Python Global Keyword

programiz.com/python-programming/global-keyword

Join our newsletter for the latest updates.

In this article, you'll learn about the global keyword, global variable and when to use global keywords.

Before reading this article, make sure you have got some basics of Python Global, Local and Nonlocal Variables.

What is the global keyword

In Python, `global` keyword allows you to modify the variable outside of the current scope. It is used to create a global variable and make changes to the variable in a local context.

Rules of global Keyword

The basic rules for `global` keyword in Python are:

- When we create a variable inside a function, it is local by default.
- When we define a variable outside of a function, it is global by default. You don't have to use `global` keyword.
- We use `global` keyword to read and write a global variable inside a function.
- Use of `global` keyword outside a function has no effect.

Use of global Keyword

Let's take an example.

Example 1: Accessing global Variable From Inside a Function

```
c = 1 # global variable

def add():
    print(c)

add()
```

When we run the above program, the output will be:

```
1
```

However, we may have some scenarios where we need to modify the global variable from inside a function.

Example 2: Modifying Global Variable From Inside the Function

```
c = 1 # global variable

def add():
    c = c + 2 # increment c by 2
    print(c)

add()
```

When we run the above program, the output shows an error:

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

This is because we can only access the global variable but cannot modify it from inside the function.

The solution for this is to use the `global` keyword.

Example 3: Changing Global Variable From Inside a Function using global

```
c = 0 # global variable

def add():
    global c
    c = c + 2 # increment by 2
    print("Inside add():", c)

add()
print("In main:", c)
```

When we run the above program, the output will be:

```
Inside add(): 2
In main: 2
```

In the above program, we define *c* as a global keyword inside the `add()` function.

Then, we increment the variable *c* by `1`, i.e `c = c + 2`. After that, we call the `add()` function. Finally, we print the global variable *c*.

As we can see, change also occurred on the global variable outside the function, `c = 2`.

Global Variables Across Python Modules

In Python, we create a single module `config.py` to hold global variables and share information across Python modules within the same program.

Here is how we can share global variables across the python modules.

Example 4: Share a global Variable Across Python Modules

Create a `config.py` file, to store global variables

```
a = 0
b = "empty"
```

Create a `update.py` file, to change global variables

```
import config

config.a = 10
config.b = "alphabet"
```

Create a `main.py` file, to test changes in value

```
import config
import update

print(config.a)
print(config.b)
```

When we run the `main.py` file, the output will be

```
10
alphabet
```

In the above, we have created three files: `config.py`, `update.py`, and `main.py`.

The module `config.py` stores global variables of *a* and *b*. In the `update.py` file, we import the `config.py` module and modify the values of *a* and *b*. Similarly, in the `main.py` file, we import both `config.py` and `update.py` module. Finally, we print and test the values of global variables whether they are changed or not.

Global in Nested Functions

Here is how you can use a global variable in nested function.

Example 5: Using a Global Variable in Nested Function

```
def foo():
    x = 20

    def bar():
        global x
        x = 25

    print("Before calling bar: ", x)
    print("Calling bar now")
    bar()
    print("After calling bar: ", x)

foo()
print("x in main: ", x)
```

The output is :

```
Before calling bar: 20
Calling bar now
After calling bar: 20
x in main: 25
```

In the above program, we declared a global variable inside the nested function `bar()`. Inside `foo()` function, *x* has no effect of the global keyword.

Before and after calling `bar()`, the variable *x* takes the value of local variable i.e `x = 20`. Outside of the `foo()` function, the variable *x* will take value defined in the `bar()` function i.e `x = 25`. This is because we have used `global` keyword in *x* to create global variable inside the `bar()` function (local scope).

If we make any changes inside the `bar()` function, the changes appear outside the local scope, i.e. `foo()`.