INTRODUCTION TO PYTHON

❖ **Python :** Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language that can be used effectively to build almost different types of applications. It was created by **Guido van Rossum, and released in 1990**.

- **Features of Python :**

  - **Python is Interpreted** − Python is processed at runtime by the interpreter. We do not need to compile the program before executing it. This is similar to PERL and PHP.
  - **Python is Object-Oriented** − Python supports Object-Oriented style or technique of programming.
  - **Python is a Beginner's Language** − Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to browsers.
  - **Easy-to-learn** − Python has few keywords, simple structure, and a clearly defined syntax. This allows the user to pick up the language quickly.
  - **Easy-to-read** − Python code is more clearly defined and visible to the eyes.
  - **Easy-to-maintain** − Python's source code is fairly easy-to-maintain.
  - **A broad standard library** − Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
  - Cross-platform Language/ Portable - Python **works on different platforms** (Windows, Mac, Linux, Raspberry Pi, etc).
  - **Databases** − Python provides interfaces to all major commercial databases.
  - **GUI Programming Support** - Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

❖ **History of Python**

- Python was developed by Guido van Rossum in 1990 at the National Research Institute for Mathematics and Computer Science in the Netherlands.It is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, Unix shell and other scripting languages. It gained popularity with the arrival of Python 2.0 in 2000. In incorporated a number of important improvements to the language and added libraries. Python 3.0 was released at the end of 2008. Python is copyrighted. Python source code is now available under the GNU General Public License (GPL). Latest version is 3.10

❖ The Basic Elements of Python

- **Python Program -** It is also called a **script**, is a sequence of definitions and commands.
- **Shell –** Shell is a command interpreter. The definitions are evaluated and the commands are executed by the **shell**. A new shell is created whenever execution of a program begins.
- **Command** – It is also called a **statement**. It instructs the interpreter to do different tasks.

❖ Comments in Python

- Comments are non executable statements. They are used to explain any program code and improve the readability of a program. Python supports two types of comments:
- **Single Line Comment**: Any statement that begins with a hash symbol (#) is treated as a single line comment.
- Eg:

```
# This is single line comment.
print "Hello Python"
```

- **Multi Line Comment:** Any statement that starts and ends with triple quotes is treated as a multiline comment
- Eg:

```
''' This
Is
Multiline comment'''
```

❖ Variables:

- Quantities that can change during the execution of a program are known as variables. They are containers for storing data values. In Python, the variables need not be declared for a particular datatype. They arecreated when a value is assigned to it.
- Example

```
x= 5                # x is int
y= “Belgaum”        #y is str
print (x)
print(y)
```

- Variable values can even change type after they have been set
- Example

```
x= 5                # x is int
x= “Belgaum”  #x is str
print (x)
```

- We can specify the data type of a variable, with casting
- Example

```
x = str(3)    # x will be '3‘
y = int(3)    # y will be 3
```

```
z = float(3)  # z will be 3.0
```

- We can assign values to multiple variables in one line
- Example

```
x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)
```

- One Value to Multiple Variables
- Example

```
x = y = z = "Orange"
print(x)
print(y)
print(z)
```

- **type() –** This function can be used to Get the Type of a variable
- Example

```
x = 5
y = "John"
print(type(x))
print(type(y))
```

- **String variables –** They can be declared either by using single or double quotes
- Example

```
x = "John"
# is the same as
x = 'John'
```

❖ **Python Operators –** Python supports a number of operator types, namely:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators

- **Arithmetic Operators-** they are used to perform arithmetic operations

Symbol	Meaning	Example	Result if i=5 and j=2
+	Addition	i+j	7
-	Subtraction	i–j	2
*	Multiplication	i*j	10
//	Integer	i//j	2
/	Division	i/j	2.5
%	Modulus	i%j	1
**	Exponent	i**j	25

- **Relational Operators**- they are used to perform comparison operations

Operator	Meaning	expression	Result if a=10 and b=20
==	Equal to	a==b	false
!= <>	Not Equal to	a!=b	true
>	Greater than	a>b	false
<	Less Than	a<b	true
>=	Greater Than or equal to	a>=b	false
<=	Less than or equal to	a<=b	true

- **Logical Operators**- they are used to perform comparison operations

Operator	Meaning	expression	Result if a=10, b=20, p=30,q=40
and	Logical And – return true when both the expressions are true	(a<b)and(p<q)	true
or	Logical OR – return true when any one of the expressions is true	(a>b)and(p<q)	True
not	Logical NOT- negates the expression	not(a<b)	False

