Python - Local and Global Variables

In general, a variable that is defined in a block is available in that block only. It is not accessible outside the block. Such a variable is called a local variable. Formal argument identifiers also behave as local variables.

The following example will underline this point. An attempt to print a local variable outside its scope will result in a NameError.

Example: Function with a Local Variable

```
def SayHello():
    user='John'
    print ("user = ", user)
    return
```

Here, `user` is a local variable for the `SayHello()` function and is not accessible outside of it.

```
>>> SayHello()
user = John
>>> user
Traceback (most recent call last):
File "<pyshell#4>", line 1, in <module>
user
NameError: name 'user' is not defined
```

Any variable present outside any function block is termed as a global variable. Its value is accessible from inside any function. In the following code snippet, ttl is initialized before the function definition. Hence, it is a global variable.

Example: Function with a Global Variable

```
user='John'
def SayHello():
    print ("user = ", user)
    return
```

Now, you can access the global variable `user` because it has been defined out of a function.

```
>>> SayHello()
user = John
>>> user
John
```

However, if we assign another value to a globally declared variable inside the function, a new local variable is created in the function's namespace. This assignment will not alter the value of the global variable. To verify this behaviour, run the following code:

```
user='John'
def SayHello():
    user='Steve'
    print ("user = ", user)
    return
```

Now, changing the value of global variable `user` inside a function will not affect its global value.

```
>>> SayHello()
user = Steve
>>> user
John
```

If you need to access and change the value of the global variable from within a function, this permission is granted by the **global** keyword.

```
user='John'
def SayHello():
    global user
    user='Steve'
    print ("user = ", user)
    return
```

Here's the output:

```
>>> user
John
>>> SayHello()
user = Steve
>>> user
Steve
```

It is also possible to use a global and local variable with the same name simultaneously. Built-in function `globals()` returns a dictionary object of all global variables and their respective values. Using the name of the variable as key, its value can be accessed and modified.

```
user='John'
def SayHello():
    globals()['user']='Mike'
    user='Steve'
    print ("user = ", user)
    return
```

The result of the above code shows a conflict between the global and local variables with the same name and how it is resolved.

```
>>> user
John
>>> SayHello()
user = Steve
>>> user
Mike
```