dkey

Release heads/master

Jan 18, 2019

Contents:

This module provides a thin wrapper that can be used to set certain keys in dictionaries as deprecated. This allows e.g. for an easy way to gently push out interface changes instead of just introducing breaking changes without any prior warnings.

CHAPTER 1

Installation

To install this package simply use pip.

```
pip install dkey
```

CHAPTER 2

Usage example

2.1 Removing a key

Let's say we have the following function that returns a dict:

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

And the following code that uses our function:

```
def my_func():
    customer = customer_info()
    print(customer['cleartext password'])
```

Now we want to remove the *'cleartext password'* from our returned dict due to security concerns. However, we want to give others time to adapt to our changes, so instead of just removing it, we deprecate the usage of that dict entry:

```
from dkey import deprecate_keys, dkey

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

We use the function `deprecate_keys` to deprecate the key *'cleartext password'*. To pass the key to `deprecate_keys` we use the convenience function `dkey`. Now if we call *my_func* again:

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

As you can see an automatically generated deprecation warning is used.

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

Again we use the *deprecate_keys* function. This time we pass two keys to *dkey.dkey*. The old key and the new key people should be using. The result is:

```
def my_func():
    customer = customer_info()
    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 the developers about which key to use instead.

2.3 More configuration options

If you have a well organised code project, you will normally also want to communicate since when a feature is deprecated and when it will get removed completely. Maybe you also want to give more detailed information about the changes than what the default message offers. 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',
            details='`name` has been replaced by the two fields `first name` and␣
↪`last name`.'))
```

Which will result in the warning:

> Key *name* is deprecated since version 1.1.12. It will be removed in version 2.0.0. *name* has been replaced by the two fields *first name* and *last name*.

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 results in:

```
def my_func():
    customer = customer_info()
    print(customer['cleartext password'])
    # Wil raise a FutureWarning: Key `cleartext password` is deprecated. It shouldn't␣
↪be used anymore.
```

`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.

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

CHAPTER 3

Limitations

- Currently, only key access can be checked and deprecation warnings are shown. There are no warnings for changes in the number of entries in the dict.
- Furthermore, no automatic doc-string adaptations are possible as of now

CHAPTER 4

Module documentation

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

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

dictionary

dict – The dictionary to wrap

***args**

Zero or more keys that should show deprecation warnings. Use *dkey.dkey* for each key.

Warns **ArbitraryWarning** – If a deprecated key is used, shows a warning that was set for this key. (see also *dkey.dkey*)

dkey.**dkey**(**args*, *deprecated_in=None*, *removed_in=None*, *details=None*, *warning_type='developer'*)

Function that converts 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.

Python Module Index

d

Index

D