Python Variables

Variable is a name that is used to refer to memory location. Python variable is also known as an identifier and used to hold value.

In Python, we don't need to specify the type of variable because Python is a infer language and smart enough to get variable type.

Variable names can be a group of both the letters and digits, but they have to begin with a letter or an underscore.

It is recommended to use lowercase letters for the variable name. Rahul and rahul both are two different variables.

Identifier Naming

Variables are the example of identifiers. An Identifier is used to identify the literals used in the program. The rules to name an identifier are given below.

- The first character of the variable must be an alphabet or underscore (_).
- All the characters except the first character may be an alphabet of lower-case(a-z), upper-case (A-Z), underscore, or digit (0-9).
- Identifier name must not contain any white-space, or special character (!, @, #, %, ^, &, *).
- Identifier name must not be similar to any keyword defined in the language.
- Identifier names are case sensitive; for example, my_name, and My_Name is not the same.
- Examples of valid identifiers: a123, _n, n_9, etc.
- Examples of invalid identifiers: 1a, n%4, n 9, etc.

Declaring Variable and Assigning Values

Python does not bind us to declare a variable before using it in the application. It allows us to create a variable at the required time.

We don't need to declare explicitly variable in Python. When we assign any value to the variable, that variable is declared automatically.

The equal (=) operator is used to assign value to a variable.

Object References

It is necessary to understand how the Python interpreter works when we declare a variable. The process of treating variables is somewhat different from many other programming languages.

Python is the highly object-oriented programming language; that's why every data item belongs to a specific type of class. Consider the following example.

```
1. print("John")
```

Output:

```
John
```

The Python object creates an integer object and displays it to the console. In the above print statement, we have created a string object. Let's check the type of it using the Python built-in **type()** function.

```
1. type("John")
```

Output:

```
<class 'str'>
```

In Python, variables are a symbolic name that is a reference or pointer to an object. The variables are used to denote objects by that name.

Let's understand the following example

```
1. a = 50
```

In the above image, the variable **a** refers to an integer object.

Suppose we assign the integer value 50 to a new variable b.

```
a = 50
```

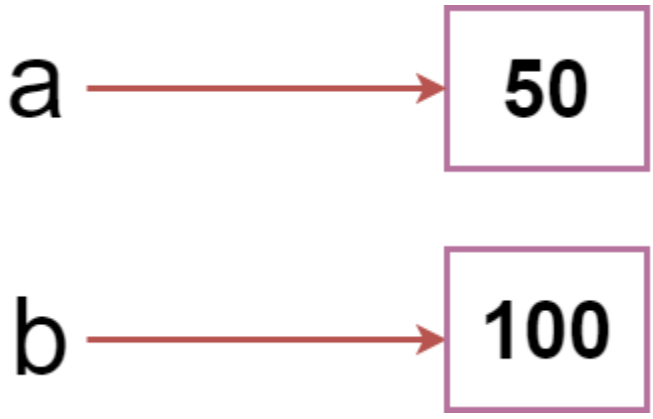
```
b = a
```

The variable b refers to the same object that a points to because Python does not create another object.

Let's assign the new value to b. Now both variables will refer to the different objects.

```
a = 50

b =100
```



Python manages memory efficiently if we assign the same variable to two different values.

Object Identity

In Python, every created object identifies uniquely in Python. Python provides the guaranteed that no two objects will have the same identifier. The built-in **id()** function, is used to identify the object identifier. Consider the following example.

1. a = 50
2. b = a
3. print(id(a))
4. print(id(b))

```
# Reassigned variable a
a = 500
print(id(a))
```

Output:

```
140734982691168
140734982691168
2822056960944
```

We assigned the **b = a, a** and **b** both point to the same object. When we checked by the **id()** function it returned the same number. We reassign **a** to 500; then it referred to the new object identifier.

Variable Names

We have already discussed how to declare the valid variable. Variable names can be any length can have uppercase, lowercase (A to Z, a to z), the digit (0-9), and underscore character(_). Consider the following example of valid variables names.

```
name = "Devansh"
age = 20
marks = 80.50

print(name)
print(age)
print(marks)
```

Output:

```
Devansh
20
80.5
```

Consider the following valid variables name.

```
name = "A"
Name = "B"
naMe = "C"
NAME = "D"
n_a_m_e = "E"
_name = "F"
```

```
7. name_ = "G"
8. _name_ = "H"
9. na56me = "I"
10. 
11. print(name,Name,naMe,NAME,n_a_m_e, NAME, n_a_m_e, _name, name_,_name, na5
    6me)
```

Output:

```
A B C D E D E F G F I
```

In the above example, we have declared a few valid variable names such as name, _name_ , etc. But it is not recommended because when we try to read code, it may create confusion. The variable name should be descriptive to make code more readable.

The multi-word keywords can be created by the following method.

- **Camel Case -** In the camel case, each word or abbreviation in the middle of begins with a capital letter. There is no intervention of whitespace. For example - nameOfStudent, valueOfVaraible, etc.
- **Pascal Case -** It is the same as the Camel Case, but here the first word is also capital. For example - NameOfStudent, etc.
- **Snake Case -** In the snake case, Words are separated by the underscore. For example - name_of_student, etc.