- **Bitwise operators -** Bitwise operators are used to compare (binary) numbers

Operator	Meaning	expression
&	Bitwise AND	Sets each bit to 1 if both bits are 1
|	Bitwise OR	Sets each bit to 1 if one of two bits is 1
^	Bitwise XOR	Sets each bit to 1 if only one of two bits is 1
~	Bitwise NOT	Inverts all the bits
<<	Bitwise Left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Bitwise Right Shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

- **Identity operators** - are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location

Operator	Meaning	example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

- **Membership Operator** - are used to test if a sequence is presented in an object

Operator	Meaning	example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	returns True if a sequence with the specified value is not present in the object	x not in y

❖ Python expressions

- **Accepting user input** – python allows users to input values from the keyboard using the input () Method
- Example

```
username = input("Enter username:")
print("Username is: " + username)
```

However the input method by default treat input values as text. Hence it may required to explicitly cast the input inorder to handle numeric data.

- Example

```
x = int(input("Enter a number")
```

❖ Simple Python Programs

 # Program to perform various arithmetic operations

```
a=int(input("enter first number"))
b=int(input("enter second number"))
result = a+b
print("addition = ",result)
result = a-b
print("subtraction = ",result)
result = a*b
print("multipliction = ",result)
result=a/b
print("Division = ",result)
result = a//b
print(" integer Division = ",result)
result = a**b
print("a to the power of b = ",result)
```

 # Program to find the area of a triangle

```
b= int(input("enter base"))
h=int(input("enter height"))
area= 0.5*b*h
print("Rea of triangle is",area)
```

 # Program to find area of a circle

```
r=float(input("Enter the radius"))
area = 3.14 *r*r
print("area=",area)
```

 # Program to swap two numbers

```
a=int(input("Enter a number"))
b=int(input("enter second number"))
print("Before swappinga=",a,"b=",b)
temp=a
```

```
a=b
b=temp
print("After swapping a = ",a,"b= ",b)
```

- **Programming Exercises:**
  1. Write a Python Program to find the surface area of a cylinder
  2. Write a Python Program to convert temperature from celcius to farenheit
  3. Write a Python Program to find the area of a cone
  4. Write a Python Program to calculate displacement of an object
  5. Write a Python Program to compute Simple Interest

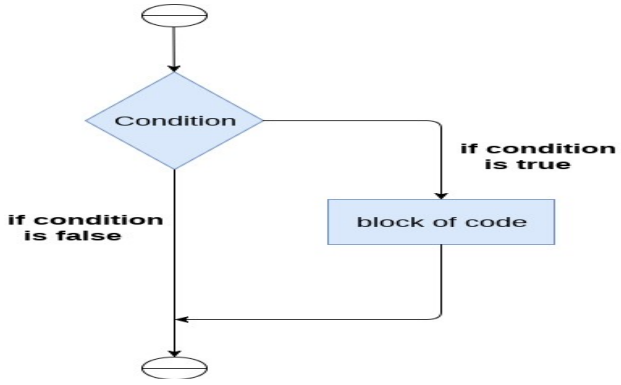
- **Control Structures:** Control Structures are used to control the flow execution of a program. Python provides following control structures(way of computation):
  - Straight line programs (Sequential)
  - Branching programs :
- **Conditional Execution -** Branching statements are also called conditional statements. A conditional statement has three parts
  1. A test
  2. Block of code to be executed if test is true
  3. Optional block of code if test is false

  There four types of if statements, viz,

- **The if statement -** The if statement is used to test a particular condition and if the condition is true, it executes a block of code known as if-block. The condition of if statement can be any valid logical expression which can be either evaluated to true or false.

- **Syntax:**

```
if expression:
        statement
```



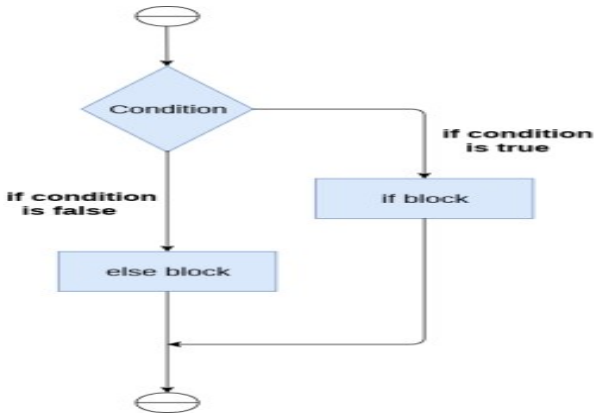
- Example 1

```
num = int(input("enter the number "))
if num%2 == 0:
    print("Number is even")
```

- **The if-else statement -**The if-else statement provides an else block combined with the if. In this form if the condition is true the if block is executed otherwise the else part is executed .

