Basic Python Syntax

By Md Saif Hossain

Instruction/Statement

A **statement is an instruction that a Python interpreter can execute**. So, in simple words, we can say anything written in Python is a statement.

Python statement ends with the token NEWLINE character. It means each line in a Python script is a statement.

```
# Statement 1
print('Hello')

# Statement 2
x = 20

# Statement 3
print(x)
```

```
# Statement 1
print('Hello', end=' ')

# Statement 2
x = 20

# Statement 3
print(x)
```

```
print("+" + 10 * "-" + "+")
print(("|" + " " * 10 + "|\n") * 3, end="")
print("+" + 10 * "-" + "+")
```

```
print("My", "name", "is", "Monty", "Python.", sep="-")
```

Basic Syntax (Comments)

- Comments can be used to explain Python code.
- Comments can be used to make the code more readable.
- Comments can be used to prevent execution when testing code.

```
# Single Line Comment

print('Bismillah_1')
# print('Bismillah_2')
```

```
# Multi Line Comment

'''
print('Bismillah_3')
print('Bismillah_4')
'''


"""
print('Bismillah_5')
print('Bismillah_6')
"""
print('Alhamdulillah')
```

Python Indentation

Right Code-

```
if 5 > 2:
        print("Five is greater than two!")
```

Wrong Code-

```
if 5 > 2:
print("Five is greater than two!")
```

Python Indentation

Right Code-

```
if 5 > 2:
        print("Five is greater than two!")
        if 3 > 2:
                print("Three is also greater than two!")
```

Wrong Code-

```
if 5 > 2:
        print("Five is greater than two!")
if 3 > 2:
print("Three is also greater than two!")
```

Python Variables

Variables: Variables are containers for storing data values.

Creating Variables: Python has no command for declaring a variable. A variable is created the moment you first assign a value to it.

```
x = 5
y = "John"
print(x)
print(y)
```

```
x = 4                    # x is of type 'int'
print(type(x))

x = "Alexander"  # x is now of type 'str'
print(type(x))
```

Python Variables

Casting: If you want to specify the data type of a variable, this can be done with casting. It is called user define data type.

```
x = str(3)     # x will be '3'
y = int(3)      # y will be 3
z = float(3)    # z will be 3.0

print(x, y, z, sep="__")
```

Get the Type: You can get the data type of a variable with the type() function.

```
x = 5
y = "John"
print(type(x))
print(type(y))
```

Python Variables

Single or Double Quotes?

String variables can be declared either by using single or double quotes:

```
# The two statement has same value
x = "John"
x = 'John'
```

Case-Sensitive: Variable names are case-sensitive.

```
# A will not overwrite a
a = 4
print(a)

A = "Sally"
print(A)
```

Python Variables

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.

```
# Assigning a single value to multiple variables
x=y=z=50
print(x)
print(y)
print(z)
```

```
# General way
x= 50
y= 50
z= 50
print(x, y, z)
```

```
# Another Approach
x, y = 10, 'Saif'

print("Number: ", x)
print("Name: ", y)
```

Python Variables

Delete a variable: We can delete the variable using the del keyword. The syntax is given below.

Syntax:

del <variable_name>

```
# Assigning a single value to multiple variables
x = 6
print(x)

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

```
# A Python program to display that we can store large numbers in Python
a = 1000000000000000000000000000000000000000000000
a = a + 1
print(type(a))
print (a)
```

Naming Variables-1

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 capitalized. For example - NameOfStudent, etc.

Snake Case- In the snake case, Words are separated by the underscore. For example - name_of_student, etc.

Naming Variables-2

When you name a variable, you need to adhere to some rules. If you don't, you'll get an error. The following are the variable rules that you should keep in mind:

- ❑ Variable names can contain only letters, numbers, and underscores (_). They can start with a letter or an underscore (_), not with a number.
- ❑ Variable names cannot contain spaces. To separate words in variables, you use underscores for example sorted_list.
- ❑ Variable names cannot be the same as keywords, reserved words, and built-in functions in Python

Naming Variables-3

The following guidelines help you define good variable names:

- Variable names should be concise and descriptive. For example, the active_user variable is more descriptive than the au.
- Use underscores (_) to separate multiple words in the variable names.
- Avoid using the letter l and the uppercase letter O because they look like the number 1 and 0.

Python Variable (Local)

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

Python Variable (Local)

