Python Tutorial

1. Data types

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

2. Numbers and Casting

Numbers:

Int : 1, 2, 3, 4, 5, …

Float : 1.5, 2.5, 3.5, 4.5, …

Complex : 1j, 1+1j, 4+7j, …

Example :

```
x = 1000
y = 13.5987
z = 4 + 7j

print (type(x))
print (type(y))
print (type(z))
```

Casting :

`Int ()` - constructs an integer number from an integer literal

`Float ()` - constructs a float number from an integer literal

`Str ()` - constructs a string from a wide variety of data types, including strings, integer literals and float literals

Example:

```
a = int (1) → a will be 1

b = int (15.8) → b will be 15

c = float (6) → b will be 6.0

d = float (12.3) → c will be 12.3

e = str (“3.9”) → d will be 3.9

f = str (“123ABC”) → e will be 123ABC

print(a)

print(b)

print(c)

print(d)

print(e)

print(f)
```

3. String

```
a = “ Xin chao, toi ten la Hua Ngoc Truc Tien ”

a = ‘ Xin chao, toi ten la Hua Ngoc Truc Tien ’

print (a)

print (b)
```

4. If ... elif ... else

Python supports the usual logical conditions from mathematics:

Equals: a == b

Not Equals: a != b

Less than: a < b

Less than or equal to: a <= b

Greater than: a > b

Greater than or equal to: a >= b

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

Example 1:

```
n = int(input("Enter n: "))

if (n > 0) :
    print(n, "+", 1, "=", n+1)
    print(n + 1)
```

Example 2:

```
a = 200
b = 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")
```

5. FOR

```
for <Variable Name> in <List, Tuple, Set etc>:
      <statement>
```

```
Example 1: Print each fruit in a fruit list

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

--------------------------------------------------------------------------------
Example 2: Loop through the letters in the word "banana"

for x in "banana":
  print(x)

--------------------------------------------------------------------------------
Example 3 : With the break statement we can stop the loop before it has
looped through all the items

fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
  if x == "banana":
    break
--------------------------------------------------------------------------------
Example 4 : Exit the loop when x is "banana", but this time the break
comes before the print

fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    break
  print(x)
--------------------------------------------------------------------------------
Example 5 : Do not print banana

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

6. **FOR - range()**

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

```
Range (start, stop,step_size)
```

```
for <Variable Name> in range (start, stop,step_size)
```

```
Example 1 :

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

## range(6) is not the values of 0 to 6, but the values 0 to 5 ##

--------------------------------------------------------------------------------
Example 2 :

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

--------------------------------------------------------------------------------
Example 3 :

for x in range(1, 30, 3):
    print(x)
```

7. Else - Break - Nested - Pass in For Loop

```
 Example 1 : The else keyword in a for loop specifies a block of code to be
executed when the loop is finished

 for x in range(6):
     print(x)
 else:
     print("Finally finished!")

--------------------------------------------------------------------------------
 Example 2 : If the loop breaks, the else block is not executed.

 for x in range(6):
     if x == 3: break
     print(x)
 else:
     print("Finally finished!")

--------------------------------------------------------------------------------
 Example 3: A nested loop is a loop inside a loop. The "inner loop" will be
executed one time for each iteration of the "outer loop"

 colors = ["red", "yellow", "black"]
 fruits = ["apple", "banana", "cherry"]

 for x in colors:
   for y in fruits:
     print(x, y)

--------------------------------------------------------------------------------
 Example 4 : for loops cannot be empty, but if you for some reason have a
for loop with no content, put in the pass statement to avoid getting an
error.

 for x in [0, 1, 2]:
      pass
 # having an empty for loop like this, would raise an error without the
pass statement
```

8. **While loops**

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

The while loops is also quite similar to the for loops

Note : With while loops you need to be careful. Because if you are careless in the programming process, it will lead to an infinite loop

