Ever so often, either while reading a question/answer on stack overflow or while going through documentations, you will come across the use of an underscore. These underscores come in all shapes and sizes, each of which have its own specific use case. All in all, there are about five different variations of these underscores that you can come across, and they are as follows:

1. Single leading underscore – Example “\_VariableName”
2. Single trailing underscore – Example “VariableName\_”
3. Double leading underscore – Example “\_\_VariableName”
4. Double leading and trailing underscore – Example “\_\_init\_\_”
5. Single underscore – Example “\_”

How this has benefited me?

So, let us go through each one at a time.

Single leading underscore – Example “\_VariableName”

Before we jump into this, let’s first take a look at a couple of examples of this specific use of underscores in python. Below, I have three different areas in which you might see the single leading underscore. These include a) before the name of a function b) before the name of a variable or class attribute and c) before the name of the class method.

def \_myfunction(myargument):

    return myargument

\_myagevariable = 26

class myclass():

    def \_myclassmethod(self):

        return self.age

The single leading underscore above is simply a convention that Python developers and certain frameworks maintain related to function, method and variable and attribute names. The convention is to use the single leading underscore for function, method and variable and attribute that are used for internal purpose only and are hence not client facing. This helps python developers uniquely identify such function, method and variable and attribute. It is important to remember that this is simply a convention used by Python developers and thereby is not enforced by the interpreter. This is to say that even if you end up using the single leading underscore for a client facing function, method and variable and attribute, the python interpreter will NOT mad and hence will NOT throw an error back at you.

There is one nuance that a developer needs to be aware of when it comes to the use of the single leading underscore, and it is related to importing function, method and variable and attribute from other modules. Lets imagine a situation in which you are trying to import functions from another python module (basically any other .py file) into your current python file.

The file that you are currently in is called main.py and you are importing from functions.py:

Functions.py

def \_internal():

    return "internal"

def external():

    return "external"

main.py

from functions import \*

>>> print(external())

"external"

>>> print(\_internal())

NameError: "name '\_internal' is not defined"

The reason why there is a NameError in this situation, even though we have imported everything from functions.py using the (from functions import \* wildcard statement), is because the internal function is defined with a single leading underscore. So do we get around this? Based on the python official documentation on this, we will need to define an \_\_all\_\_ list that overrides this behavior. In other words, if we add the following in our functions.py file, we will be able to import and use the “\_internal()” function:

\_\_all\_\_ = ['\_internal','external']

Single trailing underscore – Example “VariableName\_”

Single trailing underscores are simply used to break the name conflict with python keywords. For example, imagine a situation in which you would like to use the name “class” for a variable.

>>> class = 'myvariable'

If you attempt to run the above code in your python interpreter, you will be experience the following error:

SyntaxError: invalid syntax

This syntax error makes sense since class is a python keyword to define a class (object oriented programming paradigm), and therefore, we can not declare a variable called class and assign it a value.

However, if you still wanted to name your variable class, you can then use the single trailing underscore after the variable name and then be able to use it in python program without encountering this syntax error. See the following example:

>>> class\_ = 'myvariable'

>>> print(class\_)

myvariable

Double leading underscore – Example “\_\_VariableName”

The use of double leading underscore is something that can be used for both class methods as well as class attributes. Let us look at each one of these in an example:

Class attributes:

Class methods:

Double leading and trailing underscore – Example “\_\_init\_\_”

This one is the most common use case of double underscores in Python (also referred to as dunders), which is that you have a double leading and trailing underscore.

If you are familiar with object oriented programming in Python, you would have surely come across both of the following:

class myclass():

    def \_\_init\_\_(self, age):

        self.age = age

    def \_\_call\_\_():

        return self.age

The first one is the \_\_init\_\_ method which is basically the method used to initialize a class object, this is also called the object constructors. The other is the use of the \_\_call\_\_ method which makes the object callable.

Single underscore – Example “\_”

Nothing too fancy. Basically this "\_" single underscore is used to indicate that this is a temporary variable.

The following are the two used cases of this:

1. Lets say you have a tuple with 4 values in it:

mytuple = ('Toyota', 'Red', '2000', '250km/h')

For us to access the specific values of this mytuple object, I would need to assign each value in it to a variable. However, I am only interested in its Brand name and the max speed. So, this is what I do:

Brand, \_, \_, Max\_speed = mytuple

Now, the "Brand" variable has a value of "Toyota" and the "Max\_speed" variable has a value of "250km/h"

b. Additionally, this can be used for Python IDEs to represent the last used variable.

Example:

>>> 20 + 3

23

>>> \_

23

>>> print(\_)

23