Functions

There are 2 types of functions

1]Non-Recursive Function

```
- This is further divided into 2 parts
    - a] Non-returning
    - b] Returning
```

1]Recursive Function

What is functions

A function is a block of code which only runs when it is called.

You can pass data, known as parameters, into a function.

A function can return data as a result.

Syntax

def function_name(arguments/parameter)

```
function body
```

Creating a Function

In Python a function is defined using the def keyword:

In [ ]:
```python
def greet():
    print('Good Morning')
    print('How can I help you')
```

Calling a Function

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

In [ ]:
```python
def greet():
    print('Good Morning')
    print('How can I help you')

#thie below line shows how to call any function
greet()
```

Function consist of 4 major components:

```
- def keyword
- function name
- argument/parameter
- function body
```

Arguments

```
Information can be passed into functions as arguments.
Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with
a comma.
lets see an example below
```

In [ ]:
```python
 def name(f_name):
        print(f_name+' is good boy')
name('Rohit')
name('You')
```

Parameters or Arguments?

```
The terms parameter and argument can be used for the same thing: information that are passed into a function.
```

From a function's perspective:

```
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.

Concept of Actual and formal
parameter=formal
argument=actual
```

Number of Arguments

```
By default, a function must be called with the correct number of arguments. Meaning that if your function expects 2 arguments, you have to
call the function with 2 arguments, not more, and not less.
```

In [ ]:
```python
def name(f_name,l_name):
    print(f_name)
    print(l_name)

name('Rohit','Madke')
#if we pass just 1 argument then it will throw an error
```

4 Type of Arguments

```
1]Position
2]Keyword
3]Default
4]Variable length
```

lets see them 1 by 1

Position

```
-Position is always important i.e. how we are passing the parameter
-Passing orders matter alot
-Understaning with an exaple - if we are passing name, first in actual parameter followed by age then accepting order in function's
parameter must be same as of passing
```

In [ ]:
```python
def raw_function(name,age):
    print(name)
    print(age)

raw_function('Rohit',20)
print()
#raw_function(20,"Rohit")
#this 2nd output is wrong. hence position matter alot in functions
```

Keyword Arguments

```
You can also send arguments with the key = value syntax.

This way the order of the arguments does not matter.
```

In [ ]:
```python
def raw_function(name,age):
    print(name)
    print(age)

raw_function(age='20',name="Rohit")
```

Default Parameter Value

```
The following example shows how to use a default parameter value.

If we call the function without argument, it uses the default value:
```

In [ ]:
```python
def raw_function(name,age='Unknown'):
    print(name)
    print(age)

raw_function(name="Rohit")

raw_function('Rohit',20)
```

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 [21]:
```python
def raw_function(*b):
    print(b)
    for i in b:
        print(i)

raw_function(2,3,4,5,6)
```

```
(2, 3, 4, 5, 6)
2
3
4
5
6
```

Arbitrary Keyword Arguments, **kwargs

```
If you do not know how many keyword arguments that will be passed into your function, add two asterisk: ** before the parameter name in the
function definition.

This way the function will receive a dictionary of arguments, and can access the items accordingly:
```

In [23]:
```python
def raw_function(name,**data):
    print(name)
    for i,j in data.items():
        print(i,j)

raw_function('Rohit',age='18',Hobby='Cricket,coding',Phone='Jio')
```

```
Rohit
age 18
Hobby Cricket,coding
Phone Jio
```

In [24]:
```python
#Till now were just understanig the functions with differnt types of parameter
#now its time to see returning functions
```

In [25]:
```python
def add(x,y):
    z=x+y
    #this return statement is important
    return z

z=add(10,20)
print(z)
```

```
30
```

In [27]:
```python
def add_sub_mul(x,y):
    a=x+y
    b=x-y
    c=x*y
    return a,b,c
a,b,c=add_sub_mul(3,2)
print(a)
print(b)
print(c)
```

```
5
1
6
```

Local Variable,Global Variable and scope

Local Variable-

```
A variable declared inside the function's body or in the local scope is known as a local variable.
```

Scope-

```
A variable is only available from inside the region it is created. This is called scope.
Scope refers to the visibility of variables. In other words, which parts of your program can see or use it. Normally, every variable has a
global scope. Once defined, every part of your program can access a variable. It is very useful to be able to limit a variable's scope to a
single function
```