Multiple Assignment

Python allows us to assign a value to multiple variables in a single statement, which is also known as multiple assignments.

We can apply multiple assignments in two ways, either by assigning a single value to multiple variables or assigning multiple values to multiple variables. Consider the following example.

1. Assigning single value to multiple variables

Eg:

```
1. x=y=z=50
2. print(x)
3. print(y)
```

4. print(z)

Output:

```
50
50
50
```

2. Assigning multiple values to multiple variables:

Eg:

1. a,b,c=5,10,15
2. print a
3. print b
4. print c

Output:

```
5
10
15
```

The values will be assigned in the order in which variables appear.

Python Variable Types

There are two types of variables in Python - Local variable and Global variable. Let's understand the following variables.

Local Variable

Local variables are the variables that declared inside the function and have scope within the function. Let's understand the following example.

Example -

```
# Declaring a function
def add():
    # Defining local variables. They has scope only within a function
    a = 20
    b = 30
    c = a + b
```

```
    print("The sum is:", c)

# Calling a function
add()
```

Output:

```
The sum is: 50
```

Explanation:

In the above code, we declared a function named **add()** and assigned a few variables within the function. These variables will be referred to as the **local variables** which have scope only inside the function. If we try to use them outside the function, we get a following error.

```
1. add()
2. # Accessing local variable outside the function
3. print(a)
```

Output:

```
The sum is: 50
    print(a)
NameError: name 'a' is not defined
```

We tried to use local variable outside their scope; it threw the **NameError.**

Global Variables

Global variables can be used throughout the program, and its scope is in the entire program. We can use global variables inside or outside the function.

A variable declared outside the function is the global variable by default. Python provides the **global** keyword to use global variable inside the function. If we don't use the **global** keyword, the function treats it as a local variable. Let's understand the following example.

Example -

```
1. # Declare a variable and initialize it
2. x = 101
3.
4. # Global variable in function
```

```
5. def mainFunction():
6.     # printing a global variable
7.     global x
8.     print(x)
9.     # modifying a global variable
10.    x = 'Welcome To Python Programming'
11.    print(x)
12.
13.mainFunction()
14.print(x)
```

Output:

```
101
Welcome To Python Programming
Welcome To Python Programming
```

Explanation:

In the above code, we declare a global variable **x** and assign a value to it. Next, we defined a function and accessed the declared variable using the **global** keyword inside the function. Now we can modify its value. Then, we assigned a new string value to the variable x.

Now, we called the function and proceeded to print **x**. It printed the as newly assigned value of x.

Delete a variable

We can delete the variable using the **del** keyword. The syntax is given below.

Syntax -

```
1. del <variable_name>
```

In the following example, we create a variable x and assign value to it. We deleted variable x, and print it, we get the error **"variable x is not defined"**. The variable x will no longer use in future.

Example -

```
# Assigning a value to x
x = 6
```

```
print(x)
# deleting a variable.
del x
print(x)
```

Output:

```
6
Traceback (most recent call last):
  File "C:/Users/DEVANSH SHARMA/PycharmProjects/Hello/multiprocessing.py",
line 389, in
    print(x)
NameError: name 'x' is not defined
```

Maximum Possible Value of an Integer in Python

Unlike the other programming languages, Python doesn't have long int or float data types. It treats all integer values as an **int** data type. Here, the question arises. What is the maximum possible value can hold by the variable in Python? Consider the following example.

Example -

```
# A Python program to display that we can store
# large numbers in Python

a = 1000000000000000000000000000000000000000000000
a = a + 1
print(type(a))
print (a)
```

Output:

```
<class 'int'>
1000000000000000000000000000000000000000000001
```

As we can see in the above example, we assigned a large integer value to variable **x** and checked its type. It printed **class <int>** not long int. Hence, there is no limitation number by bits and we can expand to the limit of our memory.

Python doesn't have any special data type to store larger numbers.

Print Single and Multiple Variables in Python

We can print multiple variables within the single print statement. Below are the example of single and multiple printing values.

Example - 1 (Printing Single Variable)

```
# printing single value
a = 5
print(a)
print((a))
```

Output:

```
5
5
```

Example - 2 (Printing Multiple Variables)

```
a = 5
b = 6
# printing multiple variables
print(a,b)
# separate the variables by the comma
Print(1, 2, 3, 4, 5, 6, 7, 8)
```

Output:

```
5 6
1 2 3 4 5 6 7 8
```

Basic Fundamentals:

This section contains the fundamentals of Python, such as:

i)Tokens and their types.

ii) Comments

a)Tokens:

- The tokens can be defined as a punctuator mark, reserved words, and each word in a statement.
- The token is the smallest unit inside the given program.

There are following tokens in Python:

- Keywords.
- Identifiers.
- Literals.
- Operators.

We will discuss above the tokens in detail next tutorials.