fancy-dict Documentation

Stefan Hoelzl

Jul 09, 2018

Contents:

CHAPTER 1

fancy-dict

Extends python dictionaries with merge, load and filter functions

1.1 Key Features

- Load data from various sources (dicts, Files, HTTP)
- Customize the merging behavior on `update()`
- Filter data of dictionaries

Currently only tested on Python 3.6

1.2 Installation

```
$ pip install git+https://github.com/stefanhoelzl/fancy-dict.git
```

1.3 Usage

Basics

```
>>> from fancy_dict import FancyDict

# Load data form GitHub-API
>>> repo = FancyDict.load("https://api.github.com/repos/stefanhoelzl/fancy-dict")

# Access like plain python dicts
```

(continues on next page)

(continued from previous page)

```
>>> repo["owner"]["avatar_url"]
'https://avatars0.githubusercontent.com/u/1478183?v=4'

# Nested updates
>>> repo.update({"owner": {"avatar_url": "https://avatars0.githubusercontent.com/u/
↪254659"}})

# Other values in repo["owner"] are still present
>>> repo["owner"]["html_url"]
'https://github.com/stefanhoelzl'
```

Load and merge annotated yaml/json files.

```
# Create directories later needed
>>> import os
>>> os.makedirs("inc")

# Import used fancy_dict classes
>>> from fancy_dict import FancyDict
>>> from fancy_dict.loader import KeyAnnotationsConverter

# write settings defaults
>>> with open("inc/base.yml", "w+") as base_file:
...     base_file.write('{"counter[add]": 0, "settings": {"skip": True}}')
47

# write custom settings
>>> with open("config.yml", "w+") as config_file:
...     config_file.write('{"include": ["base.yml"], "counter": 1, "settings": {"+skip
↪": False, "?merge": True}}')
85

# merge custom and default settings
>>> FancyDict.load("config.yml", include_paths=("inc",), include_key="include",␣
↪annotations_decoder=KeyAnnotationsConverter)
{'counter': 1, 'settings': {'skip': True}}
```

Annotate keys to control updating behavior

```
>>> from fancy_dict import FancyDict
>>> from fancy_dict.merger import add
>>> from fancy_dict.conditions import if_existing, if_not_existing

# Set a custom merge method (defines how old and new value get merged)
>>> annotated_dict = FancyDict({"counter": 0})

# sets an annotation that the key "counter" should be updated by adding old and new␣
↪value
>>> annotated_dict.annotate("counter", merge_method=add)
>>> annotated_dict.update({"counter": 1})
>>> annotated_dict["counter"]
1
>>> annotated_dict.update({"counter": 1})
>>> annotated_dict["counter"]
2
```

(continues on next page)

(continued from previous page)

```
# Finalizes a value for a given key, so updates dont change this value
>>> annotated_dict.annotate("counter", finalized=True)
>>> annotated_dict.update({"counter": 1})
>>> annotated_dict["counter"]
2

# direct changes of this key are still possible
>>> annotated_dict["counter"] = 0
>>> annotated_dict["counter"]
0

# set annotations so that updates only apply under certain conditions
>>> annotated_dict.annotate("not_existing", condition=if_existing)
>>> annotated_dict.update({"not_existing": False})
>>> annotated_dict.keys()
dict_keys(['counter'])

>>> annotated_dict["not_existing"] = False
>>> annotated_dict.update({"not_existing": True})
>>> annotated_dict["not_existing"]  # value was updated, because it was existing␣
↪before
True

# same for if_not_existing condition
>>> annotated_dict.annotate("existing", condition=if_not_existing)
>>> annotated_dict["existing"] = False
>>> annotated_dict.update({"existing": True})
>>> annotated_dict["existing"]
False
>>> del annotated_dict["existing"]
>>> annotated_dict.update({"existing": True})
>>> annotated_dict["existing"]
True
```

1.4 Development status

Alpha

working towards the first release and better documentation!

