FlexDict Documentation

Release 0.0.1.a1

Berkay Öztürk

Nov 02, 2019

Contents

Easily work with deeply nested dictionaries **and** write clean code using FlexDict; a *small* subclass of `dict`. FlexDict provides automatic and arbitrary levels of nesting along with additional utility methods.

CHAPTER 1

Getting Started

1.1 Requirements

1. Any Python version >= 2.7

1.2 Setup

1. Install.

```
pip install flexdict
```

2. Import.

```
from flexdict import FlexDict
```

3. Create.

```
FlexDict()
```

1.3 Usage

Head over to the *User's Guide* to start using *FlexDict*!

CHAPTER 2

User's Guide

The main purpose of `FlexDict` is to allow you to work with deeply nested dictionaries with minimal amount of code. It achieves this purpose by providing an automatic nesting algorithm. It can be a dangerous feature if not used with caution. That's why, `FlexDict` provides some helper methods to prevent any unintentional side-effects.

2.1 Setting Items

When it comes to setting dictionary items, `FlexDict` provides many options. Let's start with the most *slick* way:

```
f = FlexDict()

f['easily', 'create', 'deeply', 'nested', 'structures'] = 1
```

The resulting dictionary would be:

```
{'easily':{'create':{'deeply':{'nested':{'structures': 1}}}}}
```

You can directly pass instances of `list`, `tuple` or `set` instead:

```
f[['easily', 'create', 'deeply', 'nested', 'structures']]
f[('easily', 'create', 'deeply', 'nested', 'structures')]
f[{'easily', 'create', 'deeply', 'nested', 'structures'}]
```

You also have other options:

```
f['easily']['create']['deeply']['nested']['structures'] = 1

f.set(['easily', 'create', 'deeply', 'nested', 'structures'], 1)
f.set(('easily', 'create', 'deeply', 'nested', 'structures'), 1)
f.set({'easily', 'create', 'deeply', 'nested', 'structures'}, 1)
```

The resulting dictionary would be the same for all these examples. However, the `set` method provides many other features. For example, you may only want to set the dictionary items if they do not already exist:

```
f = FlexDict({'a': {'b':1}})

f.set(['a', 'b'], 2, overwrite=False)
f.set(['a', 'c'], 2, overwrite=False)
```

This prevents you from overwriting existing values:

```
{'a': {'b': 1, 'c': 2}}
```

Or, if you need a counter, you can use the `increment` argument to do exactly that:

```
f = FlexDict()

for i in range(20):
    if i % 2 == 0:
        f.set('Even', 1, increment=True)
    else:
        f.set('Odd', 1, increment=True)

f
```

Output:

```
{'Even': 10, 'Odd': 10}
```

(Note that `overwrite` argument has no effect when `increment` is enabled.)

2.2 Getting Items

Again, `FlexDict` provides many alternative ways to access your dictionary items:

```
f = FlexDict({'key1': {'key2': {'key3': 1}}})

# 1
f['key1', 'key2', 'key3']

# 2
f['key1']['key2']['key3']

# 3
f.get(['key1', 'key2', 'key3'])
f.get(('key1', 'key2', 'key3'))
f.get({'key1', 'key2', 'key3'})
```

They will all return the same result:

```
1
```

There is a crucial distinction between these alternatives. Whenever you use squared brackets to access an item, `FlexDict` **will automatically create the keys and fill the value with an empty** `FlexDict` **if there is no such item**:

```
f = FlexDict()

f['a', 'b']
```

(continues on next page)

(continued from previous page)

```
f
```

Output:

```
{'a': {'b': {}}}
```

To prevent this side-effect, `FlexDict` provides two options. First one, is the `get` method:

```
f = FlexDict()

f.get(['a', 'b']), f.get(['a', 'b'], default=0), f
```

The `get` method returns the value provided with the `default` argument if the target item does not exist:

```
(None, 0, {})
```

The other option to avoid the aformentioned side-effect is to use the recursive locking mechnasim via the `lock` method. We will cover it later in this guide. However, just to give you a taste of it, the following example is added:

```
f = FlexDict()

f.lock()

f['a', 'b']
```

Output:

```
KeyError: 'a'
```

Getting the top level keys and values works just like a regular `dict`:

```
f = FlexDict({'a': 1, 'b': 2})

f.keys(), f.values()
```

