Log in Create Free Account

Hafeezul Kareem Shaik
October 26th, 2018

Experiment with the code in this tutorial

Open in Workspace

PYTHON +1

Role of Underscore(_) in Python

In this tutorial, you're going to learn about the uses of underscore(_) in python.

Many of the **Python Developers** don't know about the functionalities of **underscore(_)** in **Python**. It helps users to write **Python** code productively.

Underscore(_) is a unique character in **Python**.

If you're a **Python** programmer, you probably familiar with the following syntax:

- **for _ in range(100)**
- **__init__(self)**
- _ = 2

It has some special meaning in different conditions. Let's see all of those.

You will find max six different uses of **underscore(_)**. If you want you can use it for different purposes after you have an idea about **underscore(_)**.

1. Use In Interpreter

1. Ignoring Values

1. Use In Looping

Let's all the uses briefly with examples.

1. Use In Interpreter

Python automatically stores the value of the last expression in the **interpreter** to a particular variable called "_." You can also assign these value to another variable if you want.

You can use it as a normal variable. See the example

```
>>> 5 + 4
9
>>> _     # stores the result of the above expression
9
>>> _ + 6
15
>>> _
15
>>> a = _  # assigning the value of _ to another variable
>>> a
15
```

2. Ignoring Values

Underscore(_) is also used to ignore the values. If you don't want to use specific values while unpacking, just assign that value to **underscore(_)**.

See the example

```
## ignoring a value
a, _, b = (1, 2, 3) # a = 1, b = 3
print(a, b)

## ignoring multiple values
## *(variable) used to assign multiple value to a variable as list while unpacking
## it's called "Extended Unpacking", only available in Python 3.x
a, *_, b = (7, 6, 5, 4, 3, 2, 1)
print(a, b)
```

```
1 3
7 1
```

3. Use In Looping

You can use **underscore(_)** as a variable in looping. See the examples below to get an idea.

```
## lopping ten times using _
for _ in range(5):
    print(_)

## iterating over a list using _
## you can use _ same as a variable
languages = ["Python", "JS", "PHP", "Java"]
for _ in languages:
    print(_)

_ = 5
while _ < 10:
    print(_, end = ' ') # default value of 'end' id '\n' in python. we're changing it to sp
    _ += 1
```

```
1
2
3
4
Python
JS
PHP
Java
5 6 7 8 9
```

4. Separating Digits Of Numbers

If you have a long digits number, you can separate the group of digits as you like for better understanding.

Ex:- **million = 1_000_000**

Next, you can also use **underscore(_)** to separate the binary, octal or hex parts of numbers.

Ex:- **binary = 0b_0010**, **octa = 0o_64**, **hexa = 0x_23_ab**

Execute all the above examples to see the results.

```
## different number systems
## you can also check whether they are correct or not by coverting them into integer using
million = 1_000_000
binary = 0b_0010
octa = 0o_64
hexa = 0x_23_ab

print(million)
print(binary)
print(octa)
print(hexa)
```

```
2
52
9131
```

5. Naming Using Underscore(_)

Underscore(_) can be used to name variables, functions and classes, etc..,

- Single Pre Underscore:- **_variable**
- Signle Post Underscore:- **variable_**
- Double Pre Underscores:- **__variable**
- Double Pre and Post Underscores:- **__variable__**

5.1. _single_pre_underscore

_name

Single Pre Underscore is used for internal use. Most of us don't use it because of that reason.

See the following example.

```
class Test:

    def __init__(self):
        self.name = "datacamp"
        self._num = 7

obj = Test()
print(obj.name)
print(obj._num)
```

```
datacamp
7
```

But, **single pre underscore** effects the names that are imported from the module.

Let's write the following code in the **my_funtions** file.

```
## filename:- my_functions.py

def func():
    return "datacamp"

def _private_func():
    return 7
```

Now, if you **import** all the methods and names from **my_functions.py**, **Python** doesn't import the names which starts with a **single pre underscore**.

```
>>> from my_functions import *
>>> func()
'datacamp'
>>> _private_func()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name '_private_func' is not defined
```

You avoid the above error by importing the module normally.

```
>>> import my_functions
>>> my_functions.func()
'datacamp'
>>> my_functions._private_func()
7
```

Single Pre Underscore is only meant to use for the internal use.

5.2 single_post*underscore*

name_

You can avoid conflicts with the **Python Keywords** by adding an *underscore* at the end of the name which you want to use.

Let's see the example.

```
>>> def function(class):
  File "<stdin>", line 1
    def function(class):
                 ^
SyntaxError: invalid syntax
>>> def function(class_):
...     pass
...
>>>
```

Single Post Underscore is used for naming your variables as **Python Keywords** and to avoid the clashes by adding an underscore at last of your variable name.