1.5 Documentation

http://fancy-dict.readthedocs.io/

1.6 Contribution

- **There is a bug?** Write an Issue
- **Feedback or Questions?** Write an Issue

- **You like it?** Great!! Spread the news :)

1.6.1 Submit changes

tested on macOS and Linux (will be different on Windows)

Fork it and check out:

```
$ git checkout https://github.com/<YourUsername>/fancy-dict.git
```

setup the development environment

```
$ cd fancy-dict
$ virtualenv venv
$ source venv/bin/activate
$ make env.install
```

Write a failing test, make your changes until all tests pass

```
$ make tests           # runs all tests
$ make tests.unit      # runs only unit tests
$ make tests.lint       # runs only linter
$ make tests.coverage  # runs only code coverage
```

Before making a pull request, check if still everythinig builds

```
$ make       # runs all tests, builds the docs and creates a package
$ make clean # cleans the repository
```

Create a pull request!

1.7 License

This project is licensed under the MIT License - see the LICENSE file for details

CHAPTER 2

API

Submodules

2.1 FancyDict

Dictionary extended load/update/filter features.

Loads data from different sources using Loaders. Updates data with customizeable MergeMethods. Queries data using Transformations.

class fancy_dict.fancy_dict.**FancyDict**(*_FancyDict__dct=None*, ***kwargs*)

Bases: dict

Extends dict by merge methods, filter and load functionality.

Merging methods can define custom behavior how to merge certain values in the dict.

Conditions can prevent merging a value under certain circumstances.

Keys can be marked as finalizedd to avoid future updates.

Queries allow it to retrieve values deep inside the dict.

Loader allow it to load data from various sources.

MERGE_METHODS = (<fancy_dict.merger.MergeMethod object>, <fancy_dict.merger.MergeMetho

annotate(*key*, *annotations=None*, ***kwargs*)

Adds Annotations for specific key.

Parameters

- **key** – name of the key
- **annotations** – Annotations object with the annotions to add
- ****kwargs** – arguments used to create an Annotations (optional)

Returns:

filter(*filter_method*, *recursive=False*, *flat=False*)
Returns a filtered FancyDict

filter_method must be a method with two parameters. filter_method returns True or False for a given pair of key/value. If filter_method returns True, the key/value pair is added to the filtered dict.

Parameters

- **filter_method** – determines if key/value pair gets into return
- **recursive** – searches recursive into sub dicts
- **flat** – if recursive, flattens the result

Returns FancyDict with filtered content

get_annotations(*key*, *default=None*)
Gets the Annotations for a key.

A default value is returned if no annotations are set for the key.

Parameters

- **key** – name of the key.
- **default** – return value if no annotations for this key specified.

Returns Annotations for this key or default.

classmethod load(*source*, *annotations_decoder=None*, *loader=<class 'fancy_dict.loader.CompositeLoader'>*, ***loader_kwargs*)
Loads FancyDicts from different sources.

Parameters

- **source** – Source specifier
- **annotations_decoder** – Decoder used for annotations
- **loader** – Loader class used to load from the given source
- ****loader_kwargs** – Arguments for the Loader

Returns FancyDict with initialized data from given source

update(*_FancyDict__dct=None*, ***kwargs*)
Updates the data using MergeMethods and Annotations

When updating with a plain dict, they get first converted to FancyDicts

First key specific annotations get evaluated for each key to check if and how the value for this key can be updated.

They are evaluated in the following order. 1. When a key is finalized, the value never gets updated. 2. The condition annotation based on old and new value gets evaluated * the condition of the destination is used * if there is none, the condition of the source is used * if there is none, the default condition is used * if the condition is false, the value gets not updated

If the value can be updated, the merge method is looked up the in the following order: 1. merge method annotated in source 2. merge method annotated in destination 3. global merge methods * first the source merge methods are evaluated * second the destination merge methods are evaluated * the first merge method which applies to the old and new value is used.

Parameters

- **__dct** – source dict to merge into destination (self)
- ****kwargs** – key-value-pairs for source dict

Raises NoMergeMethodApplies if no valid MergeStrategy was found.