- **Syntax:**

```
if condition:
        #block of statements
else:
        # block of statements
```

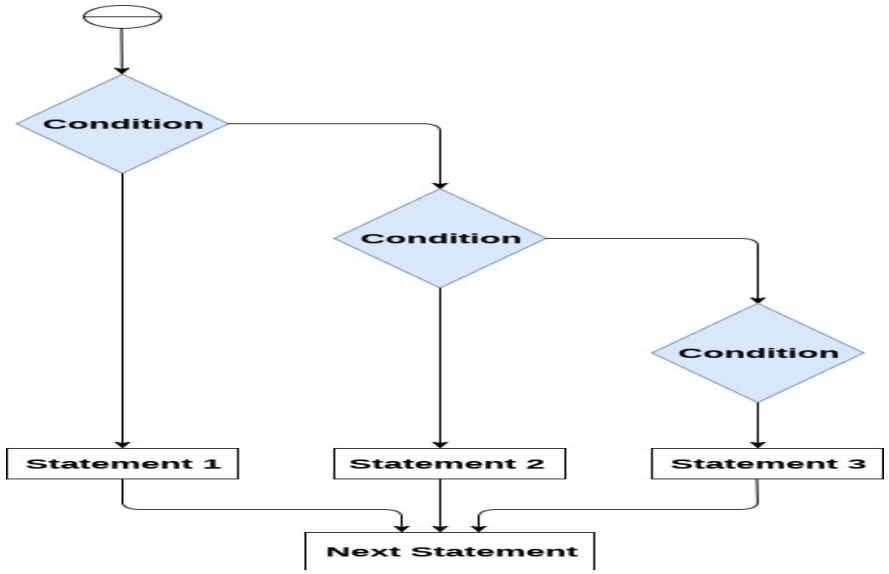


- Example 1 : Program to check whether a person is eligible to vote or not.

```
age = int (input("Enter your age? "))
if age>=18:
   print("You are eligible to vote ")
else:
   print("Not eligible to vote")
```

- **The elif statement (Chained Conditionals) -**The elif statement enables us to check multiple conditions and execute the specific block of statements depending upon the true condition among them. We can have any number of elif statements in our program depending upon our need. However, using elif is optional. The elif statement works like an if-else-if ladder statement in C.
- **Syntax :**

```
if expression 1:
   # block of statements
elif expression 2:
   # block of statements
elif expression 3:
   # block of statements
else:
   # block of statements
```



- Example 1

```
marks = int(input("Enter the marks? "))
if marks > 85:
        print("Congrats ! you scored grade A ...")
elif marks > 60:
        print("You scored grade B + ...")
elif marks > 40:
        print("You scored grade B ...")
elif marks > 30:
        print("You scored grade C ...")
else:
        print("Fail")
```

The nested if statement (Nested Conditionals) - You can have if statements inside if statements, this is called *nested* if statements. There may be a situation when you want to check for another condition after a condition resolves to true. In such a situation, you can use the nested if construct.

Syntax :

```
if expression1:
        if expression2:
                statement(s)
        else:
                 statement(s)
else:
                statement(s)
```

In a nested if construct, you can have an if...elif...else construct inside another if...elif...else construct

Example

```
if age > 50:
        if exp >10:
                sal = sal +1000
        else:
                 sal = sal +500
 else:
        sal = sal +100
```

- **Note - Indentation in Python**-For the ease of programming and to achieve simplicity, python doesn't allow the use of parentheses for the block level code. In Python, indentation is used to declare a block. If two statements are at the same indentation level, then they are the part of the same block.

