Python Global, Local and Nonlocal variables

programiz.com/python-programming/global-local-nonlocal-variables

Join our newsletter for the latest updates.

In this tutorial, you'll learn about Python Global variables, Local variables, Nonlocal variables and where to use them.

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.

Example 1: Create a Global Variable

```
x = "global"

def foo():
    print("x inside:", x)


foo()
print("x outside:", x)
```

Output

```
x inside: global
x outside: global
```

In the above code, we created *x* as a global variable and defined a `foo()` to print the global variable *x*. Finally, we call the `foo()` which will print the value of *x*.

What if you want to change the value of *x* inside a function?

```
x = "global"

def foo():
    x = x * 2
    print(x)

foo()
```

Output

```
UnboundLocalError: local variable 'x' referenced before assignment
```

The output shows an error because Python treats *x* as a local variable and *x* is also not defined inside `foo()`.

To make this work, we use the `global` keyword. Visit Python Global Keyword to learn more.

Local Variables

A variable declared inside the function's body or in the local scope is known as a local variable.

Example 2: Accessing local variable outside the scope

```
def foo():
    y = "local"


foo()
print(y)
```

Output

```
NameError: name 'y' is not defined
```

The output shows an error because we are trying to access a local variable *y* in a global scope whereas the local variable only works inside `foo()` or local scope.

Let's see an example on how a local variable is created in Python.

Example 3: Create a Local Variable

Normally, we declare a variable inside the function to create a local variable.

```
def foo():
    y = "local"
    print(y)

foo()
```

Output

```
local
```

Let's take a look at the earlier problem where *x* was a global variable and we wanted to modify *x* inside `foo()`.

Global and local variables

Here, we will show how to use global variables and local variables in the same code.

Example 4: Using Global and Local variables in the same code

```
x = "global "

def foo():
    global x
    y = "local"
    x = x * 2
    print(x)
    print(y)

foo()
```

Output

```
global global
local
```

In the above code, we declare *x* as a global and *y* as a local variable in the `foo()`. Then, we use multiplication operator `*` to modify the global variable *x* and we print both *x* and *y*.

After calling the `foo()`, the value of *x* becomes `global global` because we used the `x * 2` to print two times `global`. After that, we print the value of local variable *y* i.e `local`.

Example 5: Global variable and Local variable with same name

```
x = 5

def foo():
    x = 10
    print("local x:", x)


foo()
print("global x:", x)
```

Output

```
local x: 10
global x: 5
```

In the above code, we used the same name *x* for both global variable and local variable. We get a different result when we print the same variable because the variable is declared in both scopes, i.e. the local scope inside `foo()` and global scope outside `foo()`.

When we print the variable inside `foo()` it outputs `local x: 10`. This is called the local scope of the variable.

Similarly, when we print the variable outside the `foo()`, it outputs `global x: 5`. This is called the global scope of the variable.

Nonlocal Variables

Nonlocal variables are used in nested functions whose local scope is not defined. This means that the variable can be neither in the local nor the global scope.

Let's see an example of how a nonlocal variable is used in Python.

We use `nonlocal` keywords to create nonlocal variables.

Example 6: Create a nonlocal variable

```
def outer():
    x = "local"

    def inner():
        nonlocal x
        x = "nonlocal"
        print("inner:", x)

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


outer()
```

Output

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

In the above code, there is a nested `inner()` function. We use `nonlocal` keywords to create a nonlocal variable. The `inner()` function is defined in the scope of another function `outer()`.

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