Python Functions

is a block of related statements designed to perform a computational, logical, or evaluative task. The idea is to put some commonly or repeatedly done tasks together and make a function so that instead of writing the same code again and again for different inputs, we can do the function calls to reuse code contained in it over and over again.

Functions can be both built-in or user-defined. It helps the program to be concise, non-repetitive, and organized.

Built-in function

Python built-in functions are those functions whose functionality is pre-defined in Python

In [7]:
```
# Built-in function
x=-3
y=abs(x)
print(y)
type(y)
```

```
3
```

Out[7]:
```
int
```

User defined Functions

A function that you define yourself in a program is known as user defined function. You can give any name to a user defined function, however you cannot use the Python keywords as function name.

Creating a Function

In python, we define the user-defined function using def keyword, followed by the function name.

In [9]:
```
def fun():
    print('My First Function in Python')
```

Calling a Function

To call a function, use the function name followed by parenthesis:

In [11]:
```
def my_function():
  print("Hello from a function")

my_function()
```

```
Hello from a function
```

Python Function with arguments

Arguments are the values passed inside the parenthesis of the function.
A function can have any number of arguments separated by a comma.

In [14]:
```
def evenOdd(x):
    if (x % 2 == 0):
        print("even")
    else:
        print("odd")
evenOdd(4)
# Note
# A parameter is the variable listed inside the parentheses in the function definition.

# An argument is the value that is sent to the function when it is called.
```

```
even
```

In [16]:
```
# If your function expects 2 arguments, you have to call the function with 2 arguments, not
 more, and not less
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("Sikandar", "Hayat")
```

```
Sikandar Hayat
```

In [20]:
```
# A default argument is a parameter that assumes a default value if a value is not provided
 in the function call for that argument.
def myFun(x, y=50):
    print("x: ", x)
    print("y: ", y)
myFun(19)
```

```
x:  19
y:  50
```

Arbitrary Arguments, *args

If you do not know how many arguments that will be passed into your function, add a * before the parameter name in the function definition.
This way the function will receive a tuple of arguments, and can access the items accordingly:

In [48]:
```
def AI(*course):
  print(len(course))
  print("The first course in AI is " + course[0])


AI("ML", "DL", "CV")
```

```
3
The first course in AI is ML
```

In [54]:
```
def AI(*course):

    for i in course:
        print(i,end=', ')

AI("ML", "DL", "Cv",'NLP')
```

```
ML, DL, Cv, NLP,
```

In [55]:
```
# You can also send arguments with the key = value syntax.
# This way the order of the arguments does not matter.
```

In [60]:
```
def Artificial_intelligence (course1, course2, course3):
  print("The First course in AI is " + course1)

Artificial_intelligence(course1 = "ML", course2 = "Tobias", course3 = "Linus")
```

```
The First course in AI is ML
```

Return Values

To let a function return a value, use the return statement:

In [61]:
```
def my_function(x):
  return 5 * x

print(my_function(3))
```

```
15
```

In [74]:
```
# Factorial of a Number
def Factorial(x):
    num=1
    # check if the number is negative, positive or zero
    if x < 0:
        return "Sorry, factorial does not exist for negative numbers"
    if(x==0):
        return 1
    for i in range(1,x+1):
        num=i*num
    return num
Factorial(-1)
```

Out[74]:
```
'Sorry, factorial does not exist for negative numbers'
```

In [ ]:
```

```

In [ ]:
```

```