Global Variable-

```
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.
```

In [30]:
```python
a=10

def scope():
    #local a
    a=15
    print('value of a in local variable ',a)

scope()
print('value of a in global variable ',a)
```

```
value of a in local variable  15
value of a in global variable  10
```

globals()['variable_name']=value -

```
when programmer wants to change the value of global varibale inside a function without disturbing local variable then    globals is used
```

In [32]:
```python
s=20

def funt():
    s=9
    print('value of local s inside function ',s)
    globals()['s']=10
funt()
print('value of global s ourisde function ',s)
```

```
value of local s inside function  9
value of global s ourisde function  10
```

In [37]:
```python
#### Passsing of list to function
l=list(range(10))
def list_fun(l1):
    for i in l1:
        print(i,end=' ')
list_fun(l)
```

```
0 1 2 3 4 5 6 7 8 9
```

More on functions

```
Anonymous Function(Lambda function)


Functions without name are known asanonymous function or we can call them as lambda function
Lambda Functions are another way of defining functions to execute small functionality occuring while implementing complex
functionality.These function can take multiple parameter as input but can only execute single expression.In other words they can only
perform single operation
```

In [40]:
```python
#whenever we have just 1 line of code we use lambda function
#normal way of doing it

def square(a):
    return a*a
f=square(5)
print(f)

#using lambda
f=lambda a:a*a
print(f(5))
```

```
25
25
```

In [42]:
```python
#more example
def add(x,y):
    return x+y
f=add(5,4)
print(f)

#using lambda
a= lambda x,y:x+y
print(a(5,4))
```

```
9
9
```

In [44]:
```python
#sample example
f=lambda x:'even' if x%2==0 else 'odd'
print(f(2))
```

```
even
```

map()

```
syntax of map
map(function,iterable_object)
function can be anonymous or user defined while iterable_object can be list,tuple,set,dict,string
```

map():

```
        we can understand this using following examples
```

In [45]:
```python
l=['india','paki','german']
f=map(lambda x:x.upper(),l)
list(f)
```

Out[45]:
```
['INDIA', 'PAKI', 'GERMAN']
```

In [48]:
```python
l=list(range(1,7))
f=map(lambda x:x*x,l)
list(f)
```

Out[48]:
```
[1, 4, 9, 16, 25, 36]
```

Implementation of Map():

```
                         During execution of the map() function ,the lambda function x:x*2 would return a lambda object , but the map
handles the job of passing each element of the iterable to the lambda object and storing the value obtained in a map object. And to print
the values stored in map we use list function
```

filter():

```
           Filter is similar to map function,only distinguishing feature being that it requires the function to look for a condition and
then returns only those elements from the collection that satisfy the condition.

           syntax:
           filter(function,iterable_object)
```

In [50]:
```python
#e.g.
my_list=[3,4,5,6,7,8,9]
div_by_3=lambda x:x%3==0
div=filter(div_by_3,my_list)
list(div)
```

Out[50]:
```
[3, 6, 9]
```

```
some condtions of filter
1] the function object passed to the filter should always return boolean values
2] the iterable object can be string,list,tuple,set,dict
```

In [52]:
```python
d={1:['rohit',21],2:['abhi',24],3:['raj',22]}
f=filter(lambda x:x[1]>22,d.values())
list(f)
```

Out[52]:
```
[['abhi', 24]]
```

Reduce():

```
            reduce is an operation that break down the entire process into pair-wise operation and uses the result from rach operation
,with thes successive element .
            syntax:
            reduce(function,iterable_object)
```

- feature of reduce

  1]the iterable object can be string,lidt,tuple,set or dict

  2]also reduce function produces single output

In [62]:
```python
from functools import reduce
l=list(range(1,7))
print(l)
f=reduce(lambda x,y:x*y,l)
print(f)
```

```
[1, 2, 3, 4, 5, 6]
720
```

In [63]:
```python
l=list(range(11))
even=filter(lambda x:x%2==0,l)
doubles=map(lambda x:x*2,even)
sumx=reduce(lambda x,y:x+y,doubles)
print(sumx)
```

```
60
```