combined Work produced by recombining or relinking the
```

(continues on next page)

(continued from previous page)

```
   Application with a modified version of the Linked Version. (If
   you use option 4d0, the Installation Information must accompany
   the Minimal Corresponding Source and Corresponding Application
   Code. If you use option 4d1, you must provide the Installation
   Information in the manner specified by section 6 of the GNU GPL
   for conveying Corresponding Source.)

  5. Combined Libraries.

  You may place library facilities that are a work based on the
Library side by side in a single library together with other library
facilities that are not Applications and are not covered by this
License, and convey such a combined library under terms of your
choice, if you do both of the following:

   a) Accompany the combined library with a copy of the same work based
   on the Library, uncombined with any other library facilities,
   conveyed under the terms of this License.

   b) Give prominent notice with the combined library that part of it
   is a work based on the Library, and explaining where to find the
   accompanying uncombined form of the same work.

  6. Revised Versions of the GNU Lesser General Public License.

  The Free Software Foundation may publish revised and/or new versions
of the GNU Lesser General Public License from time to time. Such new
versions will be similar in spirit to the present version, but may
differ in detail to address new problems or concerns.

  Each version is given a distinguishing version number. If the
Library as you received it specifies that a certain numbered version
of the GNU Lesser General Public License "or any later version"
applies to it, you have the option of following the terms and
conditions either of that published version or of any later version
published by the Free Software Foundation. If the Library as you
received it does not specify a version number of the GNU Lesser
General Public License, you may choose any version of the GNU Lesser
General Public License ever published by the Free Software Foundation.

  If the Library as you received it specifies that a proxy can decide
whether future versions of the GNU Lesser General Public License shall
apply, that proxy's public statement of acceptance of any version is
permanent authorization for you to choose that version for the
Library.
```

Chapter

SIXTEEN

And Finally:

Why “Cawdrey”?

Python Module Index

C

Index

Symbols

A

B

C

D

F

G

H

I

K

M

N

P

S

T

V