```
# Declaring a function
def add():
    # Defining local variables.
    a = 20
    c = a + 5
    print("The sum is:" , c)

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

```
B = 15
#Declaring a function
def add():
    # Defining local variables.
    a = 20
    c = a + b
    print("The sum is:" , c)

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

Python Variable (Global)

Global variables can be used throughout the program, and their 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 the 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.

Python Variable (Global)

```
# Declare a variable and initialize it
x = 101

# Global variable in a function
def mainFunction():
    # printing a global variable
    global x
    print(x)
    # modifying a global variable
    x = 'Welcome To Javatpoint'
    print(x)
mainFunction()
print(x)
```

```
b = 15
#Declaring a function
def add():
    # Defining local variables.
    global a
    a = 20
    c = a + b
    print("The sum is:" , c)

add()
print(b)
print(a)
print("The sum is:" , c)
```

Single Valued Variable

Example - 1 (Printing Single Variable)

```
# printing single value
a = 5
print("Value of a is: ", a)
print("Value of a is: ", (a))
```

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)
```

Single Valued Variable

How to use string literals in Python?

```
# Printing String Variables
my_string = "This is Python"
this_char = "C"
multiline_str = """This is a multiline string with more
than one line code."""
unicode = u"\u00dcnic\u00f6de"
raw_str = r"raw \n string"

print(my_string)
print(this_char)
print(multiline_str)
print(unicode)
print(raw_str)
```

Output

```
This is Python
C
This is a multiline string with more than one line code.
Ünicöde
raw \n string
```

Single Valued Variable

Boolean literals: A Boolean literal can have any of the two values: True or False.

```
# How to use boolean literals in Python?
x = True
y = False
a = True + 4
b = False + 10

print("x is", x)
print("y is", y)
print( type( x ) )
print("a:", a)
print("b:", b)
```

```
x is True
y is False
<class 'bool'>
a: 5
b: 10
```

```
# How to use boolean as condition?
x = (1 == True)
y = (1 == False)

print("x is", x)
print("y is", y)
```

```
PS F:\myClass> python statement.py
x is True
y is False
```

Single Valued Variable

```
# How to use special literals in Python?
drink = "Available"
food = None

def menu(x):
    if x == drink:
        print(drink)
    else:
        print(food)

menu(drink)
menu(food)
```

```
Available
None
```

Multi-Valued Variable

```
# How to use literals collections in Python?
# List
fruits = ["apple", "mango", "orange"]
#set
vowels = {'a', 'e', 'i' , 'o', 'u'}
#tuple
numbers = (1, 2, 3)
#dictionary
alphabets = {'a' : 'apple', 'b' : 'ball', 'c' : 'cat'}

print("This is a list: ", fruits)
print("This is a set: ", vowels)
print("This is a tuple: ", numbers)
print("This is a dictionary: ", alphabets)
```

```
PS F:\myClass> python statement.py
This is a list:  ['apple', 'mango', 'orange']
This is a set:  {'i', 'o', 'u', 'a', 'e'}
This is a tuple:  (1, 2, 3)
This is a dictionary:  {'a': 'apple', 'b': 'ball', 'c': 'cat'}
```

Object Identity

In Python, every created object identifies uniquely in Python. Python provides the guarantee 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.

```
a = 50
b = a
print(id(a))
print(id(b))

# Reassigned variable a
a = 400
print(id(a))
```

```
a = 5
b = 3
result = a + b

print(result)
print(id(result))
print(id(b))
```

Receiving Input & showing Output

Input: Get user input in Python using the **input()** function. The user can enter keyboard input in the console.

