cawdrey

Release 0.1.6

Dominic Davis-Foster

Jun 18, 2020

CONTENTS

Several useful custom dictionaries for Python

Docs	
Tests	
PyPI	
Anaconda	
Activity	
Other	

ONE

CONTENTS

- `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.

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.

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 OrederedDict 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**.

INSTALLATION

from PyPI

```
$ pip install cawdrey
```

from Anaconda

First add the required channels

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

Then install

```
$ conda install cawdrey
```

from GitHub

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

3.1 AlphaDict

3.1.1 About

3.1.2 Usage

3.1.3 API Reference

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

class cawdrey.alphadict.**AlphaDict**(*seq=None*, ***kwargs*)

cawdrey.alphadict.**alphabetical_dict**(***kwargs*)

3.2 bdict

3.2.1 About

3.2.2 Usage

3.2.3 API Reference

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

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

Based on https://stackoverflow.com/a/1063393 by https://stackoverflow.com/users/9493/brian

Improved May 2020 suggestions from https://treyhunner.com/2019/04/why-you-shouldnt-inherit-from-list-and-dict-in-python/

3.3 frozendict

3.3.1 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 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.

3.3.2 Usage

```
>>> from frozendict 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 frozendict 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'
>>> 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'
```

3.3.3 API Reference

class cawdrey.**frozendict**(**args*, ***kwds*)

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

copy(***add_or_replace*)

dict_cls

alias of builtins.dict

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.

3.3.4 Copyright

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

Copyright (c) 2012 Santiago Lezica

Licensed under the MIT License:

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Also based on https://github.com/Marco-Sulla/python-frozendict

Copyright (c) Marco Sulla

Licensed under the GNU Lesser General Public License Version 3

3.4 FrozenOrderedDict

3.4.1 About

FrozenOrderedDict is a immutable wrapper around an OrderedDict.

FrozenOrderedDict 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.

It does implement the `Mapping` interface, so can be used just 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.

3.4.2 API Reference

class cawdrey.frozenordereddict.**FrozenOrderedDict**(**args*, ***kwds*)
> An immutable OrderedDict. It can be used as a drop-in replacement for dictionaries where immutability is desired.

> **copy**(**args*, ***kwargs*)

> **dict_cls**
> > alias of `collections.OrderedDict`

3.4.3 Copyright

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

Copyright (c) 2012 Santiago Lezica

Licensed under the MIT License:

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Also based on https://github.com/Marco-Sulla/python-frozendict Copyright (c) Marco Sulla Licensed under the GNU Lesser General Public License Version 3

3.5 NonelessDict

3.5.1 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 *NonelessDict* and `OrderedDict`, so the order of key insertion is preserved.

3.5.2 Usage

3.5.3 API Reference

Provides frozendict, a simple immutable dictionary.

class cawdrey.nonelessdict.**NonelessDict**(**args*, ***kwds*)

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 function to check only for None

copy(***add_or_replace*)

dict_cls

alias of `builtins.dict`

> **set_with_strict_none_check**(*key*, *value*)
>
> > **Return type** None

class cawdrey.nonelessdict.**NonelessOrderedDict**(**args*, ***kwds*)

> 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
>
> **copy**(**args*, ***kwargs*)
>
> **dict_cls**
>
> > alias of collections.OrderedDict
>
> **set_with_strict_none_check**(*key*, *value*)
>
> > **Return type** None

3.5.4 Copyright

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

Copyright (c) 2012 Santiago Lezica

Licensed under the MIT License:

3.6 Base Class

3.6.1 About

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

3.6.2 Usage

3.6.3 API Reference

class `cawdrey.base.`**`FrozenBase`**(**args*, ***kwds*)

    Abstract Base Class for Frozen dictionaries

    Used by frozendict and FrozenOrderedDict.

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

    `__abstractmethods__` **`= frozenset({'__init__', 'copy'})`**

    `__annotations__` **`= {'dict_cls': typing.Union[typing.Type, NoneType]}`**

    `__args__` **`= None`**

    `__contains__`(*key*)

        Return type `Any`

    `__copy__`(**args*, ***kwargs*)

    `__dict__` **`= mappingproxy({'__module__': 'cawdrey.base', '__annotations__': {'dict_cls`**

    `__eq__`(*other*)

        Return self==value.

    `__extra__` **`= None`**

    `__getitem__`(*key*)

        Return type `Any`

    `__hash__` **`= None`**

    abstract **`__init__`**(**args*, ***kwargs*)

        Initialize self. See help(type(self)) for accurate signature.

    `__iter__`()

    `__len__`()

        Return type `int`

    `__module__` **`= 'cawdrey.base'`**

    static **`__new__`**(*cls*, **args*, ***kwds*)

        Create and return a new object. See help(type) for accurate signature.

