Python Variable Scope

In Python, we can declare variables in three different scopes: local scope, global, and nonlocal scope.

A variable scope specifies the region where we can access a variable. For example,

```
def add_numbers():
    sum = 5 + 4
```

Here, the *sum* variable is created inside the function, so it can only be accessed within it (local scope). This type of variable is called a local variable.

Based on the scope, we can classify Python variables into three types:

1. Local Variables
2. Global Variables
3. Nonlocal Variables

Python Local Variables

When we declare variables inside a function, these variables will have a local scope (within the function). We cannot access them outside the function.

These types of variables are called local variables. For example,

```
def greet():

    # local variable
    message = 'Hello'

    print('Local', message)

greet()

# try to access message variable
# outside greet() function
print(message)
```

Output

```
Local Hello
NameError: name 'message' is not defined
```

Here, the *message* variable is local to the `greet()` function, so it can only be accessed within the function.

That's why we get an error when we try to access it outside the `greet()` function.

To fix this issue, we can make the variable named *message* global.

Python Global Variables

In Python, a variable declared outside of the function or in global scope is known as a global variable. This means that a global variable can be

accessed inside or outside of the function.

Let's see an example of how a global variable is created in Python.

```
# declare global variable
message = 'Hello'

def greet():
    # declare local variable
    print('Local', message)

greet()
print('Global', message)
```

Output

```
Local Hello
Global Hello
```

This time we can access the *message* variable from outside of the `greet()` function. This is because we have created the *message* variable as the global variable.

```
# declare global variable
message = 'Hello'
```

Now, *message* will be accessible from any scope (region) of the program.

Python Nonlocal Variables

In Python, the `nonlocal` keyword is used within nested functions to indicate that a variable is not local to the inner function, but rather belongs to an enclosing functionâ€™s scope.

This allows you to modify a variable from the outer function within the nested function, while still keeping it distinct from global variables.

```
# outside function
def outer():
    message = 'local'

    # nested function
    def inner():

        # declare nonlocal variable
        nonlocal message

        message = 'nonlocal'
        print("inner:", message)

    inner()
    print("outer:", message)

outer()
```

Output

```
inner: nonlocal
outer: nonlocal
```

In the above example, there is a nested `inner()` function. The `inner()` function is defined in the scope of another function `outer()`.

We have used the `nonlocal` keyword to modify the `message` variable from the outer function within the nested function.

Note : If we change the value of a nonlocal variable, the changes appear in the local variable.

Also Read:

- Python Namespace and Scope

Table of Contents

- Nonlocal Variables in Python