cawdrey

Release 0.1.3

Dominic Davis-Foster

May 20, 2020

Contents

A collection of useful custom dictionaries for Python.

CHAPTER 1

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.

1.1 Other Dictionary Packages

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

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

to this:

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**.

1.2 And Finally:

Why Cawdrey?

`cawdrey` can be installed with pip:

```
$ pip install cawdrey
```

Browse the Source Code.

Browse the GitHub Repository

1.2.1 AlphaDict

About

Usage

API Reference

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

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

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

dict_cls
alias of `collections.OrderedDict`

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

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

values() → an object providing a view on D's values

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

1.2.2 bdict

About

Usage

API Reference

class cawdrey.bdict.**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

clear() → None. Remove all items from D.

copy()

classmethod fromkeys(*iterable*, *value=None*)

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

values() → an object providing a view on D's values

1.2.3 frozendict

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.

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:

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 {}>
```

(continues on next page)

(continued from previous page)

```
>>> 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'
```

API Reference

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

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`

classmethod fromkeys(**args*, ***kwargs*)

Returns a new dict with keys from iterable and values equal to value.

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

sorted(**args*, *by='keys'*, ***kwargs*)

Return a new *frozendict*, with the element insertion sorted. The signature is the same of 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, see the documentation of *sorted()*. Take into mind that the parameters passed to the *key* function are the keys of the *frozendict* if *by == "keys"*, and are the items otherwise.

PS: Note that sort by keys and items are identical. The only difference is when you want to customize the sorting passing a custom *key* function. You *could* achive 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.

values() → an object providing a view on D's values

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

1.2.4 FrozenOrderedDict

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.

API Reference

class cawdrey.frozenordereddict.**FrozenOrderedDict**(**args*, ***kwargs*)

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

classmethod fromkeys(**args*, ***kwargs*)

Returns a new dict with keys from iterable and values equal to value.

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

values() → an object providing a view on D's values

Copyright

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

Also based on https://github.com/wsmith323/frozenordereddict

Copyright (c) 2015 Warren Smith

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.

1.2.5 NonelessDict

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.

Usage

API Reference

Provides frozendict, a simple immutable dictionary.

class cawdrey.nonelessdict.**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. Use the set_with_strict_none_check function to check only for None

clear() → None. Remove all items from D.

copy(***add_or_replace*)

dict_cls
alias of builtins.dict

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

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

set_with_strict_none_check(*key*, *value*)

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

values() → an object providing a view on D's values

class cawdrey.nonelessdict.**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. Use the set_with_strict_none_check function to check only for None

clear() → None. Remove all items from D.

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

dict_cls
alias of collections.OrderedDict

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

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

set_with_strict_none_check(*key*, *value*)

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

values() → an object providing a view on D's values

Copyright

1.2.6 Base Class

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.

Usage

API Reference

class cawdrey.base.**FrozenBase**(**args*, ***kwargs*)

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'})

__class__
alias of `abc.ABCMeta`

__contains__(*key*)

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

__delattr__
Implement delattr(self, name).

__dict__ = mappingproxy({'__module__': 'cawdrey.base', '__doc__': '\n\tAbstract Base

__dir__() → list
default dir() implementation

__eq__(*other*)
Return self==value.

__format__()
default object formatter

__ge__
Return self>=value.

__getattribute__
Return getattr(self, name).

__getitem__(*key*)

__gt__
Return self>value.

__hash__ = None

__init__(**args*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

__init_subclass__()
This method is called when a class is subclassed.

The default implementation does nothing. It may be overridden to extend subclasses.

__iter__()

__le__
Return self<=value.

__len__()

__lt__
Return self<value.

__module__ = 'cawdrey.base'

__ne__
Return self!=value.

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

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__()
Return repr(self).

__reversed__ = None

__setattr__
Implement setattr(self, name, value).

__sizeof__() → int
size of object in memory, in bytes

__slots__ = ()

__str__
Return str(self).

classmethod __subclasshook__(*C*)
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).

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

_abc_cache = <_weakrefset.WeakSet object>

_abc_negative_cache = <_weakrefset.WeakSet object>

_abc_negative_cache_version = 42

_abc_registry = <_weakrefset.WeakSet object>

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

dict_cls = None

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

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

values() → an object providing a view on D's values

1.2.7 Downloading source code

`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

1.2.8 Building from source

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

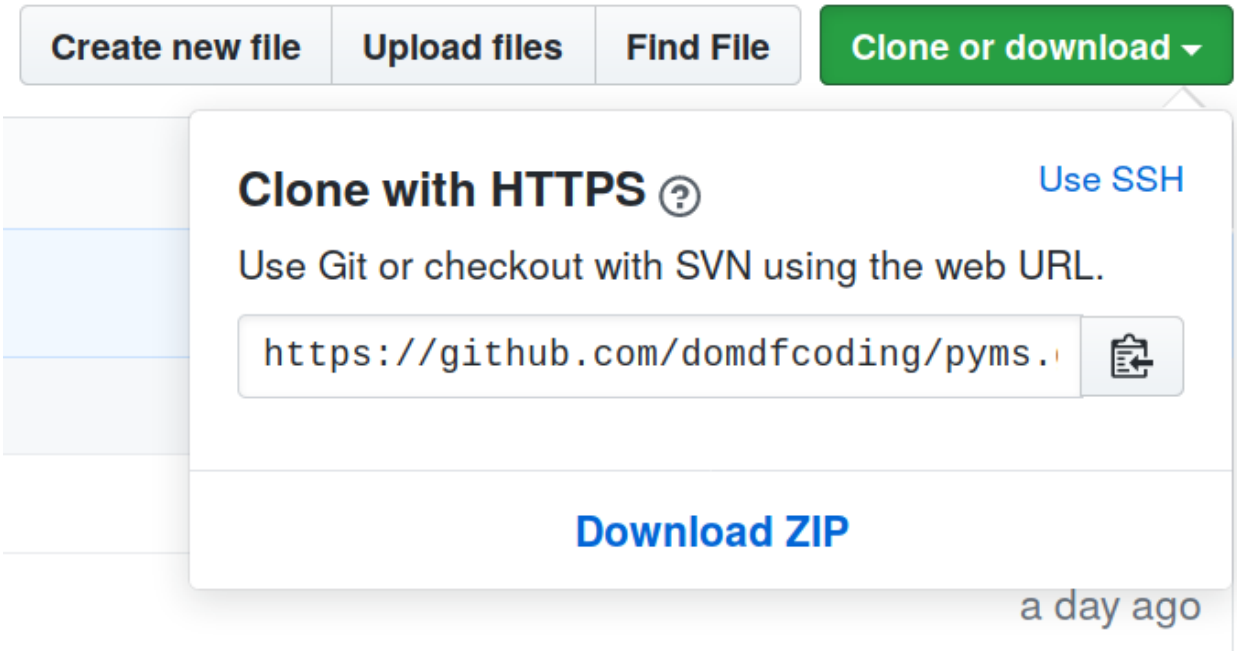

Fig. 1: Downloading a 'zip' file of the source code

