# dkey

***Release heads/master***

**Mar 30, 2019**

# Introduction


**1 Installation** **3**

**2 Usage example** **5**

2.1 Behaviour of wrapped dictionary . . . 5

2.1.1 Single item access . . . 5

2.1.2 Multi item access . . . 5

2.2 Configuration Options . . . 6

2.2.1 Removing a key . . . 6

2.2.2 Replacing a key . . . 6

2.2.3 More configuration options . . . 7

2.2.3.1 Version numbers . . . 7

2.2.3.2 Custom message . . . 7

2.2.3.3 Custom warning type . . . 8

2.2.4 Limitations . . . 8

2.3 deprecate_keys . . . 9

2.4 dkey . . . 12

The `dkey` module allows you to deprecate the use of selected keys in a given dictionary thus making API changes involving dictionaries less disruptive. To learn more about how to use it head over to the Documentation.

CHAPTER 1

# Installation

To install this package simply use pip:

```
pip install dkey
```

CHAPTER 2

# Usage example

Let's say we have a dict with an old key that we want to replace with a new one.

## 2.1 Behaviour of wrapped dictionary

There are different ways items can be accessed in dicts. `dkey` tries to find the best warning solution for each case.

### 2.1.1 Single item access

When accessing a single item in a dict, `dkey` warns only, when the associated key is deprecated. So if key *A* is deprecated, all of these operation warn:

```
my_dict['A']
my_dict['A'] = 12
my_dict.get('A')
del my_dict['A']
my_dict.pop('A')
```

Setting or removing a deprecated (key,value) pair will also remove the warning:

```
my_dict['A'] = 12  # warns
my_dict['A']       # does not warn
```

This is to make sure that warnings do not propagate completely out of context.

### 2.1.2 Multi item access

Due to the special view classes dict uses to give access to its internal structure via the functions `dict.values`, `dict.keys`, and `dict.items`, these functions return *all* warnings directly when called. For all other functions like normal iteration:

```
for key in my_dict: # warns when key is 'A'
    print(key)
```

or `dict.popitem`, the warning is instead generated, when the deprecated element is accessed.

## 2.2 Configuration Options

To deprecate a key in a dict, it has to be wrapped using *dkey.deprecate_keys*. `dkey` offers two different ways to deprecate a key. Either, the key is deprecated as it will be replace by a different one, or it will be removed completely in the future. In both scenarios it is important, that during the deprecation period, the old keys must still work the same way as before.

### 2.2.1 Removing a key

To deprecate a key and warn that it will be removed in the future, you wrap your dict using *dkey.deprecate_keys* and pass it the deprecated key using *dkey.dkey*:

```
from dkey import deprecate_keys, dkey

def customer_info():
    return deprecate_keys({
            'name': 'Smith',
            'age': 24,
            'cleartext password': 'password'
        },
        dkey('cleartext password'))
```

Should someone now try to access the `cleartext password` key, a `DeprecationWarning` is generated:

```
print(customer_info()['cleartext password'])
# Wil warn with a DeprecationWarning:
# Key `cleartext password` is deprecated. It shouldn't be used anymore.
```

An automatically generated deprecation warning is used that warns developers to no longer use this key in the future.

---

**Note:** A `DeprecationWarning` is only shown when you set your warning filters appropriately. For other warning types that are shown by default, check *this section*.

---

### 2.2.2 Replacing a key

Another scenario you might run into is that you want to rename a key (for various reasons), and again you want to give people time to adapt. For this you can do the following:

```
from dkey import deprecate_keys, dkey

def customer_info():
    return deprecate_keys({
            'first name': 'Adam',
            'last name': 'Smith',
            'age': 24,
```

(continues on next page)

(continued from previous page)

```
        },
        dkey('name', 'last name'))
```

Again we use the *deprecate_keys* function. This time we pass two string to *dkey.dkey*: The old key and the new key people should be using. Both keys will point to the same object. The result is again a warning if people use the old key:

```
print(customer['name'])
# Wil raise a DeprecationWarning:
# Key `name` is deprecated. Use `first name` from now on.
```

And again an automatically generated deprecation warning is used that also informs developers about which key to use instead.