5.3. Double Pre Underscore

__name

Double Pre Underscores are used for the **name mangling**.

Double Pre Underscores tells the **Python** interpreter to rewrite the attribute name of subclasses to avoid naming conflicts.

- *Name Mangling*:- interpreter of the **Python** alters the variable name in a way that it is challenging to clash when the class is inherited.

Let's see an example.

```
class Sample():

    def __init__(self):
        self.a = 1
```

```
obj1 = Sample()
dir(obj1)
```

```
['_Sample__c',
 '__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__le__',
 '__lt__',
 '__module__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__weakref__',
 '_b',
 'a']
```

The above code returns all the attributes of the **class** object. Let's see our variables in the attributes list.

self.a variable appears in the list without any change.

- Is there **self.__c** variable in the list?
    - If you carefully look at the attributes list, you will find an attribute called **_Sample__c**. This is the **name mangling**. It is to avoid the overriding of the variable in subclasses.

Let's create another **class** by **inheriting Sample** class to see how overriding works.

```
class SecondClass(Sample):

    def __init__(self):
        super().__init__()
        self.a = "overridden"
        self._b = "overridden"
        self.__c = "overridden"
obj2 = SecondClass()
print(obj2.a)
print(obj2._b)
print(obj2.__c)


overridden
overridden



---------------------------------------------------------------------------

AttributeError                            Traceback (most recent call last)

<ipython-input-2-4bf6884fbd34> in <module>()
      9 print(obj2.a)
     10 print(obj2._b)
---> 11 print(obj2.__c)


AttributeError: 'SecondClass' object has no attribute '__c'
```

```
print(obj2._SecondClass__c)
```

```
overridden
```

See, it's worked you can also access the previously created variable using **_Sample__c**. Let's see

```
print(obj1._Sample__c)
```

```
3
```

You can access the **Double Pre Underscore** variables using methods in the class. Let's see an example.

```
class SimpleClass:

    def __init__(self):
        self.__datacamp = "Excellent"

    def get_datacamp(self):
        return self.__datacamp

obj = SimpleClass()
print(obj.get_datacamp()) ## it prints the "Excellent" which is a __var
print(obj.__datacamp)     ## here, we get an error as mentioned before. It changes the name
```

```
Excellent



---------------------------------------------------------------------------

AttributeError                            Traceback (most recent call last)
```

```
      9 obj = SimpleClass()
     10 print(obj.get_datacamp()) ## it prints the "Excellent" which is a __var
---> 11 print(obj.__datacamp)     ## here, we get an error as mentioned before. It changes


AttributeError: 'SimpleClass' object has no attribute '__datacamp'
```

You can also use the **Double Pre Underscore** for the *method* names. Let's see an example.

```
class SimpleClass:

    def __datacamp(self):
        return "datacamp"

    def call_datacamp(self):
        return self.__datacamp()

obj = SimpleClass()
print(obj.call_datacamp()) ## same as above it returns the Dobule pre underscore method
print(obj.__datacamp())    ## we get an error here
```

```
datacamp




---------------------------------------------------------------------------

AttributeError                            Traceback (most recent call last)

<ipython-input-1-cd8ce2e83589> in <module>()
      9 obj = SimpleClass()
     10 print(obj.call_datacamp()) ## same as above it returns the Dobule pre underscore me
---> 11 print(obj.__datacamp())    ## we get an error here


AttributeError: 'SimpleClass' object has no attribute '__datacamp'
```

Underscore name.

Let's see an example.

```
_SimpleClass__name = "datacamp"

class SimpleClass:

    def return_name(self):
        return __name

obj = SimpleClass()
print(obj.return_name()) ## it prints the __name variable
```

```
datacamp
```

Did you understand the concept? If you didn't, try to reread it.

5.4. Double Pre And Post Underscores

__name__

In **Python**, you will find different names which start and end with the **double underscore**. They are called as **magic methods** or **dunder methods**.

```
class Sample():

    def __init__(self):
        self.__num__ = 7

obj = Sample()
obj.__num__
```

```
7
```

Conclusion

Congratulations! You did it. You have completed the most boring concept in **Python**. But it helps a lot when you are working with advanced code.

Most of the people like me don't understand this concept on first reading itself. So, don't lose patience reread it if you didn't get it on the first time. If you have any doubts regarding the article, don't hesitate to mention in the comment section.

If you're a **Python** beginner, I recommend you to take this course and then reread this article for the full understanding.

Resources:

- The Meaning of Underscores in Python

Use DataCamp Workspace to experiment with the code in this tutorial!

Open in Workspace

▲
76

Subscribe to RSS

About Terms Privacy