Functions in Python

Why Functions

- Code reusability
- Abstraction
- Testing becomes easy

Syntax

```
def funcName(parameters):
    # perform operations
    returnStatement;
```

A semicolon (;) in python is used to denote seperation and not termination

In [1]:
```python
def funcName():
    print("Welcome to Python functions");
funcName(); # Function call
```

```
Welcome to Python functions
```

Function returns sum of the two variables which are passed

In [2]:
```python
def summation(a, b):
    return a+b;

print(summation(2, 4));
```

```
6
```

Below function takes two string arguments and returns concatenation of those two

In [3]:
```python
def concat(x, y):
    return x+y;

print(concat("Hello", " Python"));
```

```
Hello Python
```

Below function takes two arguments and returns two values after performing some operation

In [4]:
```python
def returnTwo(a, b):
    a = a + 10;
    b = b + 20;
    return a, b;
print(returnTwo(10, 10))
```

```
(20, 30)
```

swap() function takes two arguments and swaps the content of the variables

In [5]:
```python
def swap(a, b):
    return(b, a);
x = 4;
y = 5;
print("Before swap - ",x, y);
x, y = swap(x, y);
print("After swap - ",x, y);
```

```
Before swap -  4 5
After swap -  5 4
```

Below is the function which returns a list

In [6]:
```
def returnArray():
    a = [1, 2, 6, 3, 8];
    return a;
print(returnArray())
```

```
[1, 2, 6, 3, 8]
```

Below function returns a boolean value

In [7]:
```
def returnBoolean(a, b):
    if(a>b):
        return True;
    else:
        return False;
print(returnBoolean(5, 3))
```

```
True
```

Keyword Arguments

Below function is called with parameter names mentioned explicitely

In [8]:
```
def printDetails(ID, name):
    print("ID - "+str(ID),"\nName - "+name);
    #or
    #print("ID - ",ID,"\nName - "+name);

printDetails(name="Naveen", ID=1);
```

```
ID - 1
Name - Naveen
```

Variable Length Arguments

In [9]:
```
def printDetails(ID, name, *varArg):
    print("ID - ", ID);
    print("Name - ", name);
    for arg in varArg:
        print(arg);
printDetails(1, "Naveen", "IIT Bombay", "M. Tech CSE")
```

```
ID -  1
Name -  Naveen
IIT Bombay
M. Tech CSE
```

Local and Global Variables

In [10]:
```
a = 10;
def func():
    a = 15; # local to func()
func()
print(a) # prints the global a value
```

```
10
```

In [11]:
```
a = 10;
def func():
    global a;
    a = 15; # refers to global a
func()
print(a) # prints the global a value
```

```
15
```

Anonymous Functions

Lambda Function

- Lambda Functions are anonymous as they don't have any name and they are not defined in the standard manner
- They take any number of arguments but return only one value
- Operation is performed on the arguments using the expression which is passed
- Lambda function takes only expression along with arguments. We are not allowed write any other statements like print

- They do not have access to other varaibles declared outside. They have only access to arguments which are passed
- Lambda functions helps us to perform basic operations without the need of writing functions explicitly

In [12]:
```
x = 30;
y= 40;
sum = lambda x, y:x+y
print(sum(x,y))
```

```
70
```

- Generally Lambda functions are used along with Map, Reduce and Filter

Map Function

- Map function takes lambda function along with a list on which the operation needs to be performed
- Map always returns a list

In [13]:
```
l = [1, 2, 3, 4, 5];
for i in range(len(l)):
    l[i] = l[i]**2;
print(l)
```

```
[1, 4, 9, 16, 25]
```

In [14]:
```
sqr = list(map(lambda x: x**2, [1, 2, 3, 4, 5])) # Returns
print(sqr)
```

```
[1, 4, 9, 16, 25]
```

In [15]:
```
incr = list(map(lambda x: x+1, [1,3,5,7,6]))
print(incr)
```

```
[2, 4, 6, 8, 7]
```

In [16]:
```
import math
sqrt_list = list(map(math.sqrt, [1, 4, 9, 25]))
print(sqrt_list)
```

```
[1.0, 2.0, 3.0, 5.0]
```

Reduce Function

- Reduce takes a list and a seed value and outputs a value
- Below Reduce Fucntion returns factorial of 6

In [17]:
```
fact = 1;
for i in range(1, 7):
    fact = fact*i;
print(fact);
```

```
720
```

In [18]:
```
import functools as f
fact = f.reduce(lambda x, y: x*y, [i for i in range(1, 7)], 1)
print(fact)
```

```
720
```

- seed value i.e., x = 1
- Step-1: y = 1 and x = 1. Therefore x*y = 1 which is returned
- Step-2: For this step seed becomes the output of the previous step which is 1. x = 1 and y = 2. Therefore x*y = 2 which is returned.
- Step-3: Seed value will be 2 and y = 3 and the steps continue..

Filter Function

- Filter takes a list and returns a filtered list
- Below Filter function returns list of integers which are divisible by 3

In [19]:
```
l = [i for i in range(1, 100)];
filtered_list=[];
for elem in l:
    if(elem%3 == 0):
        filtered_list.append(elem);
print(filtered_list)
```

```
[3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48, 51, 54, 57, 60, 63, 66, 69, 72, 75, 78, 81, 84, 87, 90, 9
3, 96, 99]
```

In [20]:
```
l = [i for i in range(1, 100)];
filtered_list = list(filter(lambda x: (x%3 == 0), l))
print(filtered_list)
```

```
[3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48, 51, 54, 57, 60, 63, 66, 69, 72, 75, 78, 81, 84, 87, 90, 9
3, 96, 99]
```