### 2.2.3 More configuration options

In case the default way warnings are generated are not what you need, *dkey* offers several ways to customise how deprecated keys are treated:

- A version number can be given to indicate since when a key is deprecated
- A version number can be given to indicate when a key is definitively removed
- A custom message can be given to add more information about why the change happend and how to adapt
- A custom warning type can be given to align the deprecation warnings to an existing project

#### 2.2.3.1 Version numbers

If you have a well organised code project, you will normally also want to communicate since when a feature is deprecated and when it finally will be removed. You can pass those details to *dkey.dkey*:

```
from dkey import deprecate_keys, dkey

def customer_info():
    return deprecate_keys({
            'first name': 'Adam',
            'last name': 'Smith',
            'age': 24,
        },
        dkey('name', 'last name', deprecated_in='1.1.12', removed_in='2.0.0'))
```

Which will result in the warning:

> Key *name* is deprecated since version 1.1.12. It will be removed in version 2.0.0. Use *first name* from now on.

#### 2.2.3.2 Custom message

Sometimes, changes are not as simple as *a* was replaced with *b*. In these scenarios, you can provide a custom message with more information:

```
def customer_info():
    return deprecate_keys({
            'first name': 'Adam',
            'last name': 'Smith',
            'age': 24,
        },
        dkey('name', 'last name',
            details='`name` has been replaced by the two keys `first name` and `last␣
↪name`.'))
```

Which will result in the warning:

> Key *name* is deprecated. *name* has been replaced by the two keys *first name* and *last name*.

#### 2.2.3.3 Custom warning type

By default, a `DeprecationWarning` is used. This warning does not appear for end users. If you have deprecation warnings that are actually meant for end users and not just for developers, you can change the warning type:

```
from dkey import deprecate_keys, dkey

def customer_info():
    return deprecate_keys({
            'name': 'Smith',
            'age': 24,
            'cleartext password': 'password'
        },
        dkey('cleartext password', warning_type='end user'))
```

Which will generate a `FutureWarning`. `FutureWarning` is a warning type that is shown to end users by default. If you want to show your own warning type, this is also possible. Just hand your warning type to `warning_type` instead of the string *'end user'* and it will be used to spawn the warning:

```
from dkey import deprecate_keys, dkey

class UltimateWarning(FutureWarning):
    pass

def customer_info():
    return deprecate_keys({
            'name': 'Smith',
            'age': 24,
            'cleartext password': 'password'
        },
        dkey('cleartext password', warning_type=UltimateWarning))
```

**Note:** In order for your custom warning type to work it has to be compatible with the `warnings.warn` function.

### 2.2.4 Limitations

- No automatic doc-string adaptations are possible as of now

## 2.3 deprecate_keys

**class** dkey.**deprecate_keys**(*dictionary*, **args*)

Wrapper for dicts that allows to set certain keys as deprecated.

**__init__**(*dictionary*, **args*)

Construct the wrapper class.

Internally, the items are stored in the same order as the given dictionary (CPython >=3.6). The deprecated old keys that were replaced with new ones are positioned just before their respective new keys.

**Parameters**

- **dictionary** (*dict*) – The dictionary to wrap
- ***args** – Zero or more keys that should show deprecation warnings. Use *dkey.dkey* for each key.

**__contains__**(*key*)

Return *True* if the given key *key* is in this dict, else *False*.

Warns if the given key is deprecated.

**Parameters** **key** – The key to search in this dict.

**Returns** **IsInDict** – *True* if the given key is in this dict. *False*, otherwise.

**Return type** bool

**Warns** **CustomWarning** – Warns with the warning stored for the given key if the key is deprecated.

**__delitem__**(*key*)

Remove the item with key *key* and its associated value.

Warns if the given key is deprecated.