- **Programs using conditionals**

#Program to increase the salary of an employee if his age is greater than 50

```
sal=5000
age = int(input("enter the age"))
if age>=50:
   sal=sal+500
print("Sal=",sal)
```

#Program to check whether a given number is even or odd

```
n=int(input("enter a number"))
if n%2 == 0:
   print("even number")
else:
   print("odd number")
```

#Program to check whether a given number is positive, negative or zero

```
n=int(input("enter a number"))
if n==0:
   print("The number is zero")
elif n>0:
   print("The number is positive")
else:
   print("The number negative")
```

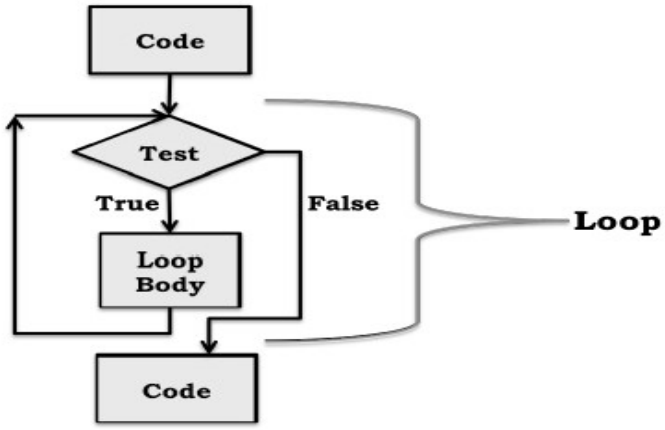
#Program to check whether a given number is equal to, greater than or less than second

```
a=int(input("enter first number"))
b=int(input("enter first number"))
if a==b:
    print("both numbers are equal")
elif a>b:
    print("First number is greater than second")
else:
    print("First number is less than second")
```

❖ **Programming Exercises:**

1. Write a Python Program to find the greatest among two numbers
2. Write a Python Program to check whether a student has passed or failed
3. Write a Python Program to check whether a student has scored distinction
4. Write a python program to check whether a person is eligible to vote
5. Write a program to check whether a given character is a vowel or a consonant
6. Write a program to check whether a number is positive negative or zero

- **Iteration/Looping:** It is a process of executing set of instructions repeatedly until a given condition is satisfied.



Looping statement begins with a test. If the test evaluates to True, the program executes the loop body once, and then goes back to reevaluate the test. This process is repeated until the test evaluates to False, after which control passes to the code following the iteration statement. The different looping statements in Python are: For loop and while loop

- **for loop** - for loop executes the block of statements until the given condition is satisfied. It is a entry controlled or a pre-tested loop. It is better to use for loop if the number of iteration is known in advance.

Syntax:

```
for <variable> in <sequence>:
        body_of_loop
```

Here <variable> is a variable that is used for iterating over a <sequence>. On every iteration it takes the next value from <sequence> until the end of sequence is reached.

- **Example:**

```
str = "Hello"

for i in str:
    print(i)
```

Output:

```
H
e
l
l
o
```

for loops are commonly used to process lists.

 # Program to print squares of all numbers present in a list

```
numbers = [1, 2, 4, 6, 11, 20]
sq = 0  # variable to store the square of each num temporary
# iterating over the given list
for val in numbers:
    sq = val * val
    print(sq)
```

for loop can also be used to process a range of values using range function

 # Program to print the sum of first 5 natural numbers

```
sum = 0
for val in range(1, 6):
    sum = sum + val
print(sum)
```

- **While loop:** It is also a pre-tested loop. In the while loop, the block of statements is executed until the specified condition is satisfied.

Syntax:

```
while test_expression:
    Body of loop
```

In the above syntax, test expression is checked first. The body of the loop is entered only if the test_expression evaluates to True. After one iteration, the test expression is checked again. This process continues until the test_expression evaluates to False. The while loop is to be used in the scenario where we don't know the number of iterations in advance. In Python, the body of the while loop is determined through indentation. Body starts with indentation and the first unindented line marks the end of the block.

 # Program to add natural numbers upto sum = 1+2+3+...+n

```
n = int(input("Enter n: "))
sum = 0
i = 1
while i <= n:
    sum = sum + i
    i = i+1
print("The sum is", sum)
```

- **Programs based on looping**

#Program to display even numbers between 1 to n

```
n=int(input("Enter the number :"))
for i in range(1,n+1):
    if (i%2)==0:
        print(i)
```

#Program to find the factorial of a given number

```
n = int(input('Enter a number: '))

if n<0:
    print('Enter a positive number')
else:
    fact = 1
    for i in range(1, n+1):
        fact = fact * i
    print('Factorial of', n, '=', fact)
```

 # Program to print first n fiboncii series

```
n=int(input("Enter value of n"))
print("Fibonacii series is:")

f1=0
f2=1
print(f1)
print(f2)

for i in range(3,n+1):
```

```
    f3=f1+f2
    print(f3)
    f1=f2
    f2=f3
```