```
$ 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

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

from __pkginfo__ import *

from setuptools import setup, find_packages

```

(continues on next page)

(continued from previous page)

```
setup(
                author=author,
                author_email=author_email,
                classifiers=classifiers,
                description=short_desc,
                entry_points=entry_points,
                extras_require=extras_require,
                include_package_data=True,
                install_requires=install_requires,
                license=license,
                long_description=long_description,
                name=modname,
                packages=find_packages(exclude=("tests", "doc-source")),
                project_urls=project_urls,
                py_modules=py_modules,
                python_requires=">=3.6",
                url=web,
                version=VERSION,
                keywords=keywords,

                )
```

__pkginfo__.py

```
# This file is managed by `git_helper`. Don't edit it directly
# Copyright (C) 2019-2020 Dominic Davis-Foster <dominic@davis-foster.co.uk>
#
#  This program is free software: you can redistribute it and/or modify
#  it under the terms of the GNU General Public License as published by
#  the Free Software Foundation, either version 3 of the License, or
#  (at your option) any later version.
#
#  This program is distributed in the hope that it will be useful,
#  but WITHOUT ANY WARRANTY; without even the implied warranty of
#  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#  GNU General Public License for more details.
#
#  You should have received a copy of the GNU General Public License
#  along with this program.  If not, see <http://www.gnu.org/licenses/>.

# This script based on https://github.com/rocky/python-uncompyle6/blob/master/__
↪pkginfo__.py

import pathlib

copyright = """
2019-2020 Dominic Davis-Foster <dominic@davis-foster.co.uk>
"""

VERSION = "0.1.3"

modname = "cawdrey"
py_modules = []
entry_points = None

license = 'LGPLv3+'

```

(continues on next page)

(continued from previous page)

```
short_desc = 'Several useful custom dictionaries'

author = "Dominic Davis-Foster"
author_email = "dominic@davis-foster.co.uk"
github_username = "domdfcoding"
web = github_url = f"https://github.com/domdfcoding/cawdrey"
project_urls = {
                "Documentation": f"https://cawdrey.readthedocs.io",  # TODO: Make this link match the package version
                "Issue Tracker": f"{github_url}/issues",
                "Source Code": github_url,
                }

repo_root = pathlib.Path(__file__).parent

# Get info from files; set: long_description
long_description = (repo_root / "README.rst").read_text() + '\n'
conda_description = """Several useful custom dictionaries


Before installing please ensure you have added the following channels: domdfcoding, conda-forge"""
install_requires = (repo_root / "requirements.txt").read_text().split('\n')
extras_require = {'all': []}

classifiers = [
                'Development Status :: 3 - Alpha',
                'Intended Audience :: Developers',
                'License :: OSI Approved :: GNU Lesser General Public License v3 or later (LGPLv3+)',
                'Operating System :: OS Independent',
                'Programming Language :: Python',
                'Programming Language :: Python :: 3.6',
                'Programming Language :: Python :: 3.7',
                'Programming Language :: Python :: 3.8',
                'Programming Language :: Python :: 3 :: Only',
                'Programming Language :: Python :: Implementation :: CPython',
                'Programming Language :: Python :: Implementation :: PyPy',
                'Topic :: Software Development :: Libraries :: Python Modules',
                'Topic :: Utilities',

                ]

keywords = "ordereddict frozenordereddict orderedfrozendict ordered frozen immutable frozendict dict dictionary map Mapping MappingProxyType developers"
```

Python Module Index

C

Index

Symbols

A

B

C

D

F

G

I

K

N

P

S

U

V