Python Global Keyword

In Python, the `global` keyword allows us 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.

Before we learn about the `global` keyword, make sure you have got some basics of Python Variable Scope.

Access and Modify Python Global Variable

First let's try to access a global variable from the inside of a function,

```
c = 1 # global variable

def add():
    print(c)

add()

# Output: 1
```

Here, we can see that we have accessed a global variable from the inside of a function.

However, if we try to modify the global variable from inside a function as:

```
# global variable
c = 1

def add():

     # increment c by 2
    c = c + 2

    print(c)

add()
```

Output

```
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: Changing Global Variable From Inside a Function using global

```
# global variable
c = 1

def add():

    # use of global keyword
    global c
    # increment c by 2
    c = c + 2

    print(c)

add()

# Output: 3
```

In the above example, we have defined *c* as the `global` keyword inside `add()`.

Then, we have incremented the variable *c* by **2**, i.e `c = c + 2`.

As we can see while calling `add()`, the value of global variable *c* is modified from **1** to **3**.

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 the `global` keyword.
- We use the `global` keyword to modify (write to) a global variable inside a function.
- Use of the `global` keyword outside a function has no effect.

Also Read:

- Python Variables, Constants and Literals
- Python globals()

Table of Contents