Execute Python Syntax

Python syntax can be executed by writing directly in the Command Line:

Print ("Hello, World!")

Hello, World!

Or by creating a python file on the server, using the .py file extension, and running it in the Command Line:

C:\Users\Your Name>python myfile.py

Python Indentation

Indentation refers to the spaces at the beginning of a code line. Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important. Python uses indentation to indicate a block of code.

Example

```
if 5 > 2:
  print("Five is 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.

Example:

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

Variables do not need to be declared with any particular *type*, and can even change type after they have been set.

Example:

```
x = 4       # x is of type int
x = "Sally" # x is now of type str
print(x)
```

Casting

If you want to specify the data type of a variable, this can be done with casting.

Example:

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

Get the Type

You can get the data type of a variable with the `type()` function.

Example:

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

Single or Double Quotes?

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

Example:

```
x = "John"
# is the same as
x = 'John'
```

Case-Sensitive

Variable names are case-sensitive.

Example:

This will create two variables:

```
a = 4
A = "Sally"
#A will not overwrite a
```

Variable Names

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume). Rules for Python variables:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)

Example

Legal variable names:

```
myvar = "John"
my_var = "John"
_my_var = "John"
myVar = "John"
MYVAR = "John"
myvar2 = "John"
```

Example

Illegal variable names:

2myvar = "John"

my-var = "John"

my var = "John"

Multi Words Variable Names

Variable names with more than one word can be difficult to read. There are several techniques you can use to make them more readable:

Camel Case

Each word, except the first, starts with a capital letter:

myVariableName = "John"

Pascal Case

Each word starts with a capital letter:

MyVariableName = "John"

Snake Case

Each word is separated by an underscore character:

my_variable_name = "John"

Many Values to Multiple Variables

Python allows you to assign values to multiple variables in one line:

Example

```
x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)
```

One Value to Multiple Variables

And you can assign the *same* value to multiple variables in one line:

Example

```
x = y = z = "Orange"
print(x)
print(y)
print(z)
```

Unpack a Collection

If you have a collection of values in a list, tuple etc. Python allows you to extract the values into variables. This is called *unpacking*.

Example

Unpack a list:

```
fruits = ["apple", "banana", "cherry"]
x, y, z = fruits
print(x)
print(y)
print(z)
```

Output Variables

The Python `print()` function is often used to output variables.

Example

```
x = "Python is awesome"
print(x)
```

In the `print()` function, you output multiple variables, separated by a comma:

Example

```
x = "Python"
y = "is"
z = "awesome"
print(x, y, z)
```

You can also use the `+` operator to output multiple variables:

Example

```
x = "Python "
y = "is "
z = "awesome"
print(x + y + z)
```

Notice the space character after "Python" and "is ", without them the result would be "Pythonisawesome".

For numbers, the `+` character works as a mathematical operator:

Example

```
x = 5
y = 10
print(x + y)
```

In the `print ()` function, when you try to combine a string and a number with the `+` operator, Python will give you an error:

Example

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

The best way to output multiple variables in the `print()` function is to separate them with commas, which even support different data types:

Example

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

Global Variables

Variables that are created outside of a function (as in all of the examples above) are known as global variables. Global variables can be used by everyone, both inside of functions and outside.

Example

Create a variable outside of a function, and use it inside the function

```
x = "awesome"
def myfunc():
  print("Python is " + x)
myfunc()
```

If you create a variable with the same name inside a function, this variable will be local, and can only be used inside the function. The global variable with the same name will remain as it was, global and with the original value.

Example

Create a variable inside a function, with the same name as the global variable

```
x = "awesome"

def myfunc():
  x = "fantastic"
  print("Python is " + x)

myfunc()

print("Python is " + x)
```

The global Keyword

Normally, when you create a variable inside a function, that variable is local, and can only be used inside that function. To create a global variable inside a function, you can use the `global` keyword.

Example

If you use the `global` keyword, the variable belongs to the global scope:

