**Python Variables: Declare, Concatenate, Global & Local**

## **What is a Variable in Python?**

A Python variable is a reserved memory location to store values. In other words, a variable in a python program gives data to the computer for processing.

Every value in Python has a datatype. Different data types in Python are Numbers, List, Tuple, Strings, Dictionary, etc. Variables can be declared by any name or even alphabets like a, aa, abc, etc.

## **How to Declare and use a Variable**

Let see an example. We will declare variable "a" and print it.

a=100

print (a)

## **Re-declare a Variable**

You can re-declare the variable even after you have declared it once.

Here we have variable initialized to f=0.

Later, we re-assign the variable f to value "guru99"

# Declare a variable and initialize it

f = 0

print f

# re-declaring the variable works

f = 'guru99'

print f

## **Concatenate Variables**

Let's see whether you can concatenate different data types like string and number together. For example, we will concatenate "Guru" with the number "99".

Unlike Java, which concatenates number with string without declaring number as string, Python requires declaring the number as string otherwise it will show a TypeError

For the following code, you will get undefined output -

a="Guru"

b = 99

print a+b

Once the integer is declared as string, it can concatenate both "Guru" + **str**("99")= "Guru99" in the output.

a="Guru"

b = 99

print(a+str(b))

For the following code, you will get undefined output -

a="Guru"

b = 99

print a+b

Once the integer is declared as string, it can concatenate both "Guru" + **str**("99")= "Guru99" in the output.

a="Guru"

b = 99

print(a+str(b))

## **Local & Global Variables**

In Python when you want to use the same variable for rest of your program or module you declare it a global variable, while if you want to use the variable in a specific function or method, you use a local variable.

Let's understand this difference between local and global variable with the below program.

1. Variable "f" is **global** in scope and is assigned value 101 which is printed in output
2. Variable f is again declared in function and assumes **local** scope. It is assigned value "I am learning Python." which is printed out as an output. This variable is different from the global variable "f" define earlier
3. Once the function call is over, the local variable f is destroyed. At line 12, when we again, print the value of "f" is it displays the value of global variable f=101
4. # Declare a variable and initialize it
5. f = 101
6. print(f)
7. # Global vs. local variables in functions
8. def someFunction():
9. # global f
10. f = 'I am learning Python'
11. print(f)
12. someFunction()
13. print(f)

Using the keyword **global,**you can reference the global variable inside a function.

1. Variable "f" is **global** in scope and is assigned value 101 which is printed in output
2. Variable f is declared using the keyword **global**. This is **NOT** a **local variable**, but the same global variable declared earlier. Hence when we print its value, the output is 101
3. We changed the value of "f" inside the function. Once the function call is over, the changed value of the variable "f" persists. At line 12, when we again, print the value of "f" is it displays the value "changing global variable"
4. f = 101;
5. print(f)
6. # Global vs.local variables in functions
7. def someFunction():
8. global f
9. print(f)
10. f = "changing global variable"
11. someFunction()
12. print(f)

## **Delete a variable**

You can also delete variable using the command **del** "variable name".

In the example below, we deleted variable f, and when we proceed to print it, we get error "**variable name is not defined**" which means you have deleted the variable.

f = 11;

print(f)

del f

print(f)

## **Private Variables**

“Private” instance variables that cannot be accessed except from inside an object don’t exist in Python. However, there is a convention that is followed by most Python code: a name prefixed with an underscore (e.g. \_spam) should be treated as a non-public part of the API (whether it is a function, a method or a data member). It should be considered an implementation detail and subject to change without notice.

Since there is a valid use-case for class-private members (namely to avoid name clashes of names with names defined by subclasses), there is limited support for such a mechanism, called name mangling. Any identifier of the form \_\_spam (at least two leading underscores, at most one trailing underscore) is textually replaced with \_classname\_\_spam, where classname is the current class name with leading underscore(s) stripped. This mangling is done without regard to the syntactic position of the identifier, as long as it occurs within the definition of a class.