2.2 Loader

Loader and Dumper to serialize FancyDicts

class fancy_dict.loader.**AnnotationsDecoder**
Bases: object

Interface to decode annotations

classmethod decode(*key=None*, *value=None*)
Decodes an annotation from a key/value-pair

Parameters

- **key** – can be used to decode annotation
- **value** – can be used to decode annotation

Returns

- key: decoded key
- value: decoded value
- decoded: annotation

Return type dict with the following keys

class fancy_dict.loader.**AnnotationsEncoder**
Bases: object

Interface to encode annotations

classmethod encode(*annotation*, *key=None*, *value=None*)
Encodes an annotation into a key/value-pair

Parameters

- **annotation** – annotation to encode
- **key** – initial key value to encode annotation into
- **value** – initial value to encode annotation into

Returns

- key: key with annotation encoded
- value: value with annotation encoded

Return type dict with the following keys

class fancy_dict.loader.**CompositeLoader**(*output_type*, ***loader_args*)
Bases: *fancy_dict.loader.LoaderInterface*

Composition of different Loader

Selects the right Loader for the source.

Can load from dicts and yaml/json files.

LOADER = [<class 'fancy_dict.loader.DictLoader'>, <class 'fancy_dict.loader.IoLoader'>

classmethod can_load(*source*)
Checks if the loader can load the given source

Parameters **source** – source to load

Returns True if Loader can load the source else False

load(*source*, *annotations_decoder=None*)
Loads a FancyDict from a given source

If an annotations_decoder is given, Annotations can be decoded from the source data.

Parameters

- **source** – source to load from
- **annotations_decoder** – Decoder to decode annotations from source data

Returns FancyDict

class fancy_dict.loader.**DictLoader**(*output_type*)
Bases: *fancy_dict.loader.LoaderInterface*

Loads a dict as FancyDict

classmethod can_load(*source*)
Checks if the loader can load the given source

Parameters **source** – source to load

Returns True if Loader can load the source else False

load(*source*, *annotations_decoder=None*)
Loads a FancyDict from a given source

If an annotations_decoder is given, Annotations can be decoded from the source data.

Parameters

- **source** – source to load from
- **annotations_decoder** – Decoder to decode annotations from source data

Returns FancyDict

class fancy_dict.loader.**FileLoader**(*output_type*, *include_paths=('.',)*, *include_key=None*)
Bases: *fancy_dict.loader.IoLoader*

Loads a FancyDict from a YAML/JSON file

Looks up files in given base directoies. Supports a special include key to include other files.

DEFAULT_INCLUDE_PATHS = ('.',)

classmethod can_load(*source*)
Checks if the loader can load the given source

Parameters **source** – source to load

Returns True if Loader can load the source else False

load(*source*, *annotations_decoder=None*)
Loads a FancyDict from a given source

If an annotations_decoder is given, Annotations can be decoded from the source data.

Parameters

- **source** – source to load from

- **annotations_decoder** – Decoder to decode annotations from source data

Returns FancyDict

class fancy_dict.loader.**HttpLoader**(*output_type*)
Bases: *fancy_dict.loader.IoLoader*

Loads YAML/JSON files from an URL

classmethod can_load(*source*)
Checks if the loader can load the given source

Parameters **source** – source to load

Returns True if Loader can load the source else False

load(*source*, *annotations_decoder=None*)
Loads a FancyDict from a given source

If an annotations_decoder is given, Annotations can be decoded from the source data.

Parameters

- **source** – source to load from
- **annotations_decoder** – Decoder to decode annotations from source data

Returns FancyDict

class fancy_dict.loader.**IoLoader**(*output_type*)
Bases: *fancy_dict.loader.DictLoader*

Loads a FancyDict from an IO-like object

classmethod can_load(*source*)
Checks if the loader can load the given source

Parameters **source** – source to load

Returns True if Loader can load the source else False

load(*source*, *annotations_decoder=None*)
Loads a FancyDict from a given source

If an annotations_decoder is given, Annotations can be decoded from the source data.

Parameters

- **source** – source to load from
- **annotations_decoder** – Decoder to decode annotations from source data

Returns FancyDict

class fancy_dict.loader.**KeyAnnotationsConverter**
Bases: *fancy_dict.loader.AnnotationsEncoder*, *fancy_dict.loader.AnnotationsDecoder*

Encodes/Decodes an Annotations from the key

The following format is used to encode and decode annotation strings: ?(key)[add] * The first character defines the condition (optional) * Followed by the key name * if the key is in round brackets, the key gets finalized (optional) * at the end the merge method can be specified in square brackets

CONDITIONS = {'#': <function always at 0x7ff1135901e0>, '+': <function if_not_existi