```
while <condition> :
      <command block>
```

Equals: a == b

Not Equals: a != b

Less than: a < b

Less than or equal to: a <= b

Greater than: a > b

Greater than or equal to: a >= b

```
Example 1 : Print i as long as i is less than 6

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

--------------------------------------------------------------------------------
Example 2 : Sum of positive integers less than 8

n = 0
sum = 0
while n < 8:
    sum = sum + n
    n = n + 1
print("Sum of numbers less than 8 is : ", sum)
```

9. Else - break - continue - pass in while loops

```
Example 1 : With the else statement we can run a block of code once when
the condition no longer is true

 i = 1
 while i < 20:
   print(i)
   i += 1
 else:
   print("i is no longer less than 20")

--------------------------------------------------------------------------------
 Example 2 : 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

--------------------------------------------------------------------------------
 Example 3 : 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)

--------------------------------------------------------------------------------
 Example 4 : while loops cannot be empty, but if you for some reason have a
while loops with no content, put in the pass statement to avoid getting an
error.

 number = 0
 for number in range(10):
     if number == 5:
         pass    # pass here
     print('Number is ' + str(number))
 print('Out of loop')
```

10. Random

Method	Description
seed()	Initialize the random number generator
getstate()	Returns the current internal state of the random number generator
setstate()	Restores the internal state of the random number generator
getrandbits()	Returns a number representing the random bits
randrange()	Returns a random number between the given range
randint()	Returns a random number between the given range
choice()	Returns a random element from the given sequence
choices()	Returns a list with a random selection from the given sequence
shuffle()	Takes a sequence and returns the sequence in a random order
sample()	Returns a given sample of a sequence
random()	Returns a random float number between 0 and 1
uniform()	Returns a random float number between two given parameters
triangular()	Returns a random float number between two given parameters, you can also set a mode parameter to specify the midpoint between the two other parameters
betavariate()	Returns a random float number between 0 and 1 based on the Beta distribution (used in statistics)
expovariate()	Returns a random float number based on the Exponential distribution (used in statistics)
gammavariate()	Returns a random float number based on the Gamma distribution (used in statistics)
gauss()	Returns a random float number based on the Gaussian distribution (used in probability theories)
lognormvariate()	Returns a random float number based on a log-normal distribution (used in probability theories)
normalvariate()	Returns a random float number based on the normal distribution (used in probability theories)
vonmisesvariate()	Returns a random float number based on the von Mises distribution (used in directional statistics)
paretovariate()	Returns a random float number based on the Pareto distribution (used in probability theories)
weibullvariate()	Returns a random float number based on the Weibull distribution (used in statistics)

11. FUNCTION

- **Function (also known as Function)** : Is a block of instructions packaged into an independent unit, used to perform a task in the program.
- **Functions** provide better program division, and allow **code reuse** .
- Python provides many **built-in functions,** plus you can define your own functions. These functions are also known as **user-defined functions** .
    - The function after being defined will not execute itself.
    - The function executes only when called.

NOTE : *When defining a function, we should name the function a verb, because the function represents an action, a task of the program.*

** Some rules when defining functions in Python **

In Python, we define functions according to the following rule:

- The function definition will start with the keyword `def`, followed by **the function name** and parentheses `( )`
- The pair of signs `( )` will **contain the function's parameters** (if any).
- The first statement of a function can be an optional statement, to describe the function (also known as a docstring).
- The body of the function will start with an ampersand `:` and be indented.
- The command `return` is used to exit the function, and return the value from the function.

SYNTAX OF FUNCTION :

```
def Function name (parameter_1 [Parameters/Arguments], …, parameter_n):
      function-block
```

- **Functions** can **have parameters** .
- Parameters allow to change the content inside the function, making the function more flexible, more dynamic.
- Functions may **return different results** based on **different parameter values** .

NOTE : *When defining a function, we declare how many parameters, then when calling the function, we need to pass as many values into the function.*