Output: User can show/display the output using **print()** function in python.

```
name = input()

print("Your name is: ", name)
```

```
name = input("Enter your first name: ")

print("Nice to meet you", name)
```

```
print("Enter your first name: ")
name = input()

print("Nice to meet you", name)
```

```
print("Enter your first name: ", end=" ")
name = input()

print("Nice to meet you", name)
```

Input & Output

Getting the Data Type: You can get the data type of any object by using the type() function:

Example: Print the data type of the variable x:

```
# This is Example-1
num = input("Enter a Number: ")
print("Your Number is: ", num)
print(type(num))

# This is Example-2
y = 5
print("Value of y:", y)
print(type(y))
```

Type Conversion/Casting

Type Conversion: You can convert from one type to another with the int(), float(), str(), and complex() methods:

```
x = 1       # int
y = 2.8     # float
z = 2j      # complex

#convert from int to float:
a = float(x)

#convert from float to int:
b = int(y)

#convert from int to complex:
c = complex(x)
```

```
print(a)
print(b)
print(c)

print(type(a))
print(type(b))
print(type(c))
```

Type Conversion/Casting

Example Integers:

```
x = int(1)      # x will be 1
y = int(2.8)    # y will be 2
z = int("3")    # z will be 3
```

Example Floats:

```
x = float(1)      # x will be 1.0
y = float(2.8)    # y will be 2.8
z = float("3")    # z will be 3.0
w = float("4.2") # w will be 4.2
```

Example Strings:

```
x = str("s1")     # x will be 's1'
y = str(2)        # y will be '2'
z = str(3.0)      # z will be '3.0'
```

Type Conversion/Casting

Type Conversion

The process of converting the value of one data type (integer, string, float, etc.) to another data type is called type conversion. Python has two types of type conversion.

- **Implicit Type Conversion**

  In Implicit type conversion, Python automatically converts one data type to another data type. This process doesn't need any user involvement.

```
num_int = 123
num_flo = 1.23

num_new = num_int + num_flo

print("datatype of num_int:",type(num_int))
print("datatype of num_flo:",type(num_flo))

print("Value of num_new:",num_new)
print("datatype of num_new:",type(num_new))
```

Type Conversion/Casting

❑ **Explicit Type Conversion**

In Explicit Type Conversion, users convert the data type of an object to required data type. We use the predefined functions like int(), float(), str(), etc to perform explicit type conversion.

```
num_int = 123
num_str = "456"

print("Data type of num_int:",type(num_int))
print("Data type of num_str:",type(num_str))

print(num_int+num_str)
```

```
num_int = 123
num_str = "456"

print("Data type of num_int:",type(num_int))
print("Data type of num_str before Type
Casting:",type(num_str))

num_str = int(num_str)
print("Data type of num_str after Type Casting:",type(num_str))

num_sum = num_int + num_str

print("Sum of num_int and num_str:",num_sum)
print("Data type of the sum:",type(num_sum))
```

Type Conversion/Casting

❑ Three Way of Casting Type

```
m = input("Enter First Number: ")
n = input("Enter Second Number: ")
print(type(m))   #To show type of m
print(type(n))   #To show type of n

sum = m+n
print("Sum is: ",sum)
print(type(sum))
```

```
m = input("Enter First Number: ")
n = input("Enter Second Number: ")
print(type(m))   #To show type of m
print(type(n))   #To show type of n

sum = int(m)+int(n)
print("Sum is: ",sum)
print(type(sum))
```

```
m = input("Enter First Number: ")
n = input("Enter Second Number: ")
a = int(m)              #Casting Type
b = int(n)              #Casting Type
sum = a+b
print("Sum is: ", sum)
print(type(sum))
```

```
m = int(input("Enter First Number: "))
n = int(input("Enter Second Number: "))

sum = a+b
print("Sum is: ", sum)
print(type(sum))
```

Python Data Types

Built-in Data Types: In programming, the data type is an important concept. Variables can store data of different types, and different types can do different things. Python has the following data types built-in by default, in these categories:

Text Type	= str
Numeric Types	= int, float, complex
Sequence Types	= list, tuple, range
Mapping Type	= dict
Boolean Type	= bool
None Type	= NoneType
Set Types	= set, frozenset
Binary Types	= bytes, bytearray, memoryview

Text Data Types

Python Text Types: **'str'**

In Python, **string** data type is defined by any value with 'single' or "double" Quate.

