Python Variable Scope – Local, Global, Built-in, Enclosed

Today, we will discuss Python Variable Scope. Here, we will learn different types of variable scope in Python, Python Global Keyword and Python Non-local keywords.

So, let's start our learning.

What is Python Variable Scope?

The scope of a variable in **python** is that part of the code where it is visible. Actually, to refer to it, you don't need to use any prefixes then.

Let's take an example, but before let's revise **python Syntax**.

```
>>> b=8
>>> def func():
        a=7
        print(a)
        print(b)
>>> func()
```

Output

```
7
8
```

```
>>> a
```

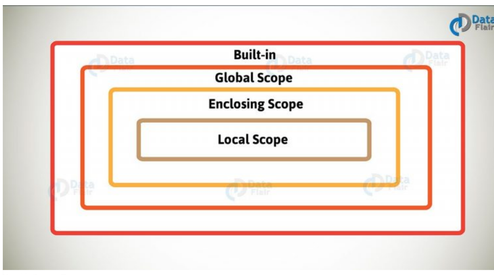
Output

```
Traceback (most recent call last):
File "<pyshell#6>", line 1, in <module>
a
NameError: name 'a' is not defined
```

Also, the duration for which a variable is alive is called its 'lifetime'.

Types of Python Variable Scope

There are 4 types of Variable Scope in **Python**, let's discuss them one by one:



Python Variable Scope – Types

1. Local Scope in Python

In the above code, we define a variable 'a' in a function 'func'. So, 'a' is local to 'func'. Hence, we can read/write it in func, but not outside it.

When we try to do so, it raises a NameError. Look at this code.

```
>>> a=2
>>> def func():
        print(a)
        a=1
        print(a)
>>> func()
```

Output

```
Traceback (most recent call last):
File "<pyshell#79>", line 1, in <module>
func()
File "<pyshell#78>", line 2, in func
print(a)
UnboundLocalError: local variable 'a' referenced before assignment
```

Here, we could've accessed the global Scope 'a' inside 'func', but since we also define a local 'a' in it, it no longer accesses the global 'a'.

Then, the first print statement inside 'func' throws an **error in Python**, because we're trying to access the local scope 'a' before assigning it.

However, it is bad practice to try to manipulate global values from inside local scopes. Try to pass it as a parameter to the function.

```
>>> def func(a=0):
        a+=1
        print(a)
>>> func()
```

Output

```
1
```

2. Global Scope in Python

We also declare a variable 'b' outside any other python Variable scope, this makes it global scope.

Consequently, we can read it anywhere in the program. Later in this article, we will see how to write it inside func.

3. Enclosing Scope in Python

Let's take another example.

```
>>> def red():
        a=1
        def blue():
            b=2
            print(a)
            print(b)
        blue()
        print(a)
>>> red()
```

Output

```
1
2
1
```

In this code, 'b' has local scope in **Python function** 'blue', and 'a' has nonlocal scope in 'blue'.

Of course, a python variable scope that isn't global or local is nonlocal. This is also called the enclosing scope.

4. Built-in Scope

Finally, we talk about the widest scope. The built-in scope has all the names that are loaded into python variable scope when we start the interpreter.

For example, we never need to import any module to access functions like print() and id().

Global Keyword in Python

So far, we haven't had any kind of a problem with global scope. So let's take an example.

```
>>> a=1
>>> def counter():
        a=2
        print(a)
>>> counter()
```

Output

```
2
```

Now, when we make a reference to 'a' outside this function, we get 1 instead of 2.

```
>>> a
```

Output

```
1
```

Why does this happen? Well, this is because when we set 'a' to 2, it created a local variable 'a' in the local scope of 'counter'.

This didn't change anything for the global 'a'. Now, what if you wanted to change the global version of 'a'? We use the 'global' keyword in python for this.

```
>>> a=1
>>> def counter():
        global a
        a=2
        print(a)
>>> counter()
```

Output

```
2
```

```
>>> a
```

Output

```
2
```

What we do here is, we declare that the 'a' we're going to use in this function is from the global scope.

After this, whenever we make a reference to 'a' inside 'counter', the interpreter knows we're talking about the global 'a'.

In this example, it changed the value of the global 'a' to 2.

Nonlocal Keyword in Python

Like the 'global' keyword, you want to make a change to a nonlocal variable, you must use the 'nonlocal' keyword. Let's first try this without the keyword.

```
>>> def red():
        a=1
        def blue():
            a=2
            b=2
            print(a)
            print(b)
        blue()
        print(a)
>>> red()
```

Output

```
2
2
1
```

As you can see, this did not change the value of 'a' outside function 'blue'. To be able to do that, we use 'nonlocal'.

```
>>> def red():
        a=1
        def blue():
            nonlocal a
            a=2
            b=2
            print(a)
            print(b)
        blue()
        print(a)
>>> red()
```

Output

```
2
2
2
```

See? This works perfectly fine.