**Parameters** **key** – The key of the item which to remove.

**Raises** KeyError – Raises a KeyError if the given key does not exist

**Warns** **CustomWarning** – Warns with the warning stored for the given key if the key is deprecated. Further access to the given key will not spawn additional warnings.

**__eq__**(*other*)

Return *True* if identical to *other*.

Warns if either this or *other* contains deprecated keys.

**Parameters** **other** – The other object to compare this one to

**Returns** True if they are equal, False otherwise

**Return type** bool

**Warns** **CustomWarning** – Warns with the warnings stored for all contained keys.

**__getitem__**(*key*)

Get the value of the item of the given key *key*.

Warns if the given key is deprecated. If the key does not exist, it will behave the same as a normal dict, i.e. it will raise a KeyError.

**Parameters** **key** – The key for which to return the value

**Returns** The value stored for the given key

**Return type** value

**Raises** `KeyError` – If the key is not found

**Warns** **CustomWarning** – Warns with the warning stored for the given key if the key is deprecated.

**__iter__**()
Return an iterator over the keys of the dictionary.

Warns for each deprecated item accessed.

**Returns** **Iterator** – An iterator over the wrapped dict

**Return type** iterator

**Warns** **CustomWarning** – Warns whenever a deprecated item in the dict is returned. Each item will warn with its set warning type and message.

**__len__**()
Return the number of items in the dict.

Will raise warnings, if the dict contains deprecated values. One for each deprecated value.

**Returns** The number of items in the dict

**Return type** int

**Warns** **CustomWarning** – Warns for each deprecated item in the dictionary before returning.

**__ne__**(*other*)
Return *True* if not identical to *other*.

Warns if either this or *other* contains deprecated keys.

**Parameters** **other** – The other object to compare this one to

**Returns** True if they are not equal, False otherwise

**Return type** bool

**Warns** **CustomWarning** – Warns with the warnings stored for all contained keys.

**__setitem__**(*key*, *value*)
Set the value of the item of the given key *key* to *value*.

Warns if the given key is deprecated.

**Parameters**

- **key** – The key under which to store the given value
- **value** – The value to store

**Warns** **CustomWarning** – Warns with the warning stored for the given key if the key is deprecated. Further access to the given key will not spawn additional warnings.

**clear**()
Remove all entries from the dict.

Will also remove all deprecation warnings and all keys.

**copy**()
Return a shallow copy of this wrapped dict.

Will return a wrapped dict that, as the original, will warn with the deprecation warnings set for this dict.

**Returns** **copy** – A shallow copy of the underlying dict and a shallow copy of the underlying deprecation key structure.

**Return type** *deprecate_keys*

**get**(*key*, *default=None*)
Get the value stored under *key* or *default* if this key doesn't exist.

This function is the same as `deprecate_keys.__getitem__` except that it does not throw for non existing keys, but returns the given default value instead.

**Parameters**

- **key** – The key for which to return the value
- **optional** (*default,*) – The value to return, if the given key is not stored in the dict. Defaults to `None` if not given.

**Returns** The value stored for *key* or *default* if *key* is not in the dict.

**Return type** value

**items**()
Return a new view of the dictionary's items: iterator of *(key, value)* pairs.

Works the same as the plain `dict.items` function except that it warns about all deprecated keys contained before returning.

**Returns** Basically an iterator over *(key, value)* pairs of the dict. For more information about dict views see: dictionary view

**Return type** dict_items

**Warns** **CustomWarning** – Warns for each deprecated item in the dictionary before returning.

**keys**()
Return a new view of the dictionary's keys.

Works the same as the plain `dict.keys` function except that it warns about all deprecated keys before returning the view.

**Returns**

Basically an iterator over the contained keys of the dict. For more information about dict views see: dictionary view

**Return type** dict_keys

**Warns** **CustomWarning** – Warns for each deprecated item in the dictionary before returning.

**pop**(*key*, *default=<object object>*)
Remove and return the item with key *key*.

If the key is not in the dict and no default value is set, this function will raise an exception. If a default value is given, this value will be returned instead.