Output:

```
Hello World
<class 'str'>
```

Example	Data Type	Code

x = "Hello World"	str	x = "Hello World"
		
		#display x:
		print(x)
		
		#display the data type of x:
		print(type(x))

Numeric Data Types

Python Numeric Types: **int, float, complex**

In Python, **integers** are zero, positive or negative whole numbers without a fractional part.

Output:

```
20
<class 'int'>
```

Example	Data Type	Code	Try this:

x = 20	int	x = 20
		#display x:
		print(x)
		#display the data type of x:
		print(type(x))
			num1 = 0b11011000 # binary
			print(num1) # 216 (Decimal)
			num2 = 0o12 # octal
			print(num2) # 10 (Decimal)
			num3 = 0x12 # hexadecimal
			print(num3) # 18 (Decimal)

Numeric Data Types

Python Numeric Types: **int, float, complex**

In Python, floating point numbers **(float)** are positive and negative real numbers with a fractional part denoted by the decimal symbol . or the scientific notation E or e

Output:

```
20.5
<class 'float'>
```

Example	Data Type	Code	Try this:

x = 20.5	float	x = 20.5
		#display x:
		print(x)
		#display the data type of x:
		print(type(x))
			num=100.25
			print(num)
			num=123_42.222_013
			print(num)
			num=1e3
			print(num)

Numeric Data Types

Python Numeric Types: **int, float, complex**

A **complex number** is a number with real and imaginary components. For example, 5 + 6j is a complex number where 5 is the real component and 6 multiplied by j is an imaginary component.
Complex data types is used while developing scientific applications where complex mathematical operation is required.

Output:

```
5j
<class 'complex'>
```

Example	Data Type	Code	Try This:

x = 5j	complex	x = 5j
		
		#display x:
		print(x)
		
		#display the data type of x:
		print(type(x))
			a=5+2j
			print(a)
			print(a.real)
			print(a.imag)
			print(type(a))
			
			b=5+5j
			print(a+b)

Sequence Data Types

Python Sequence Types: **list, tuple, range**

Setting the Data Type: In Python, the data type is set when you assign a value to a variable:

Output:
['apple', 'banana', 'cherry']
<class 'list'>

Example	Data Type	Code

x = ["apple", "banana", "cherry"]	list
	
	Data from the List can be changed
		x = ["apple", "banana", "cherry"]
		
		#display x:
		print(x)
		
		#display the data type of x:
		print(type(x))

Sequence Data Types

Python Sequence Types: **list, tuple, range**

Setting the Data Type: In Python, the data type is set when you assign a value to a variable:

Output:
('apple', 'banana', 'cherry‘)
<class ‘tuple'>

Example	Data Type	Code

x = ("apple", "banana", "cherry")	tuple
	
	Data from the tuple can be changed
		x = ("apple", "banana", "cherry")
		
		#display x:
		print(x)
		
		#display the data type of x:
		print(type(x))

Sequence Data Types

Python Sequence Types: **list, tuple, range**

Setting the Data Type: In Python, the data type is set when you assign a value to a variable:

Output:

```
range(0, 6)
<class 'range'>
```

Example	Data Type	Code

x = range(6)	range	x = range(6)
		
		#display x:
		print(x)
		
		#display the data type of x:
		print(type(x))

Mapping Data Types

Python Mapping Types: **dict**

Setting the Data Type: In Python, the data type is set when you assign a value to a variable:

```
Output:
{'name': 'John', 'age': 36}
<class 'dict'>
```

Example	Data Type	Code	Try this:

person = {
"name" : "John",
"age" : 36
}
	dict	person = {"name" : "John",
		"age" : 36}
		#display person:
		print(person)
		
		#display the data type
		print(type(person))
			book = { "chapter_1" : 12,
			"chapter_2" : 40,
			"chapter_3" : 73}
			
			#display person:
			print(book)

Boolean Data Types

Python Boolean Types: **bool**

The boolean value can be of two types only i.e. either True or False. The output <class 'bool'> indicates the variable is a boolean data type.

```
Output:
True
False
<class 'bool'>
```

