Functions :-

built- in-function

Functions take some i/p

Function is piece of code that can reuse

```
def function (a):
    b= a+1;
print (a," +1 =" b)
    return b
```

The function sum takes a iterable tuple or list and return the total of all the elements

```
a_r = [10, 8.5, 9.5, 7, 7,
       9.5, 9.0, 9.5]
```

* Python Built-in functions

LEN → takes in a i/p of type sequence such as string, or list

Returns legth of that seque

```
L = len (album_ratings)
L : 8
  ⇓
 for
album_ratings = [        ]
```

1	2	. . . . .	8

```
s= sum (album_ratings)
s : 70
```

① There are two ways to sort a list

i) function sorted

method can also be used

```
sorted_album_rating = sorted (album_ratings)
album_rating = [ . . . ]
sorted_album_rating = [ . . . ]
```

album_rating :- doesn't change

For sort :-

album_ratings = [10, 8.5, 9.5, 7]

album_ratings.sort()

album_ratings = [7, 7, 8.5, 9, 9.5]

The album_rating has changed.

no new list is created

* How to build your own function

Ex. def add1(a): → formal parameters in parentheses

b = a+1

return b

In this eg., python returns its i/p value plus one

→ After we define the function, we can call it add1(5)

6

c = add1(10)

c : 11

* Documentation is surrounded in triple quotes.

help(add1) → will display documentation

Multiple parameters :-

→ A function can have multiple parameters

def mult (a,b):

c = a*b

return c

mult (2,3)

c

mult (2, "MJ")

⇒ "MJ MJ" ⟶ the string is repeated two times

this is because the multiplication symbol can also mean repeat a sequence

* Function doesn't have a return statement

```
def MJ():
    print ('Michael Jackson')
```

⟶ In this case, Python will return the special none object

(2)
```
def NoWork():
    pass
print(NoWork())
```
None

If the return statement is not called

Loops in Functions :-

```
def printstuff (Stuff):
    for i, s in enumerate (Stuff):
        print ("Album", i, "Rating is", s)
```

⟶ Function prints out the values and indexes of a loop or tuple

eg. album-ratings = [10.0, 8.5, 9.5]

printstuff (album-ratings)

Stuff is used as an i/p to the function enumerate

This operation will pass the index to i and value in s

stuff : [10, 8.5, 9.5]

index : [0 | 1 | 2]

o/p :- Album 0 Rating is 10
— 11 — 1 — 11 — 8.5
— 11 — 2 Rating is 9.5

Collecting arguments

```
def ArtistNames (*names):
    for name in names:
        print (name)
```

* Variadic parameters allow us to i/p a variable no. of elements

→ The function has an asterisk on the parameter names

→ When we call function three parameters are packed into the tuple names

ArtistNames ("MJ", "AC/DC", "Pink Floyd")

Scope :- The scope of a variable is the part of the program where that variable is accessible

Variables are defined outside of any function are said to be global scope meaning they can be accessed anywhere after they are defined.

Global scope

```
def AddDC(x):
    x = x + "DC"
    print(x)
    return(x)

x = "AC"
z = AddDC(x)
```

x
"AC"

x is accessible anywhere after it is defined

A variable defined in the global scope is called a global variable.

```
Scope AddDC
    x = "AC"
```

x
"ACDC"

Local variables

global scope

```
def Thriller():
    Date = 1982
    return Date

Thriller()
Date
```

Local variables only exist within the scope of a function

```
Global Scope
def Thriller():
    Date = 1982
    return(Date)

Date = 2017
print(Thriller())
1982
print(Date)
2017
```

Objects and Classes

~~Easy ERA~~

Built-in Types in Python

- Python has lots of data types
- Types :
  - → Int : 1, 2, 567 ...
  - → Float : 1.2, 0.62,
  - → String : 'a', 'abc',
  - → List : [1, 2, 'abc']
  - → Dictionary : { "dog" : 1 , "cat" : 2 }
  - → Bool : False, True

* Each is an object

- every object has :
  - → A type
  - → An internal data representation (a blueprint)
  - → A set of procedures for interacting with the object (methods)

→ An object is an instance of a particular type

type

type 1 type 2

→ You can find the type of object by using the command type()

>> type ([1, 34, 3])
<class 'list'>

>> type (1)
<class 'int'>