Functions

1. A function is a block of code which only runs when it is called.
2. You can pass data, known as parameters, into a function.
3. A function can return data as a result.
4. Creation of function we use def keyword def my_function():
5. To call a function, use the function name followed by parenthesis
   my_function()
6. Information can be passed into functions as arguments.
7. Arguments are specified after the function name, inside the parentheses. We can write as many arguments as we want, just separate them with a comma.

```
def my_fun (name, age):
print(“name , age”)
my_fun(“ABC” , 15)
my_fun(“XYZ” , 23)
my_fun(“LMN” , 68)
```

8. 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.

9. when we do not know how many arguments will be passed into the functionthen we add a * before the parameter name in the function definition.

```
def my_function(*kids):
```

Arbitrary Keyword Arguments

```
def my_function(**kid):
    print("His last name is " + kid["lname"])
my_function(fname = "Ram", lname = "Pendase")
```

Keyword Arguments

```
def my_function(child3, child2, child1):
    print("The youngest child is " + child3)
my_function(child1 = "Ram", child2 = "Laxman", child3 = "Bharat")
```

Default Parameter Value

```
def my_function(country = "India"):
    print("I am from " + country)
my_function("Sweden")
my_function("Norway")
my_function()
my_function("Brazil")
```

Pa

You function (string, number, list, dictionary etc.),
and be inside the function.

```
def my_function(food):
    for x in food:
        print(x)
fruits = ["apple", "banana", "cherry"]
my_function(fruits)
```

Return Values

```
def my_function(x):
    return 5 * x
print(my_function(3))
print(my_function(5))
print(my_function(9))
```

Recursive Functions

Python also accepts function recursion, which means a defined function can call itself.

Advantages of Recursion

1. Recursive functions make the code look clean and elegant.
2. A complex task can be broken down into simpler sub-problems using recursion.
3. Sequence generation is easier with recursion than using some nested iteration.

Disadvantages of Recursion

1. Sometimes the logic behind recursion is hard to follow through.
2. Recursive calls are expensive (inefficient) as they take up a lot of memory and time.
3. Recursive functions are hard to debug.

```
# Program to print the fibonacci series upto n_terms
# Recursive function
def recursive_fibonacci(n):
    if n <= 1:
      return n
    else:
      return(recursive_fibonacci(n-1) + recursive_fibonacci(n-2))
n_terms = 10
# check if the number of terms is valid
if n_terms <= 0:
    print("Invalid input ! Please input a positive value")
else:
    print("Fibonacci series:")
for i in range(n_terms):
     print(recursive_fibonacci(i))
# print factorial of a number
def factorial(x):
    """This is a recursive function
    to find the factorial of an integer"""
    if x == 1:
        return 1
    else:
        return (x * factorial(x-1))
```

```
num = 3
print("The factorial of", num, "is", factorial(num))
```

Lambda Functions

1. A lambda function is a small anonymous function.
2. A lambda function can take any number of arguments, but can only have one expression.

lambda *arguments* : *expression*

3. The power of lambda is better shown when you use them as an anonymous function inside another function.
4. Use lambda functions when an anonymous function is required for a short period of time.

```
x = lambda a : a + 10
print(x(5))
```

```
x = lambda a, b : a * b
print(x(5, 6))
```

```
x = lambda a, b, c : a + b + c
print(x(5, 6, 2))
```

Modules

1. a module to be the same as a code library.
2. A file containing a set of functions you want to include in your application.
3. To create a module just save the code you want in a file with the file extension .py
4. we can rename the module by using keyword as

mymodule.py

```
def greeting(name):
  print("Good Morning, " + name)
```

```
import mymodule

mymodule.greeting("Ram")
```

```
import mymodule as mx

mx.greeting("Ram")
```

Python Function to Display Calendar

```
# First import the calendar module
import calendar
# ask of month and year
yy = int(input("Enter year: "))
mm = int(input("Enter month: "))
# display the calendar
print(calendar.month(yy,mm))
```

Q] write a Python Program to Make a Simple Calculator using function

Sample Output is given

```
Please select the operation.
a. Add
b. Subtract
c. Multiply
d. Divide
Please enter choice (a/ b/ c/ d): b
Please enter the first number: 12
Please enter the second number: 11
12 - 11 = 1
```

python program to find Find LCM

```
# defining a function to calculate LCM
def calculate_lcm(x, y):
   # selecting the greater number
   if (x > y):
      greater = x
   else:
      greater = y
   while(True):
      if((greater % x == 0) and (greater % y == 0)):
         lcm = greater
         break
      greater += 1
   return lcm
 # taking input from users
num1 = int(input("Enter first number: "))
num2 = int(input("Enter second number: "))
# printing the result for the users
print("The L.C.M. of", num1,"and", num2,"is", calculate_lcm(num1, num2))
```

PYTHON PROGRAM USING MODULE FOR CALCULATOR

```
# A simple module, calc.py
def add(x, y):
  return (x+y)
def subtract(x, y):
  return (x-y)
```

```
# importing module calc.py
import calc
print(calc.add(10, 2))
```