**Program 1 :**

Write a Python function to check whether a number is perfect or not.

In Python, a function is a group of related statements that performs a specific task.

Functions help break our program into smaller and modular chunks. As our program grows larger and larger, functions make it more organized and manageable.

Furthermore, it avoids repetition and makes the code reusable.

* **Syntax of Function**

def function\_name(parameters):

"""docstring""" # not compulsory

statement(s)

return data

* **Call the function**

Once we have defined a function, we can call it from another function, program, or even the Python prompt. To call a function we simply type the function name with appropriate parameters.

* **Types of Functions**

1. **Built-in functions -** Functions that are built into Python.
2. **User-defined functions** **-** Functions defined by the users themselves.

**Code :**

def isPerfect(n) :

    sum = 0

    for i in range(1,n) :

        if(n % i == 0) :

            sum += i

    if(sum == n) : return True

    return False

n = int(input("Enter the number to be checked : "))

if(isPerfect(n)) : print("Perfect number")

else : print("Not a perfect number")

**Program 2 :**

Write a Python function to check whether a string is a pangram or not.

**Function used**:

**1. not in:** ‘not in’ operator- Evaluates to true if it does not finds a variable in the specifed sequence and false otherwise.

**Code :**

def isPangram(s) :

    s=s.lower()

    alphabetSet = "abcdefghijklmnopqrstuvwxyz"

    for i in alphabetSet :

        if(i not in s) : return False

    return True

s = str(input("Enter the string to be checked :"))

if(isPangram(s)) : print('"'+s+'" is pangram')

else : print('"'+s+'" is not a pangram')

**Program 3 :**

Python menu driven program to develop simple calculator using variable

length argument.

**Variable length arguments in Python**

A variable length argument as the name suggests is an argument that can *accept variable number of values*. To indicate that the function can take variable number of argument .

**Syntax :**

write a variable argument using a ‘\*’, for example func(\*args).

Variable arguments help in the scenario where the exact number of arguments are not known in the beginning, it also helps to make your function more flexible.

**Code :**

def add(\*nums) :

    sum = 0

    for i in range(len(nums)):

        sum += int(nums[i])

    return sum

def sub(\*nums) :

    diff = nums[0]

    for i in range(1,len(nums)):

        diff -= int(nums[i])

    return diff

def multiply(\*nums) :

    multi = 1

    for i in range(len(nums)):

        multi \*= int(nums[i])

    return multi

def div(\*nums) :

    div = nums[0]

    for i in range(1,len(nums)):

        div /= int(nums[i])

    return div

print("1. Add \t2. Sub \t3. Multiply  4. Divide")

choice = int(input("Enter your choice : "))

nums = str(input("Enter the numbers : "))

parm = list(map(int,nums.split()))

if(choice == 1) :

    print("Result of addition : " +str(add(\*parm)))

elif(choice == 2) :

    print("Result of subtraction : " +str(sub(\*parm)))

elif(choice == 3) :

    print("Result of multiplication : " +str(multiply(\*parm)))

elif(choice == 4) :

     print("Result of division : " +str(div(\*parm)))

**Program 4 :**

Program to calculate factorial of a number using recursion.

**Recursion :**

Recursion is a common mathematical and programming concept. It means that a function calls itself. This has the benefit of meaning that you can loop through data to reach a result.

**Code :**

def fact(n) :

    if(n==1) : return 1

    return (n\*fact(n-1))

n = int(input("Enter a number : "))

print("Factorial = "+ str(fact(n)))

**Program 5 :**

Python program to calculate square and cube of a number and use two decorators one to increase result by 4 and another to multiply result by 2.

**Decorators in Python :**

Decorators are like gift wrappers.

If you want to extend the behavior of a function but don’t want to modify it permanently, you can wrap a decorator on it.

A decorator is a function that takes another function as its argument, and returns yet another function. Decorators can be extremely useful as they allow the extension of an existing function, without any modification to the original function source code.

**Creating Decorator** **:**

Decorators are nested functions. The outer function takes the function to decorate as an argument and then the inner function calls it.

**Code :**

def square(n) : return n\*n

def cube(n) : return n\*n\*n

def incBy4(func,n) :

    return func(n) + 4

def multiBy2(func,n) :

    return func(n) \* 4

print(incBy4(square, 3))

print(multiBy2(square, 3))

print(incBy4(cube, 3))

print(multiBy2(cube, 3))

**Output :**