```
def myfunc():
  global x
  x = "fantastic"

myfunc()

print("Python is " + x)
```

Built-in Data Types

In programming, 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
Set Types:	set, frozenset

Boolean Type:	bool
Binary Types:	bytes, bytearray, memoryview
None Type:	NoneType

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:

```
x = 5

print(type(x))
```

Setting the Data Type

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

```
x = "Hello World"

#display x:

print(x)

#display the data type of x:

print(type(x))
```

Example	**Data Type**
x = "Hello World"	str
x = 20	int
x = 20.5	float
x = 1j	complex
x = ["apple", "banana", "cherry"]	list
x = ("apple", "banana", "cherry")	tuple
x = range(6)	range
x = {"name" : "John", "age" : 36}	dict
x = {"apple", "banana", "cherry"}	set
x = frozenset({"apple", "banana", "cherry"})	frozenset
x = True	bool
x = b"Hello"	bytes
x = bytearray(5)	bytearray
x = memoryview(bytes(5))	memoryview
x = None	NoneType

Setting the Specific Data Type

If you want to specify the data type, you can use the following constructor functions:

Example	Data Type
x = str("Hello World")	str
x = int(20)	int
x = float(20.5)	float
x = complex(1j)	complex
x = list(("apple", "banana", "cherry"))	list
x = tuple(("apple", "banana", "cherry"))	tuple
x = range(6)	range
x = dict(name="John", age=36)	dict
x = set(("apple", "banana", "cherry"))	set
x = frozenset(("apple", "banana", "cherry"))	frozenset
x = bool(5)	bool
x = bytes(5)	bytes
x = bytearray(5)	bytearray
x = memoryview(bytes(5))	memoryview

Python Operators

Operators are used to perform operations on variables and values.

In the example below, we use the + operator to add together two values:

Example

```
Print (10 + 5)
```

Python divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators

Python Arithmetic Operators

Arithmetic operators are used with numeric values to perform common mathematical operations:

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

Python Assignment Operators

Assignment operators are used to assign values to variables:

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Python Comparison Operators

Comparison operators are used to compare two values:

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Python Logical Operators

Logical operators are used to combine conditional statements:

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

Python Identity Operators

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

Python Membership Operators

Membership operators are used to test if a sequence is presented in an object:

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

Python Bitwise Operators

Bitwise operators are used to compare (binary) numbers:

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1

^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

How to take input in Python?

Taking input is a way of interact with users, or get data to provide some result. Python provides two built-in methods to read the data from the keyboard. These methods are given below.

- input(prompt)
- raw_input(prompt)

input()

The input function is used in all latest version of the Python. It takes the input from the user and then evaluates the expression. The Python interpreter automatically identifies the whether a user input a string, a number, or a list. Let's understand the following example.

Example -

```
1. name = input("Enter your name: ")
2. print(name)
```

The Python interpreter will not execute further lines until the user enters the input.

Let's understand another example.

Example - 2

```
1. # Python program showing
2. # a use of input()
3. name = input("Enter your name: ")  # String Input
```

```
4. age = int(input("Enter your age: ")) # Integer Input
5. marks = float(input("Enter your marks: ")) # Float Input
6. print("The name is:", name)
7. print("The age is:", age)
8. print("The marks is:", marks)
```

Explanation:

By default, the **input()** function takes input as a string so if we need to enter the integer or float type input then the **input()** function must be type casted.

```
1. age = int(input("Enter your age: ")) # Integer Input
2. marks = float(input("Enter your marks: ")) # Float Input
```

We can see in the above code where we type casted the user input into **int** and **float**.

How input() function works?

- The flow of the program has stopped until the user enters the input.
- The text statement which also knows as prompt is optional to write in **input()** function. This prompt will display the message on the console.
- The **input()** function automatically converts the user input into string. We need to explicitly convert the input using the type casting.
- **raw_input() -** The raw_input function is used in Python's older version like Python 2.x. It takes the input from the keyboard and return as a string. The Python 2.x doesn't use much in the industry. Let's understand the following example.