Functions :-

A function is a block of code that performs a specific task whenever it is called. In bigger programs, where we have large amounts of code, it is advisable to create or use existing functions that make the program flow organized and neat.

There are two types of functions:

1) Built in functions.
2) User-defined functions.

Built in function:

These functions are defined and precode in python. Some example of built in function are as follow:

min(), max(), len(), sum(), type(), range(), dict(), list(), tuple(), set(), print(), ch.

User defined function:

we can create function to perform specific tasks as per our needs: such functions are called user-defined functions.

without function

```
a=9
b=8
gmean1 = (a*b)/(a+b)
print(gmean1)
c=8
d=7
gmean2 = (c*d)/(c+d)
print(gmean2)
```

with function

```
def Calculate_Gmean(a, b):
    mean = (a*b)/(a+b)
    print(mean)
a=8
Calculate_Gmean(a, b)
c=8
d=7
Calculate_Gmean(c, d)
```

Pass :) If we want to skip after some time we use pass function.

User defined function

Syntax:-

```
def function_name(parameters):
    pass
```

- Creat a function using the def keyword, followed by a function name, followed by a parentheses (()) and a colon (:).
- Any parameters and argument should be placed within the parenthesis
- Rules to naming function are similar to that of naming variable.
- Any statement and other code within the function should be indented

Calling a function

We call a function by giving the function name, followed by parameters (if any) in the parenthesis

Example

```
def name(fname, lname):
        print("Hello,", fname, lname)
name("som", "wilson")
```

Function Arguments and return statement

There are four types of arguments that we can provide in a function:

- **Default Argument**:- we can provide a default value while creating a function. This way the function assumes a default value even if a value is not provided in the function call for that argument:

Example:

```
def name(fname, mname = "Kumar", lname = "Kumar"):
    print("Hello", fname, mname, lname)

name("Dinesh")


def average(a=9, b=1):
    print("The average is", (a+b)/2)

average(1,5)        # It will ignore the value of a=9, and b=1 and will take, 1 and 5
```

Keyword Arguments:

We can provide arguments with key = value this way the interpreter recognize the arguments by the parameter name, Hence, the order in which the arguments are passed does not matter.

Example

```
def name(fname, mname, lname):
    print("Hello", fname, mname, lname)

name(mname = "kumar", lname = "Pandka", fname = "Dinesh")


def average(a=9, b=1):
    print("The average is", (a+b)/2)

average(b=9, a=21)   # We can write b at the place of a it does not matter in keyword arguments to write a argument in line wise
```

Required Arguments:-

In case we don't pass the arguments with a key value syntax, then it is necessary to pass my arguments in the correct positional order and the number of arguments is passed should match with actual definition.

Ex: when number of arguments passed does not match to the actual function definition.

```
def name(fname, mname, lname):
    print("Hello", fname, mname, lname)
name("Peter", "Duell")

def avg(a, b, c=1)          # we have to value
    print("The avg is", (a+b+c)/2)
avg(5,6)                    we have to pass the value of a, b
```

Variable length Arguments

Sometimes we may need to pass more arguments than most defined in the actual function. This can be done using variable length argument.

Arbitrary ~~key~~ Arguments.

```
def avg(*numbers):                    single * = tuple.
    sum=0
    for i in numbers:
        sum=sum+i
    print("avg is :", sum/len(numbers))
avg(1,5)
```

Keyword Arbitrary Arguments.

While creating a function I pass a ** keyword ~~argument~~, the parameter name while defining the function. The function access the arguments by processing them as the form of dictionary

double * * Dictionary

```
def name(**name):
    print("Hello,", name["fname"],
name["mname"], name["lname"])

name(mname = "Buchanan", lname = "Barnes" fname = "James")
```

Return Statement:

The return statement is used to return the value of an expression back to the calling function.

```
def average(*numbers):
    sum = 0
    for i in numbers:
        sum = sum + i
    return sum/len(numbers)

c = average(5, 6, 7, 1)
print(c)
```