Variable Scope in Python

When we define a variable in Python when we are not within a function or class, that variable is said to be *Global* meaning it is *visible*, (visible means we can read the variable and also alter it). If we define a variable within a function or class, it is said to be *local* to the class, and it is not visible in the global workspace, only within the function.

Making variables *local* or *global* is important in *protecting* variables from unexpected changes. This becomes critical in very large programming projects.

Python is a relatively permissive language, it does not do as well at protecting data as many other languages.

In [1]:
```
# create a variable and set it to 10

a=10
```

In [2]:
```
# create a function that has input x, and uses variables a and b

def qfunk(x):
    a=3
    b=4
    return 4*(x**a)

qfunk(2)
```

Out[2]: 32

In [3]:
```
# did the value of a change, because we used a with a different value in the
#
# Remember we set it to 10

a
```

Out[3]: 10

In [4]:
```
# can we use b now, since it was created and used in the function?

b
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[4], line 3
      1 # can we use b now, since it was created and used in the function?
----> 3 b

NameError: name 'b' is not defined
```

Scope

a did not change despite being used in the function. It was created in the *global environment*, the a within the function is in a *local environment* specific to the function.

b was not visible outside the function it was used in. It exists only as a local variable in the function

This is an example of variable scope, it protects variables from unwanted damage due to re-use of the same name

In [5]:
```
#trying to use a variable from the global environment in the function

def qfunk2(x):
    return a*x

qfunk2(3)
```

Out[5]: 30

We can use a variable defined in the global environment within a function defined in the global environment

Why is this generally a bad idea?

-If a is changed in the global environment, and we don't know, or it gets changed by mistake, we now have an error in the function

-Suppose we had intended to create a in the local environment of the function and simply forgot. If, by bad luck, a was defined in the global function, then our function would work by "borrowing" the global value. It may work in kind of uncontrolled ways.

Try not to do this....

layered environments

Here we have the global environment, an environment in qfunk3 and a third environment in qfunk4

In the example below, c is created in the qfunk3 environment and then used within qfunk4

Since qfunk4 was defined within qfunk3, it can make use of c within qfunk4 without defining it

We have a *layered environment*

Functions can *see* variables in the levels of the layer above them

In [6]:
```
# More complex example

def qfunk3(x):
    c=3
    def qfunk4(y):
        print(c*y)
    qfunk4(x)
    return(0)

qfunk3(2)
```

```
6
```

Out[6]: 0

In [7]:
```
# c is not visible in the global environment
c
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[7], line 2
      1 # c is not visible in the global environment
----> 2 c

NameError: name 'c' is not defined
```

Global variables

We have seen that Python functions can use the values of variables in the Global environment, or from environments "above" them in stacked environments (like the qfunk3 example).

We can allow a function to modify a global variable by declaring it global within the function definition

In [8]:
```
def qfunk5(x):
    global a
    a=3
    return( 3*(x**a))

qfunk5(2)
```

Out[8]: 24

In [9]:
```
# qfunk5 has now changed the value of a

a
```

Out[9]: 3

Comments

It is generally bad practice to use global variables like this in a function

There are cases where you may need to do this, or you will see it done in code, but in generally don't do it.

Pass values into a function as function inputs rather than using global variables.

If you need to send information back to the global environment, do it using return values.

If you need to return a massive amount of information, do so using a dictionary or a user-defined class

Python is bad language to learn object-oriented programming in

If you really need to learn more about object oriented programming, you will want to learn Java or C++, which have a much more reliable approach to protecting variables and creating classes.

In [ ]: