WORKING WITH FUNCTIONS USING PYTHON

MODULE 3/4

Mrs. SUJATA PRADHAN
PGT(SS)
AECS,ANUPURAM

Concepts to be taught…

- Scope of a variable
- LEGB rule of python
- Local scope and Global scope
- Use of global keyword

SCOPE AND LIFETIME OF VARIABLES

Scope of a variable is the portion of a program where the variable is recognized. **Parameters and variables** defined inside a function is not visible from outside. Hence, they have a **local scope**.

There are <u>**two**</u> types of scope for variables in PYTHON

i) Local Scope

ii) Global Scope

The **scope of a variable** depends on the location where the variable is being declared. The variable declared in one part of the program is not accessible to other parts of the program.

The variable defined outside any function has a **global scope** whereas the variable defined inside a function has a **local scop**e. Local scope refers to variables defined in current function . Always, a function will first look up for a variable name in its local scope. Only if it does not find it there, the outer scopes are checked.

Local scope

```
def test():
    X = 5
    print(X)
```

The local scope of a variable is its function. As we know, each **variable name** belongs to a **namespace**. If a variable is created in a particular function, then its scope is within that **function's namespace .** So any variable inside the function will be local to that namespace. In the above example, the scope of the variable X is **test()** only.

```
def  message():
    M= "hello ”
    print(M)

message()
print(M)
```

print() statement outside the function will cause an error as variable M has a local scope.

M can not be accessible outside the function.

EXAMPLE

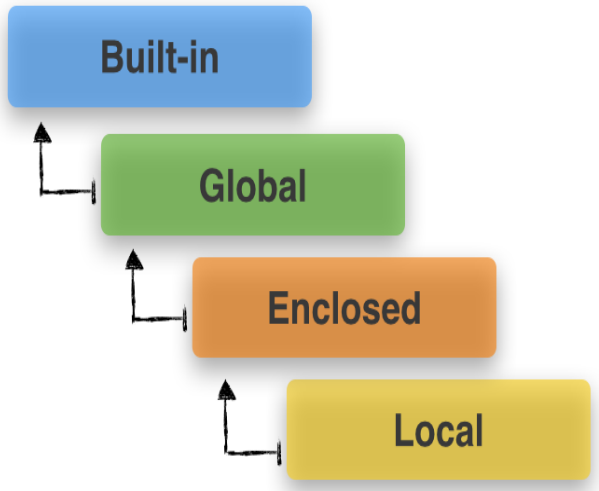
```
A = ‘Global variable'
def  fun():
     print(A, ' inside a function')
fun()
print(A, ‘outside a function')
```

OUTPUT

```
Global variable inside function
Global variable outside function
```

We call **fun()** first, which is supposed to print the value of **A**. The function will first look in its own **local scope o**f A if is defined there. Since func() does not define its own **A**, it will look one-level above in the global scope in which **A** has been defined previously.

Scope resolution via LEGB rule



In Python, the **LEGB rule** is used to decide the order in which the namespaces are to be searched for scope resolution.
The scopes are listed below in terms of hierarchy from highest to lowest.

- Local(L): Defined inside function
- Enclosed(E): Defined inside enclosing functions(Nestedfunction)
- Global(G): Defined at the uppermost level
- Built-in(B): Reserved names in Python built in modules

Scope resolution for variable names

namespace-hierarchy search order
Local -> Enclosed -> Global -> Built-in

Local can be inside a function.

Enclosed can be its enclosing function, e.g., if a function is wrapped inside another function.

Global refers to the uppermost level of the executing entire program.

Built-in are special names that Python reserves for itself.

So, if a particular **name:object** mapping cannot be found in the local namespace, the namespace of the enclosed scope is being searched next.If the search in the enclosed scope is unsuccessful too, Python moves on to the global namespace and eventually, it will search the built-in namespace.If a name cannot be found in any of the namespaces, a **NameError** will be raised.

EXAMPLE

```
def  my_func():
     X = 10
     print("Value inside function:",X)

X = 20
print("Value outside function:", X)
my_func()
print("Value outside function:", X)
```

OUTPUT:

```
Value outside function: 20
Value inside function: 10
Value outside function: 20
```

- **Here, we can see that the value of X is 20 initially.**
- **Even though the function my_func() changed the value of X to 10, it did not affect the value outside the function.**
- **This is because the variable X inside the function is different (local to the function) from the one outside.**
- **Although they have same names, they are two different variables with two different scope.**

EXAMPLE (using GLOBAL AND LOCAL scope)

```
X=10

def exam():
    print(X)

def test():
    X = 5
    print(X)

def marks(X):
    print(X)

print(X)
exam()
test()
marks(20)
```

output:

```
10
10
5
20
```

- The first line creates a variable X that belongs to the namespace of the file, so its scope is the entire file. Hence print(X) displays 10.
- The exam() function creates its namespace, but that namespace doesn't have an X in it. As Python doesn't find X there, it checks the next larger enclosing namespace and finds X. So exam uses the variable X defined at the top and displays 10.
- However, the test() function defines its own variable named X with value 5, which has higher priority over the first definition of X. So any mention of X within the test function will refer to that X, hence displaying 5.
- The marks() function also has an X in its own namespace just like test() function has. So X gets bound to whatever value is passed as an argument to marks() function. Hence the outer X is shadowed again in this function displaying the output as 20.

NOTE : It is possible to **modify the global variable** by **re-assigning a new value** to it if we use the **global** keyword**.** In order to modify the value of variables outside the function, they must be declared as global variables using the keyword **global**

PROGRAM 1

```
A = ‘G value'
def func():
      A= ‘L value'
      print(A , ’inside function’)

print(A, ‘outside function')
func()
print(A, 'outside function')
```

```
OUTPUT
G value outside function
L value inside function
G value outside function
```

PROGRAM 2

```
A = ‘G value'
def func():
      global  A
      A= ‘L value'
      print(A , ’inside function’)

print(A, ‘outside function')
func()
print(A, 'outside function')
```

```
OUTPUT
G value outside function
L value inside function
L value outside  function
```

Referenced Before Assignment

```
a=1
def fun():
	global a
	a+=1
	print(a)

print(a)
fun()
print(a)

OUTPUT
1
2
2
```

```
a = 1
def  fun():
	a=5
	a = a + 1
	print(a)
print(a)
fun()
print(a)

OUTPUT
1
6
1
```

```
a = 1
def  fun():
	a = a + 1
	print(a)
print(a)
fun()
print(a)
OUTPUT
UnboundLocalError: local variable 'a' referenced before
	assignment
```

conclusion:

Local Scope:

- **Variable used inside the function has local scope.**
- **It cannot be accessed outside the function.**
- **In this scope, the lifetime of a variable inside a function is as long as the function executes.**
- **They are destroyed once we return from the function. Hence, a function does not remember the value of a variable from its previous calls.**

Global Scope:

- **Variable can be accessed outside the function.**
- **In this scope, Lifetime of a variable is the period throughout which the variable exists in the memory.**
- **On the other hand, variables outside the function are visible from inside. They have a global scope.**
- **We can read these values from inside the function but cannot change (write) them.**
- **In order to modify the value of variables outside the function, they must be declared as global variables using the keyword global.**

WORKSHEET

1. Define scope of a variable.
2. What is LEGB rule? Explain it.
3. What is the difference between local and global variable?
4. What is the significance of global keyword?
5. What is scope and what is the scope resolving rule in python.
6. What will be the output of the following program?

```
num=1
def myfunc():
    global num
    num=10
    return num
print(num)
print(myfunc())
print(num)
```

7. Name the local,global variables and inbuilt functions used in the program and write the output.

```
L=”big names”
pos=200
level=1
def play():
    max=level+10
    print(len(L)==0)
    return max
res=play()
print(res)
```

WORKSHEET

8. Find and write the output of the following Python code:-

```
x=1
def cg():
  global x
  x=x+1
cg()
print(x)
```

9. What will be the output of the following Python code?

```
def f():
   global a
   print(a)
   a = "hello“
   print(a)
a = "world“
f()
print(a)
```

10. What will be the output of the following Python code?

```
x = 5
def f1():
     global x
     x = 4
def f2(a,b):
     global x
     return a+b+x
 f1()
total = f2(1,2)
print(total)
```

THANK YOU