**Parameters**

- **key** – The key to pop
- **default** (*optional*) – The value to return if the given *key* is not in the dict. If non is given, an exception is raised instead.

**Returns** The value of the key given or the given default value, or if no default value is given, an exception is raised.

**Return type** value

**Raises** `KeyError` – If *key* is not in the dict and no default value is given, this function raises a `KeyError`.

**Warns** **CustomWarning** – Warns if the popped item is a deprecated key. The deprecation information for this key is removed.

**popitem**()
Pop an item from the dict.

This function has the same behaviour as `dict.popitem`. Therefore, for Python < 3.6 this function will pop an arbitrary item, whereas for Python >= 3.6 this function will raise the last item added to this dict. If the dict is empty, this function will raise a `KeyError`.

**Returns**

- *key* – The key popped
- *value* – The associated value of the popped key

**Raises** `KeyError` – If the dict is empty.

**Warns** **CustomWarning** – Warns if the popped item is a deprecated key. The deprecation information for this key is removed.

**values**()
Return a new view of the dictionary's values.

Works the same as the plain `dict.values` function except that it warns about all deprecated keys associated with returned values before returning the view.

**Returns**

Basically an iterator over the contained values of the dict. For more information about dict views see: dictionary view

**Return type** dict_values

**Warns** **CustomWarning** – Warns for each deprecated item in the dictionary before returning.

## 2.4 dkey

dkey.**dkey**(**args*, *deprecated_in=None*, *removed_in=None*, *details=None*, *warning_type='developer'*)
Convert a key into a deprecation lookup dict.

To use the *`dkey.deprecate_keys`* function it is easiest to generate its input with this function. This function generates:

- A key removed deprecation warning object if one key is provided
- A key replaced deprecation warning object if two keys are provided

**Parameters**

- ***args** – One or two keys. If one key is passed, it is assumed that this key will be removed in the future. If two keys are passed, it is assumed that the second key is the replacement for the first one.
- **deprecated_in** (*str, optional*) – Version in which this key was deprecated. If given, will appear in the warning message.
- **removed_in** (*str, optional*) – Version in which this key will be removed and will no longer work. If given, will appear in the warning message.

- **details** (*str, optional*) – Will remove the default final sentence (do no longer use, or use *xxx* from now on).
- **warning_type** (*{'developer', 'end user', ArbitraryWarning}, optional*) – The warning type to use when the old key is accessed

  By default, deprecation warnings are intended for developers only which means a any:*DeprecationWarning* is used which isn't shown to end users. If it should be shown to end users, this can be done by passing 'end user' which will raise `FutureWarning`. If you want to use your custom warning type this is also possible.

  **Note:** Your custom warning must work with `warnings.warn`

**Returns** A dict that can be used as a deprecated key input for *dkey.deprecate_keys*.

**Return type** dict

**Raises** `ValueError` – If zero or more than two keys are passed to this function.

# Index

## Symbols


__contains__() (*dkey.deprecate_keys method*), 9
__delitem__() (*dkey.deprecate_keys method*), 9
__eq__() (*dkey.deprecate_keys method*), 9
__getitem__() (*dkey.deprecate_keys method*), 9
__init__() (*dkey.deprecate_keys method*), 9
__iter__() (*dkey.deprecate_keys method*), 10
__len__() (*dkey.deprecate_keys method*), 10
__ne__() (*dkey.deprecate_keys method*), 10
__setitem__() (*dkey.deprecate_keys method*), 10


## C


clear() (*dkey.deprecate_keys method*), 10
copy() (*dkey.deprecate_keys method*), 10


## D


deprecate_keys (*class in dkey*), 9
dkey() (*in module dkey*), 12


## G


get() (*dkey.deprecate_keys method*), 11


## I


items() (*dkey.deprecate_keys method*), 11


## K


keys() (*dkey.deprecate_keys method*), 11


## P


pop() (*dkey.deprecate_keys method*), 11
popitem() (*dkey.deprecate_keys method*), 12


## V


values() (*dkey.deprecate_keys method*), 12