    `__next_in_mro__`

        alias of `builtins.object`

    `__orig_bases__` **`= (cawdrey.base.DictWrapper[~KT, ~VT],)`**

    `__origin__` **`= None`**

    `__parameters__` **`= (~KT, ~VT)`**

    `__repr__`()

        Return repr(self).

        Return type `str`

    `__reversed__` **`= None`**

    `__slots__` **`= ()`**

__subclasshook__()
>   Abstract classes can override this to customize issubclass().

>   This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__tree_hash__ = -9223366114991902158

__weakref__
>   list of weak references to the object (if defined)

_abc_cache = <_weakrefset.WeakSet object>

_abc_generic_negative_cache = <_weakrefset.WeakSet object>

_abc_generic_negative_cache_version = 42

_abc_negative_cache = <_weakrefset.WeakSet object>

_abc_negative_cache_version = 42

_abc_registry = <_weakrefset.WeakSet object>

_gorg
>   alias of *FrozenBase*

abstract copy(**args*, ***kwargs*)

dict_cls: Optional[Type] = None

classmethod fromkeys(**args*, ***kwargs*)
>   Returns a new dict with keys from iterable and values equal to value.

get($k[, d]$) $\rightarrow$ D[k] if k in D, else d. d defaults to None.

items() $\rightarrow$ a set-like object providing a view on D's items

keys() $\rightarrow$ a set-like object providing a view on D's keys

values() $\rightarrow$ an object providing a view on D's values

3.7 Downloading source code

The `cawdrey` source code resides on publicly accessible GitHub servers, 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:
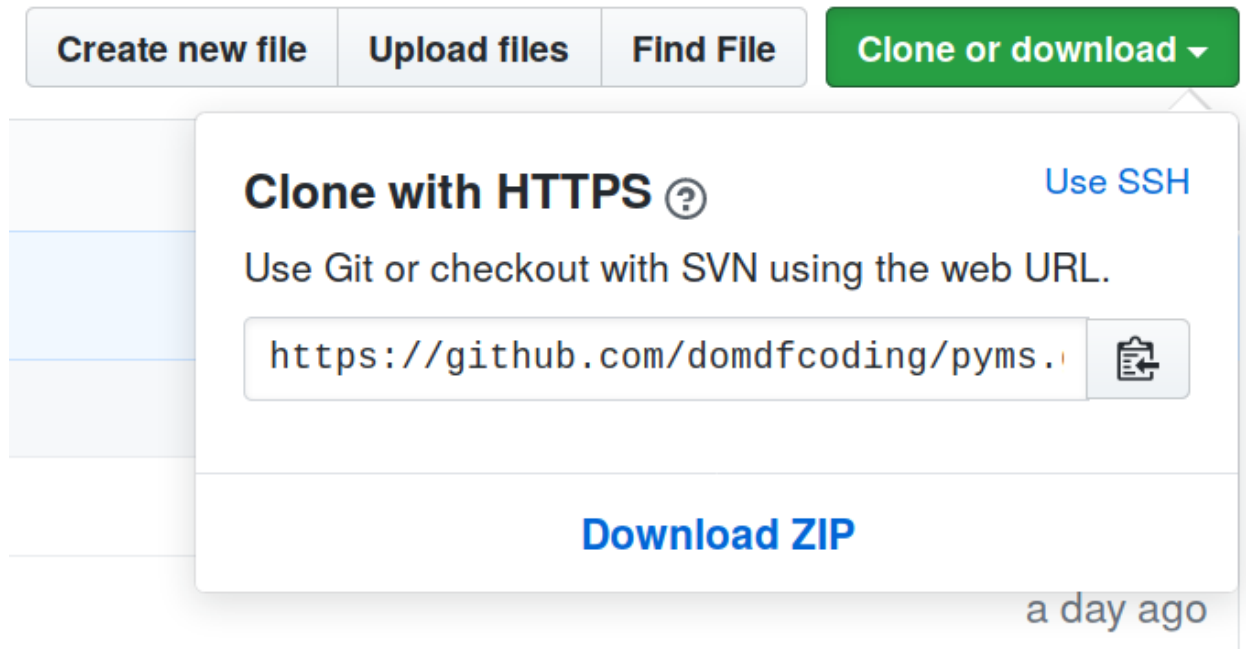
Clone or download –> Download Zip

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

3.8 Building from source

To build the `cawdrey` package from source using `setuptools`, run the following command:

```
$ python3 setup.py sdist bdist_wheel
```

`setuptools` is configured using the file `setup.py`.