The only difference you would notice is `f.values()` returns a `list` instead of `dict_values`. This is an intentional behavior since we are working with nested dictionaries:

```
(dict_keys(['a', 'b']), [1, 2])
```

You may also want to get every key and/or value inside your `FlexDict` instance, even the nested ones. `FlexDict` can do this with recursion:

```
f = FlexDict({
    'a': {
        'b': 1,
        'c': {
            'd': 1,
            'e': {
                'a': 3
            }
        }
    },
    'g': 4
})
```

(continues on next page)

(continued from previous page)

```
f.keys(nested=True), f.values(nested=True)
```

This allows you to check exactly what is inside your `FlexDict` instance:

```
(['a', 'b', 'c', 'd', 'e', 'a', 'g'], [1, 1, 3, 4])
```

You can even get rid of the duplicates:

```
f.keys(nested=True, unique=True), f.values(nested=True, unique=True)
```

Note that unique items gets returned inside of a `set`:

```
({'a', 'b', 'c', 'd', 'e', 'g'}, {1, 3, 4})
```

If you wish, you can flatten the entire `FlexDict` instance. The `flatten` method returns a `list` of values and their respective key-paths:

```
f.flatten()
```

Output:

```
[(['a', 'b'], 1), (['a', 'c', 'd'], 1), (['a', 'c', 'e', 'a'], 3), (['g'], 4)]
```

Last but not least, if you wish to get the last item and remove it from the `FlexDict` instance, you can use the `pop` method:

```
f = FlexDict({'a': 1, 'b': 2})

f.pop(), f
```

Output:

```
({'b': 2}, {'a': 1})
```

2.3 Locking & Unlocking Automatic Nesting

Like we discussed above, automatic nesting can be very dangerous in some cases. Thats why, aside from the previously mentioned workarounds, `FlexDict` provides a recursive algorithm to lock and unlock this feature:

```
f = FlexDict()

f.lock()

f['a'] = 1  # Normal `dict` behavior works as expected

try:
    f['b', 'c'] = 1 # Will throw a KeyError
except KeyError:
    f.unlock()
    f['b', 'c'] = 1

f
```

Output:

```
{'a': 1, 'b': {'c': 1}}
```

Each `FlexDict` instance has an attribute called `locked` which tells if it is locked. **Each nested dictionary inside a `FlexDict` instance is also a seperate `FlexDict` instance!** This means, each of them has seperate `locked` attributes. The `lock` method sets the `locked` attribute of the specified `FlexDict` instance and of all the other nested dictionaries inside of it to `True`. `unlock` method on the other hand, does the exact opposite. This means that you can create any hybrid lock structure you want (Do that with caution!):

```
f = FlexDict({'secure': {}, 'not_secure': {}})

f['secure'].lock()

f.locked, f['secure'].locked, f['not_secure'].locked
```

Output:

```
(False, True, False)
```

Both `lock` and `unlock` methods provide an argument called `inplace` which allows you to create locked/unlocked copies of your `FlexDict` instances:

```
f = FlexDict()

f_locked = f.lock(inplace=False)

f.locked, f_locked.locked
```

Output:

```
(False, True)
```

2.4 Other Utility Methods

You can check if your `FlexDict` instance contains (is a superset of) or inside of (is a subset of) another `dict` instance.

```
f = FlexDict({'a': {'b': 1}})

f.contains({'b': 1}), f.inside({'c': {'a': {'b': 1}}})
```

Output:

```
(True, True)
```

`FlexDict` also allows you to easily get the length (number of keys) and size (number of keys and values) inside your dictionaries via `length` and `size` methods. They both utilize the previously mentioned `keys` and `values` methods. Hence, they can work recursively and get rid of duplicates if you wish:

```
f = FlexDict({
    'a': {
        'b': 1,
        'c': {
```

(continues on next page)

(continued from previous page)

```
            'd': 1,
            'e': {
                'a': 3
            }
        }
    },
    'g': 4
})

# Can be used as a replacement for len()
print(f'Number of keys:', f.length())
print(f'Number of keys (Recursive):', f.length(nested=True))
print(f'Number of keys (Recursive, Unique):', f.length(nested=True, unique=True))

# Saves some of your time
print(f'\nNumber of items (Recursive):', f.size())
print(f'Number of items (Recursive, Unique):', f.size(unique=True))
```

