Execute Python Syntax

As we learned in the previous page, 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 will give you an error if you skip the indentation:

### **Example**

Syntax Error:

if 5 > 2:

! print("Five is greater than two!")

## Python Variables

In Python, variables are created when you assign a value to it:

### **Example**

Variables in Python:

x = 5  
y = "Hello, World!"

## Comments

Python has commenting capability for the purpose of in-code documentation.

Comments start with a #, and Python will render the rest of the line as a comment:

### **Example**

Comments in Python:

#This is a comment.  
print("Hello, World!")

## Creating a Comment

**Comments starts with a #, and Python will ignore them:**

### **Example**

#This is a comment  
print("Hello, World!")

**Comments can be placed at the end of a line, and Python will ignore the rest of the line:**

### **Example**

print("Hello, World!") #This is a comment

**A comment does not have to be text that explains the code, it can also be used to prevent Python from executing code:**

### **Example**

#print("Hello, World!")  
print("Cheers, Mate!")

## Multiline Comments

Python does not really have a syntax for multiline comments.

To add a multiline comment you could insert a # for each line:

### **Example**

#This is a comment  
#written in  
#more than just one line  
print("Hello, World!")

**Or, not quite as intended**, you can use a multiline string.Since Python will ignore string literals that are not assigned to a variable, you can add a multiline string (triple quotes) in your code, and place your comment inside it:

### **Example**

"""  
This is a comment  
written in  
more than just one line  
"""  
print("Hello, World!")

As long as the string is not assigned to a variable, Python will read the code, but then ignore it, and you have made a multiline comment.

## 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)
* A variable name cannot be any of the **Python keywords**.

### **Example:**

Legal variable names:

myvar = "John"  
my\_var = "John"  
\_my\_var = "John"  
myVar = "John"  
MYVAR = "John"  
myvar2 = "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"

# **Assign Multiple Values**

## 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)

**Note:** Make sure the number of variables matches the number of values, or else you will get an error.

## 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**

## 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)

Also, use the global keyword if you want to change a global variable inside a function.

### **Example**

To change the value of a global variable inside a function, refer to the variable by using the global keyword:

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