Different formats are available for built distributions

Format	Description	Notes
gztar	gzipped tar file (`.tar.gz`)	default on Unix
bztar	bzipped tar file (`.tar.bz2`)	
xztar	bzipped tar file (`.tar.bz2`)	
tar	tar file (`.tar`)	
zip	zip file (`.zip`)	default on Windows
wininst	self-extracting ZIP file for Windows	
msi	Microsoft Installer	

setup.py

```
1  #!/usr/bin/env python
2  # This file is managed by `git_helper`. Don't edit it directly
```

(continues on next page)

```python
3   """Setup script"""
4
5   # 3rd party
6   from setuptools import find_packages, setup
7
8   # this package
9   from __pkginfo__ import *  # pylint: disable=wildcard-import
10
11
12
13  setup(
14                  author=author,
15                  author_email=author_email,
16                  classifiers=classifiers,
17                  description=short_desc,
18                  entry_points=entry_points,
19                  extras_require=extras_require,
20                  include_package_data=True,
21                  install_requires=install_requires,
22                  license=__license__,
23                  long_description=long_description,
24                  name=pypi_name,
25                  packages=find_packages(exclude=("tests", "doc-source")),
26                  project_urls=project_urls,
27                  py_modules=py_modules,
28                  python_requires=">=3.6",
29                  url=web,
30                  version=__version__,
31                  keywords=keywords,
32                  zip_safe=False,
33
34                  )
```

__pkginfo__.py

```python
1   #  This file is managed by `git_helper`. Don't edit it directly
2   #  Copyright (C) 2020 Dominic Davis-Foster <dominic@davis-foster.co.uk>
3   #
4   #  This file is distributed under the same license terms as the program it came with.
5   #  There will probably be a file called LICEN[S/C]E in the same directory as this
    →file.
6   #
7   #  In any case, this program is distributed in the hope that it will be useful,
8   #  but WITHOUT ANY WARRANTY; without even the implied warranty of
9   #  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
10  #
11  # This script based on https://github.com/rocky/python-uncompyle6/blob/master/__
    →pkginfo__.py
12  #
13
14  # stdlib
15  import pathlib
16
17  __all__ = [
18                  "__copyright__",
19                  "__version__",
20                  "modname",
```

```
21                    "pypi_name",
22                    "py_modules",
23                    "entry_points",
24                    "__license__",
25                    "short_desc",
26                    "author",
27                    "author_email",
28                    "github_username",
29                    "web",
30                    "github_url",
31                    "project_urls",
32                    "repo_root",
33                    "long_description",
34                    "install_requires",
35                    "extras_require",
36                    "classifiers",
37                    "keywords",
38                    "import_name",
39                    ]
40
41  __copyright__ = """
42  2019-2020 Dominic Davis-Foster <dominic@davis-foster.co.uk>
43  """
44
45  __version__ = "0.1.6"
46
47  modname = "cawdrey"
48  pypi_name = "cawdrey"
49  import_name = "cawdrey"
50  py_modules = []
51  entry_points = {
52                    "console_scripts": []
53                    }
54
55  __license__ = "GNU Lesser General Public License v3 or later (LGPLv3+)"
56
57  short_desc = "Several useful custom dictionaries for Python "
58
59  __author__ = author = "Dominic Davis-Foster"
60  author_email = "dominic@davis-foster.co.uk"
61  github_username = "domdfcoding"
62  web = github_url = f"https://github.com/domdfcoding/cawdrey"
63  project_urls = {
64                    "Documentation": f"https://cawdrey.readthedocs.io",
65                    "Issue Tracker": f"{github_url}/issues",
66                    "Source Code": github_url,
67                    }
68
69  repo_root = pathlib.Path(__file__).parent
70
71  # Get info from files; set: long_description
72  long_description = (repo_root / "README.rst").read_text(encoding="utf-8").replace("0.
    ↪1.6", __version__) + '\n'
73  conda_description = """Several useful custom dictionaries
74
75
76  Before installing please ensure you have added the following channels: domdfcoding,␣
    ↪conda-forge"""
```

```
77  __all__.append("conda_description")
78
79  install_requires = (repo_root / "requirements.txt").read_text(encoding="utf-8").split(
    ↪'\n')
80  extras_require = {'all': []}
81
82  classifiers = [
83                  'Development Status :: 3 - Alpha',
84                  'Intended Audience :: Developers',
85                  'License :: OSI Approved :: GNU Lesser General Public License v3 or␣
    ↪later (LGPLv3+)',
86                  'Operating System :: OS Independent',
87                  'Programming Language :: Python',
88                  'Programming Language :: Python :: 3.6',
89                  'Programming Language :: Python :: 3.7',
90                  'Programming Language :: Python :: 3.8',
91                  'Programming Language :: Python :: 3 :: Only',
92                  'Programming Language :: Python :: Implementation :: CPython',
93                  'Programming Language :: Python :: Implementation :: PyPy',
94                  'Topic :: Software Development :: Libraries :: Python Modules',
95                  'Topic :: Utilities',
96                  'Programming Language :: Python :: 3.9',
97                  'Typing :: Typed',
98
99                  ]
100
101  keywords = "frozenordereddict orderedfrozendict frozen immutable frozendict dict␣
    ↪dictionary map Mapping MappingProxyType"
```

View the Function Index or browse the Source Code.

Browse the GitHub Repository

AND FINALLY:

Why "Cawdrey"?

PYTHON MODULE INDEX

C