Output:

```
Number of keys: 2
Number of keys (Recursive): 7
Number of keys (Recursive, Unique): 6

Number of items (Recursive): 11
Number of items (Recursive, Unique): 9
```

CHAPTER 3

Contributing

1. Get the source.

```
git clone https://github.com/ozturkberkay/FlexDict.git
```

2. Install tox.

```
pip install tox
```

3. Test your changes.

```
tox
```

4. If you pass every test, make a PR request using your own branch.

```
git checkout -b mychange
git push origin mychange
```

CHAPTER 4

API Reference

class flexdict.**FlexDict**(*data=None*)

Bases: dict

Provides automatic and arbitrary levels of nesting along with additional utility methods.

Parameters **data** (*dict*) – Data to initialize the FlexDict with.

locked

Flag indicating if auto-nesting is locked.

Type bool

contains(*subset*)

Checks if this dictionary is a superset of a given one.

Parameters **subset** (*dict*) – Dictionary to check if it is a subset.

Returns *True* if *self* contains *subset* else *False*.

Return type bool

flatten()

Flattens the dictionary.

Returns A list of tuples containing key-paths and values.

Return type list

get(*keys*, *default=None*)

Gets a value from the dictionary with the provided keys.

Parameters

- **keys** – Keys pointing to the target value.
- **default** (*any*) – Default value to return if target does not exists.

Returns The corresponding dictionary value.

Return type any

inside(*superset*)
Checks if this dictionary is a subset of a given one.

Parameters **superset** (*dict*) – Dictionary to check if it is a superset.

Returns *True* if *self* is inside the *superset* else *False*.

Return type bool

keys(*nested=False*, *unique=False*)
Gets keys from the dictionary.

Parameters

- **nested** (*bool*) – Gets all keys recursively if set to *True*.
- **unique** (*bool*) – Gets only the unique keys if set to *True*.

Returns

dict_keys If *nested* is *False* and *unique* is *False*.

list If *nested* is *True* and *unique* is *False*.

set If *unique* is *True*.

Return type Union[dict_keys, list, set]

length(*nested=False*, *unique=False*)
Counts the number of keys inside the dictionary.

Parameters

- **nested** (*bool*) – Counts all keys recursively if set to *True*.
- **unique** (*bool*) – Counts only the unique keys if set to *True*.

Returns Number of keys.

Return type int

lock(*inplace=True*)
Locks the automatic nesting mechanism.

Parameters **inplace** (*bool*) – Creates a locked copy if *True*.

Returns

None If *inplace* is set to *True*.

FlexDict If *inplace'set to 'False*.

Return type Union[None, *FlexDict*]

pop()
Removes and returns the last key-value pair from the dictionary.

Returns

FlexDict The last key-value pair of the dictionary.

None If *self* is empty.

Return type Union[*FlexDict*, None]

set(*keys*, *value*, *overwrite=True*, *increment=False*)
Sets a dictionary value with the given keys.

Parameters

- **keys** (*any*) – Key(s) pointing to the value.
- **value** (*any*) – Value to set.
- **overwrite** (*bool*) – If *False*, only sets a value if it not exists.
- **increment** (*bool*) – Increments the value by *value* if set to *True*. *overwrite* argument has no effect on this. Causes the method to return the target value.

Returns Final state of the target value if *increment* is enabled.

Return type Union[int, float, None]

size(*unique=False*)

Counts the number of keys and values inside the dictionary.

Parameters

- **nested** (*bool*) – Counts all items recursively if set to *True*.
- **unique** (*bool*) – Counts only the unique items if set to *True*.

Returns Number of items.

Return type int

unlock(*inplace=True*)

Unlocks the automatic nesting mechanism.

Parameters **inplace** (*bool*) – Creates an unlocked copy if *True*.

Returns

None If *inplace* is set to *True*.

FlexDict If *inplace'set to 'False*.

Return type Union[None, *FlexDict*]

values(*nested=False*, *unique=False*)

Gets values from the dictionary.

Parameters

- **nested** (*bool*) – Gets all values recursively if set to *True*.
- **unique** (*bool*) – Gets only the unique values if set to *True*.

Returns

dict_values If *nested* is *False* and *unique* is *False*.

list: If *nested* is *True* and *unique* is *False*.

list: If *unique* is *True*.

Return type Union[dict_values list, set]

Index