Foundational FlashCards

March 2, 2017

1 FAQ: What are the rules for local and global variables in Python?

1.1 Answer:

In Python, variables that are only referenced (**and not assigned value to**) inside a function are implicitly global. If a variable is assigned a value anywhere within the function's body, it's assumed to be a local **unless explicitly declared as global**.

Let's understand this by taking an example:

```
In [1]: def fun():
            x = v
```

In function "fun" x is implicitly local and v is implicitly global. Now, if we change the definition of the above function to:

```
In [2]: def fun():
            x = v
            v = 1
```

then v becomes implicitly local because inside the body of the function fun() v got assigned to. if one wants to keep referring to global v, then v must be declared global like this:

```
In [3]: def fun():
            global b
            x = v
            v = 1
```

In the above case v will still be global because it's explicitly declared in such manner.

We have this mechanism to avoid using global for accessing the imported modules in a function. To elaborate: If global was not implicit (to only accessed variable and not assigned variables), then every library usage inside function would have needed to be declared global.

2 FAQ: Why are default values shared between objects?

2.1 Answer

If you define a reenterant function with default parameters, like this:

```
In [4]: def re_enterant_function(knowledge_dictionary={}):
            knowledge_dictionary = {"name": "Amit"}
            #OR
            knowledge_dictionary[key] = value
            # Do something else
```

then the first time re_enterant_function() is called, the dictionary "knowledge_dictionary" will be {} as expected. But if the function is called the next time "knowledge_dictionary" will be {"name":"Amit"} and not {}.

The above is the behavior we are talking about.

By definition: immutable objects such as numbers, strings, tuples, and None, are safe from change. Changes to mutable objects such as dictionaries, lists, and class instances can lead to confusion.

so the above code should be written like the following:

```
In [6]: def re_enterant_function(knowledge_dictionary=None):
            if knowledge_dictionary == None:
                knowledge_dictionary = {}
            # Do something else
```

2.2 Tip : Memoization: Use mutable object like dictionary to create cache for memoization.

```
In [8]: # Like this:

        def function(param1, param2, __cache={}):
            if (param1, param2) in __cache:
                return __cache[(param1, param2)]
            # do some expensive work and get the result to be stored and returned
            __cache[(param1, param2)] = result
            return result
```