MERGE_METHODS = {'add': <function add at 0x7ff113693e18>, 'overwrite': <function ove

classmethod decode(*key=None*, *value=None*)
Decodes an annotation from a key/value-pair

Parameters

- **key** – can be used to decode annotation
- **value** – can be used to decode annotation

Returns

- key: decoded key
- value: decoded value
- decoded: annotation

Return type dict with the following keys

classmethod encode(*annotation*, *key=None*, *value=None*)
Encodes an annotation into a key/value-pair

Parameters

- **annotation** – annotation to encode
- **key** – initial key value to encode annotation into
- **value** – initial value to encode annotation into

Returns

- key: key with annotation encoded
- value: value with annotation encoded

Return type dict with the following keys

class fancy_dict.loader.**LoaderInterface**(*output_type*)
Bases: object

Interface for a FancyDict Loader

classmethod can_load(*source*)
Checks if the loader can load the given source

Parameters **source** – source to load

Returns True if Loader can load the source else False

load(*source*, *annotations_decoder=None*)
Loads a FancyDict from a given source

If an annotations_decoder is given, Annotations can be decoded from the source data.

Parameters

- **source** – source to load from
- **annotations_decoder** – Decoder to decode annotations from source data

Returns FancyDict

2.3 Annotations

Annotations for FancyDict keys

class fancy_dict.annotations.**Annotations**(**_values_)
Bases: object

Annotates a FancyDict key

Composed of a merge method, a merge condition and if the key is finalized

The merge method specifies a method how values for this key gets merged with other values.

A conditions can block merging two values based on the old and new value.

If a key is finalized, the value cannot be updated anymore.

DEFAULTS = {'condition': <function always at 0x7ff1135901e0>, 'finalized': False, 'm

get(*key*)
get the value of an annotation

Returns annotation value or None if not set

update(*new_annotations*)
Updates annotations.

Updates only values which are set in the new annotations and keeps the other values.

Parameters **new_annotations** – Annotations with value to update

2.4 Conditions

FancyDict Conditions

A condition is used by a FancyDict to prevent a value from being changed.

fancy_dict.conditions.**always**(*_old_value*, *_new_value*)
Value can be changed always

Parameters

- **_old_value** – value to change
- **_new_value** – new value

Returns True

fancy_dict.conditions.**if_existing**(*old_value*, *_new_value*)
Value can be changed if value already exists

Parameters

- **old_value** – value to change
- **_new_value** – new value

Returns True if old_value exists

fancy_dict.conditions.**if_not_existing**(*old_value*, *new_value*)
Value can be changed if value exists not yet

Parameters

- **old_value** – value to change

- **new_value** – new value

Returns True if old_value exists not yet

2.5 Merger

Default merging methods for FancyDict

class fancy_dict.merger.**MergeMethod**(*method*, *from_types=None*, *to_types=None*)
Bases: object

Wrapper for a merging method

Can check if the method can merge two values. Provides a method to merge to values.

Method applies when the old value is an instance of from_types and the new value is an instance of to_types.

applies(*old*, *new*)
Checks if the types of old and new match from_types and to_types

Parameters

- **old** – old value to merge
- **new** – new value to merge

Returns True if old of type from_types and new of type to_types.

fancy_dict.merger.**add**(*old*, *new*)
Adds the new value to the old value

Parameters

- **old** – old value to extend
- **new** – new value

Returns extended old value

fancy_dict.merger.**overwrite**(*_old*, *new*)
Overwrites the old value with the new value

Parameters

- **_old** – old value to be overwritten
- **new** – new value to overwrite

Returns new value

fancy_dict.merger.**update**(*old*, *new*)
Updates the old value with the new value

Parameters

- **old** – old value to update
- **new** – new value with updates

Returns updated old value

2.6 Exceptions

FancyDict Exceptions

exception fancy_dict.errors.**FancyDictException**
Bases: Exception

Base Exception for FancyDict errors

exception fancy_dict.errors.**NoLoaderForSourceAvailable**(*source*)
Bases: *fancy_dict.errors.FancyDictException*

Exception when no Loader can load from the given source

exception fancy_dict.errors.**NoMergeMethodApplies**(*old_value*, *new_value*)
Bases: *fancy_dict.errors.FancyDictException*

Exception when no merge method applies

Python Module Index

f

Index

M

N

O

U