mingrammer Follow
I love computer science, automation and mathematics. github.com/mingrammer
Jan 15, 2017 · 4 min read

Understanding the underscore(_) of Python

> *I'm not a native speaker. Sorry for my english. Please understand.*

The *underscore* (_) is special in Python.

While the *underscore* (_) is used for just snake-case variables and functions in most languages (Of course, not for all), but it has special meanings in Python. If you are python programmer, `for _ in range(10)` , `__init__(self)` like syntax may be familiar.

This post will explain the about when and how use the *underscore* (_) and help you understand it.

There are 5 cases for using the *underscore* in Python.

1. For storing the value of last expression in interpreter.
2. For ignoring the specific values. (so-called "I don't care")
3. To give special meanings and functions to name of vartiables or functions.
4. To use as 'Internationalization(i18n)' or 'Localization(l10n)' functions.
5. To separate the digits of number literal value.

Let's look at each case.

When used in interpreter

The python interpreter stores the last expression value to the special variable called '_'. This feature has been used in standard CPython interpreter first and you could use it in other Python interpreters too.

```
>>> 10
10
>>> _
10
>>> _ * 3
30
>>> _ * 20
600
```

For Ignoring the values

The *underscore* is also used for ignoring the specific values. If you don't need the specific values or the values are not used, just assign the values to *underscore*.

```
# Ignore a value when unpacking
x, _, y = (1, 2, 3) # x = 1, y = 3


# Ignore the multiple values. It is called "Extended
Unpacking" which is available in only Python 3.x
x, *_, y = (1, 2, 3, 4, 5) # x = 1, y = 5


# Ignore the index
for _ in range(10):
    do_something()


# Ignore a value of specific location
for _, val in list_of_tuple:
    do_something()
```

Give special meanings to name of variables and functions

The *underscore* may be most used in 'naming'. The PEP8 which is Python convention guideline introduces the following 4 naming cases.

_single_leading_underscore
This convention is used for declaring ***private*** variables, functions, methods and classes in a module. Anything with this convention are ignored in `from module import *`.
However, of course, Python does not supports ***truly private,*** so we can

not force somethings private ones and also can call it directly from other modules. So sometimes we say it “weak internal use indicator”.

```
_internal_name = 'one_nodule' # private variable
_internal_version = '1.0' # private variable


class _Base: # private class
    _hidden_factor = 2 # private variable

    def __init__(self, price):
        self._price = price

    def _double_price(self): # private method
        return self._price * self._hidden_factor

    def get_double_price(self):
        return self._double_price()
```

single_trailing_underscore_
This convention could be used for avoiding conflict with Python keywords or built-ins. You might not use it often.

```
Tkinter.Toplevel(master, class_='ClassName') # Avoid
conflict with 'class' keyword


list_ = List.objects.get(1) # Avoid conflict with 'list'
built-in type
```

__double_leading_underscore
This is about syntax rather than a convention. *double underscore* will mangle the attribute names of a class to avoid conflicts of attribute names between classes. (so-called “mangling” that means that the compiler or interpreter modify the variables or function names with some rules, not use as it is)
The mangling rule of Python is adding the “_ClassName” to front of attribute names are declared with *double underscore*.
That is, if you write method named “__method” in a class, the name will be mangled in “_ClassName__method” form.

```
class A:
    def _single_method(self):
        pass


    def __double_method(self): # for mangling
        pass


class B(A):
    def __double_method(self): # for mangling
        pass
```

Because of the attributes named with *double underscore* will be mangled like above, we can not access it with “ClassName.__method”. Sometimes, some people use it as like real private ones using these features, but it is not for private and not recommended for that. For more details, read Python Naming.

__double_leading_and_trailing_underscore__
This convention is used for special variables or methods (so-called “magic method”) such as `__init__` , `__len__` . These methods provides special syntactic features or does special things. For example, `__file__` indicates the location of Python file, `__eq__` is executed when `a == b` expression is excuted.
A user of course can make custom special method, it is very rare case, but often might modify the some built-in special methods. (e.g. You should initialize the class with `__init__` that will be executed at first when a instance of class is created.)

```
class A:
    def __init__(self, a): # use special method '__init__'
for initializing
        self.a = a


    def __custom__(self): # custom special method. you might
almost do not use it
        pass
```

As Internationalization(i18n)/Localization(l10n) functions

It is just convention, does not have any syntactic functions. That is, the underscore does not means **i18n/l10n**, and it is just a convention that binds the **i18n/l10n** to *underscore* variable has been from C convention.
The built-in library `gettext` which is for **i18n/l10n** uses this convention, and Django which is Python web framework supports **i18n/l10n** also introduces and uses this convention.

```
# see official docs :
https://docs.python.org/3/library/gettext.html
import gettext


gettext.bindtextdomain('myapplication','/path/to/my/language
/directory')
gettext.textdomain('myapplication')
_ = gettext.gettext


# ...


print(_('This is a translatable string.'))
```

To separate the digits of number literal value

This feature was added in Python 3.6. It is used for separating digits of numbers using *underscore* for readability.

```
dec_base = 1_000_000
bin_base = 0b_1111_0000
hex_base = 0x_1234_abcd


print(dec_base) # 1000000
print(bin_base) # 240
print(hex_base) # 305441741
```

Conclusion

So far we've covered the *underscore* of Python. While I'm a Python programmer, I didn't know some of them till wrote this post. Especially, the **i18n/l10n** is very new to me.
Like me, I hope you gain some useful knowledges about *underscore* from this post.

Next, I'll cover more interesting things about Python. Thank you.

Update
Added the new feature (PEP 515) was added in Python 3.6