Python Tutorial

Adopted from: https://www.w3schools.com/python/default.asp

Many PCs and Macs will have python already installed. To check if you have python installed on a Windows PC, search in the start bar for Python or run the following on the Command Line (cmd.exe):

in Windows: C:\Users\Your Name>python --version

In Linux :

```
$python --version
```

Let's write our first Python file, called helloworld.py, which can be done in any text editor.

```
print("Hello, World!")
```

Python is an interpreted programming language, this means that as a developer you write Python (.py) files in a text editor and then put those files into the python interpreter to be executed.

```
$python helloworld.py
```

Creating Variables

Variables are containers for storing data values.

Unlike other programming languages, Python has no command for declaring a variable.

A variable is created the moment you first assign a value to it.

```
x = 5      #This is a comment
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.

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

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

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

Rule for Naming Variable

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)

```
#Legal variable names:
myvar = "John"
my_var = "John"
_my_var = "John"
myVar = "John"
MYVAR = "John"
myvar2 = "John"
#Illegal variable names:
2myvar = "John"
my-var = "John"
my var = "John"
```

Assign Value to Multiple Variables

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

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

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

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

Output Variables

The Python `print` statement is often used to output variables.

To combine both text and a variable, Python uses the + character:

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

You can also use the + character to add a variable to another variable:

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

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

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

If you try to combine a string and a number, Python will give you an error:

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

Input Value from Keyboard

`'input()'` is used to take input from the user. We can also write something inside `input()` to make

```
print("Enter your name")
x = input()
```

Anything given to `input` is returned as a string. So, if we give an integer like 5, we will get a string i.e. **`'5'` (a string)** and **not 5 (int)**.

Now, let's learn to take integer input from the user.

```
x = input("Enter an integer >>>")
y = int(x)
print("You have entered",y)
```

Python Conditions and If statements

Python supports the usual logical conditions from mathematics:

- Equals: a == b   Not Equals: a != b
- Less than: a < b Greater than: a > b
- Less than or equal to: a <= b
- Greater than or equal to: a >= b

These conditions can be used in several ways, most commonly in "if statements" and loops.

An "if statement" is written by using the if keyword.

```
a , b = 33, 200 #you can assign two variable
if b > a:
   print("b is greater than a")
```

Indentation

Python relies on indentation (whitespace at the beginning of a line) to define scope in the code. Other programming languages often use curly-brackets for this purpose.

if statement, without indentation (will raise an error):

```
a = 33
b = 200
if b > a:
print("b is greater than a") # raise an error
```

Elif

The elif keyword is pythons way of saying "if the previous conditions were not true, then try this condition".

```
a , b = 33, 33
if b > a:
        print("b is greater than a")
elif a == b:
        print("a and b are equal")
```

Else

The else keyword catches anything which isn't caught by the preceding conditions.

```
a , b = 200, 33
if b > a:
        print("b is greater than a")
elif a == b:
        print("a and b are equal")
else:
        print("a is greater than b")
```

Short Hand If

If you have only one statement to execute, you can put it on the same line as the if statement.

One line if statement:

```
if a > b: print("a is greater than b")
```

Short Hand If ... Else

```
print("A") if a > b else print("=") if a == b else
print("B")
```

Boolean Operator

```
if a > b and c > a:
        print("Both conditions are True")
if a > b or a > c:
        print("At least one conditions is True")
```

Nested If

```
if x > 10:
        print("Above ten,")
        if x > 20:
                print("and also above 20!")
        else:
                 print("but not above 20.")
```

The pass Statement

`if` statements cannot be empty, but if you for some reason have an `if` statement with no content, put in the `pass` statement to avoid getting an error.

```
if b > a:
        pass
```

Python Loops

Python has two primitive loop commands:

- while loops and   for loops

While Loops

With the while loop we can execute a set of statements as long as a condition is true.

```
i = 1
while i < 6:
        print(i)
        i += 1
```

The break Statement

With the break statement we can stop the loop even if the while condition is true:

```
i = 1
while i < 6:
        print(i)
        if i == 3:
                break
        i += 1
```

The continue Statement

With the continue statement we can stop the current iteration, and continue with the next:

```
i = 0
while i < 6:
        i += 1
        if i == 3:
```

```
        continue
    print(i)
```

Python For Loops

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
     print(x)
```

Loop through the letters in the word "banana":

```
for x in "banana":
    print(x)
```

The break Statement

With the break statement we can stop the loop before it has looped through all the items:

Exit the loop when x is "banana":

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    print(x)
    if x == "banana":
        break
```

The continue Statement

With the continue statement we can stop the current iteration of the loop, and continue with the next:
Do not print banana:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    if x == "banana":
        continue
    print(x)
```

The range() Function

To loop through a set of code a specified number of times, we can use the range() function,

The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

```
for x in range(6):
    print(x)
```

Note that range(6) is not the values of 0 to 6, but the values 0 to 5.

The range() function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter: range(2, 6), which means values from 2 to 6 (but not including 6):

```
for x in range(2, 6):
    print(x)
```

The range() function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a third parameter: range(2, 30, **3**):

Increment the sequence with 3 (default is 1):

```
for x in range(2, 30, 3):
    print(x)
```

Nested Loops

A nested loop is a loop inside a loop.

The "inner loop" will be executed one time for each iteration of the "outer loop":

Print each adjective for every fruit:

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]
for x in adj:
    for y in fruits:
        print(x, y)
```