**Python Variable** is containers that store values. Python is not “statically typed”. We do not need to declare variables before using them or declare their type. A variable is created the moment we first assign a value to it. A Python variable is a name given to a memory location. An Example of a Variable in Python is a representational name that serves as a pointer to an object. Once an object is assigned to a variable, it can be referred to by that name. The value of a variable can change throughout the program, hence the term "variable."

A Variables in Python is only a name given to a memory location, all the operations done on the variable effects that memory location.

Rules for Python variables

1. A Python variable name must start with a letter or the underscore character.
2. A Python variable name cannot start with a number.
3. A Python variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _ ).
4. Variable in Python names are case-sensitive (name, Name, and NAME are three different variables).
5. The reserved words(keywords) in Python cannot be used to name the variable in Python.

Python handles memory allocation and deallocation automatically through a process called garbage collection. When a variable is no longer referenced by any part of the program, Python will free up the memory associated with it.

```
x = 5
salary = 1456.8
name = "Govind"
is_student = True
print("integer = ",x," float = ",salary," string = ", name," boolean = ",is_student)
```

```
integer =  5  float =  1456.8  string =  Govind  boolean =  True
```

```
# Python is a dynamically typed language, meaning you don’t need to declare the type of a variable when you create it.
a = 10      # a is an integer
a = "Hello" # a is now a string
print(a) # a changes from an integer to a string when a new value is assigned.
```

```
Hello
```

```
# Multiple assignment
x, y, z = 1, 2, 3.2
a = b = c = 10
print(x,y,z,a,b,c)
```

```
1 2 3.2 10 10 10
```

```
# Swapping Values Between Variables
x , y = y , x
print(x,y)
```

```
2 1
```

```
# Python doesn’t have built-in constant types. However, by convention, variables that should not change are named using upperca
PI = 3.14159
print(PI)
# This doesn’t enforce immutability but signals to other developers that the variable should be treated as a constant.
```

```
3.14159
```

```
# Type casting we will see in detail later
x = "123"
y = int(x) # y is now an integer with value 123
print(y)
```

```
123
```

```
a = 10
b = 20
print(a+b)

c = "Govind"
d = "Parab"
print(c+d)
# print(a+c) This will give type error
```

```
30
GovindParab
```

```
# Global and Local Python Variables
# Variables defined outside of a function are global and can be accessed from anywhere in the code.
# Variables defined inside a function are local and can only be accessed within that function.

# Variable shadowing occurs when a local variable has the same name as a global variable. In such cases, the local variable "sh


# This function uses local variable s
def func():
    s = "Hello World!"
    print(s)

func()

# This function has a variable with
# name same as r
def func2():
    print(r)

# Global scope
r = "radius"
func2()
```

```
Hello World!
radius
```

```
#local variable
a = 10
def my_func1():
  a = 5 #local variable to this function
  print("inside1 function a = ",a)

  def my_func2():
    a = 2 #local variable to this function
    print("inside2 function a = ",a)

  my_func2()

my_func1()
my_func1()

print("outside function a = ",a)
```

```
inside1 function a =  5
inside2 function a =  2
inside1 function a =  5
inside2 function a =  2
outside function a =  10
```

```
#global variable
a = 10
def my_func1():
  a = 5 #local variable to this function
  print("inside1 function a = ",a)
  # global b # this is not a recommended to do
  # b = 30

  def my_func2():
    global a
    a = 2 #accessing the global variable
    print("inside2 function a = ",a)

  my_func2()

my_func1()
my_func1()

print("outside function a = ",a)
# print("outside function b = ",b)
```

```
inside1 function a =  5
inside2 function a =  2
inside1 function a =  5
```

```
inside2 function a =  2
outside function a =  2
```

```
#nonlocal variable
outer_a = 20 #global var
def my_func1():
  inner1_a = 10 #local var
  print("inside1 function a = ",inner1_a)

  def my_func2():
    inner2_a = 5 #local var
    print("inside2 function a = ",inner2_a)

    def my_func3():
      inner3_a = 2 #local var
      print("inside3 function a = ",inner3_a)

      #suppose we want to change the value of inner2_a to 100
      #but we know that inner2_a is not a global var so we can't access it from anywhere except my_func2()
      #here if we try to do the following then it create a new var in this function local scope and it won't change the value c
      #inner2_a = 100
      #print("inside3 function inner2_a = ",inner2_a)
      #this is not what we wanted we want to change the value of inner2_a in outer function i.e my_func2()
      #so to achieve this we can use nonlocal keyword which specifies that var is not local to current function and is neither
      #this will cause python to search it in outer scope rather than in global or local scope
      nonlocal inner2_a
      inner2_a = 100
      print("inside3 function inner2_a = ",inner2_a)

    print("before calling myfunc3() inside2 function a = ",inner2_a)
    my_func3()
    print("after calling myfunc3() inside2 function a = ",inner2_a)

  my_func2()

my_func1()
my_func1()

print("outside function a = ",outer_a)
```

```
inside1 function a =  10
inside2 function a =  5
before calling myfunc3() inside2 function a =  5
inside3 function a =  2
inside3 function inner2_a =  100
after calling myfunc3() inside2 function a =  100
inside1 function a =  10
inside2 function a =  5
before calling myfunc3() inside2 function a =  5
inside3 function a =  2
inside3 function inner2_a =  100
after calling myfunc3() inside2 function a =  100
outside function a =  20
```

```
class Student:
    # Class Variable
    stream = 'msc'
    # The init method or constructor
    def __init__(self, roll):
        # Instance Variable
        self.roll = roll

# Objects of Student class
a = Student(101)
b = Student(102)

print(a.stream)  # prints "msc"
print(b.stream)  # prints "msc"
print(a.roll)    # prints 101

# Class variables can be accessed using class name also
print(Student.stream)  # prints "msc"
```

```
msc
msc
101
msc
```

```
outer_a = 20

def my_func1():
    inner1_a = 10
    print("inside1 - before change, inner1_a =", inner1_a)

    def my_func2():
        print("inside2 - accessing inner1_a of my_func1()=", inner1_a) # read-only
        def my_func3():
            nonlocal inner1_a  # Refers to inner1_a in my_func1
            inner1_a = 999
            print("inside3 - changed inner1_a to", inner1_a)

        my_func3()

    my_func2()
    print("inside1 - after change, inner1_a =", inner1_a)

my_func1()


# Python will look up the nearest enclosing non-global function to find inner1_a.
# It will skip the local scope of my_func3(), then look in my_func2() (doesn't exist there), then in my_func1() — and there it f

# inner1_a is defined in my_func1().

# my_func2() is defined inside my_func1().

# Therefore, my_func2() can see and access inner1_a (read-only unless nonlocal is used).
```

```
inside1 - before change, inner1_a = 10
inside2 - accessing inner1_a of my_func1()= 10
inside3 - changed inner1_a to 999
inside1 - after change, inner1_a = 999
```