Example	Data Type	Code	Try this:

x = True
y = False
	bool	x = True
		y = False
		
		#display x & y:
		print(x)
		print(y)
		
		#display the data type of x:
		print(type(x))
			a = True
			print(a)
			print(type(a))
			
			b = False
			print(b)
			print(type(b))

None Data Types

Python None Types: **NoneType**

Setting the Data Type: In Python, the data type is set when you assign a value to a variable:

```
Output:
None
<class 'NoneType'>
```

Example	Data Type	Code	Try this:

x = None	NoneType	x = None
		
		#display x:
		print(x)
		
		#display the data type of x:
		print(type(x))	

Set Data Types

Python Set Types: **set, frozenset**

Setting the Data Type: In Python, the data type is set when you assign a value to a variable:

Output:

```
{'cherry', 'apple', 'banana'}
<class 'set'>
```

Example	Data Type	Code

x = {"apple", "banana", "cherry"}	set	x = {"apple", "banana", "cherry"}
		#display x:
		print(x)
		#display the data type of x:
		print(type(x))

Set Data Types

Python Set Types: **set, frozenset**

Setting the Data Type: In Python, the data type is set when you assign a value to a variable:

Output:

```
frozenset({'banana', 'cherry', 'apple'})
<class 'frozenset'>
```

Example	Data Type	Code

x = frozenset({"apple", "banana", "cherry"})	Frozenset	x = frozenset({"apple", "banana", "cherry"})
		#display x:
		print(x)
		#display the data type of x:
		print(type(x))

Binary Data Types

Binary Types: **bytes, bytearray, memoryview**

Setting the Data Type: In Python, the data type is set when you assign a value to a variable:

Output:
b'Hello'
<class 'bytes'>

Example	Data Type	Code	Try this:

x = b"Hello"	bytes	x = b"Hello"
		#display x:
		print(x)
		#display the data type of x:
		print(type(x))	

Binary Data Types

Binary Types: **bytes, bytearray, memoryview**

Setting the Data Type: In Python, the data type is set when you assign a value to a variable:

```
Output:
bytearray(b'\x00\x00\x00\x00\x00')
<class 'bytearray'>
```

Example	Data Type	Code	Try this:

x = bytearray(5)	bytearray	x = bytearray(5)
		#display x:
		print(x)
		#display the data type of x:
		print(type(x))	

Binary Data Types

Binary Types: **bytes, bytearray, memoryview**

Setting the Data Type: In Python, the data type is set when you assign a value to a variable:

Output:

```
<memory at 0x000001B858F1F940>
<class 'memoryview'>
```

Example	Data Type	Code	Try this:

x = memoryview(bytes(5))	memoryview	x = memoryview(bytes(5))
		#display x:
		print(x)
		#display the data type of x:
		print(type(x))	

Python Booleans

Boolean Values: In programming, you often need to know if an expression is True or False. You can evaluate any expression in Python, and get one of two answers, True or False. When you compare two values, the expression is evaluated and Python returns the Boolean answer:

Example:

```
print(10 >9)
print(10==  9)
print(10<9)
```

Python Booleans

Example: Print a message based on whether the condition is True or False:

```
a = 200
b = 33

if b > a:
  print("b is greater than a")
else:
  print("b is not greater than a")
```

Formatted Float

```
x = 211911461911461819112146
y = 2**70
z = x / y

print(z)
print("%.2f" % z)
print("%.3f" % z)
print("{:.2f}".format(z))
print(f'{z:.2f}')
```

```
z = 2119.1461911461819112146

print("Value of z is:",z)
print("Value of z is: %.2f" %z)
print("Value of z is: %.3f" %z)
print("Value of z is: {:.2f}".format(z))
print(f'Value of z is: {z:.3f}')
```

Swapping

```
a = 10
b = 20

temp = a
a = b
b = temp

print(a, b)
```

```
a = 10
b = 20

a = a+b
b = a - b
a = a -b

print(a, b)
```

```
a = 10
b = 20

a, b = b, a

print(a, b)
```