#Program to reverse number and check if it is palindrome

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

temp=n
rev=0
while n>0:
    dig=n%10
    rev=(rev*10)+dig
    n=n//10

print("The reversed number is:",rev)
if(temp==rev):
    print("given number is palindrome")
else:
    print("given number is not palindrome")
```

#program to input a number and find sum of digits

```
n = int(input('Enter a number: '))
if n<0:
    print('Enter a positive number')
else:
    s = 0
    temp = n
    while n>0:
        digit = n % 10
        s = s+digit
        n = n//10
    print('Sum of digits of', temp, '=', s)
```

- **Programming Exercises:**
  1. Write a program to check whether a given number is Armstrong number
  2. Write a program to display the multiplication table of a given number
  3. Write a program to check whether a given number is prime or not
  4. Write program to find the value of a^n
  5. Write a program to find the sum of first n natural numbers
  6. Write a program to generate all even numbers up to n
  7. Write a program to generate all odd numbers up to n

FUNCTIONS

- **Python Functions**

Python allows us to divide a large program into the basic building blocks known as function. A function is subprogram or a subroutine that performs a particular task. A function can be defined as the organized block of reusable code which can be called whenever required.A function can be called multiple times to provide reusability and modularity to the python program.
Functions can be divided into 2 types.

1. **Built-in Functions:** These are in built functions defined in Python. They can be called in any program. Eg. range(), print(), len etc
2. User Defined Functions: The functions defined by the user or programmer are called user-defined functions.

- **Decalaring (Creating) a function**

In python, we can use **def** keyword to define the function.

- **Syntax**

def function_name(list of formal parameters):
" " " doc string" " "
Function_suite
return <expression>

The function block starts with the colon (:). Generally we should write a string as the first statement in the function body. This sstatement is known as the doc string that gives information about the function. However, the doc string is optional. The function suite consists of statements in that make up the function body. These block of statements is indented including the return statement. The last statement in the function block is the **optional** return statement that specifies the value returned by the function.

Example:

```
#Function definition
def sum( a,b ):
        " " " This function find sum" " "
        c=a+b
        return(c)
```

Calling a function

In python, a function must be defined before the function calling otherwise the python interpreter gives an error. Once the function is defined, we can call it from another function or the python prompt. To call the function, use the function name followed by the parentheses.

- **Syntax**
  Function_name(parameters)

- **Example:**
  #Function call
  Z=add(x,y)

- **Parameters**
  The information into the functions can be passed as the parameters. The parameters are specified in the parentheses. We can give any number of parameters, separated with a comma.

```
#python function to calculate the sum of two variables
#defining the function
def sum (a, b):
        c=a+b
        return c

x = int(input("Enter x: "))
y = int(input("Enter y: "))

z=sum(x, y)
print("Sum = ",z)
```

The parameters present in the function call are known as actual parameters and the parameter present in function definition are known as formal parameters. In the above example x,y are actual parameters and a,b are formal parameters. There is one to one correspondence between actual and formal parameters. Hence the number, data type and position of th parameters must match

- **Default Values**
  Python allows us to initialize the arguments at the function definition. If the value of any of the argument is not provided at the time of function call, then that argument can be initialized with the value given in the definition even if the argument is not specified at the function call. Hence, an argument assumes a default value if a value is not provided to aparemeter in the function call.

Example:

```
def grocery(item, price=40.00):
                print(“item =”, item)
                print(“price=”,price)

#function call
grocery(“sugar”, 50.50)      #price is set to 50.00
```

grocery("sugar")

- **Void and fruitful functions**

Void Functions	Fruitful Functions
Do not return a value	Always return a value
No need to assign it to a variable	Must assign it to a variable to hold return value
return statement may or may not be present	return statement is always present
Example def greeting(name) print ("hello") return	Example def add(a, b) c=a+b return c

- **Recursion**

Recursion is a process in which a function calls itself repeatedly till certain condition is met. Functions that incorporate recursion are called recursive functions.

- **Examples:**

#Recursive factorial function

```
def fact(n):
    if n==0 or n==1:
        return 1
    else:
        return n*fact(n-1)

n=int(input("Enter the value of n"))

f=fact(n)
print("Factorial of",n,"=",f)
```

#Recursive function to compute the value of a^n

```
def power(a,n):
    if n==0:
        return 1
    elif n==1:
        return a
    else:
        return a*power(a,n-1)

a=int(input("Enter the vlue of base"))
n=int(input("Enter the value of exponent"))

f=power(a,n)
```

```
print(a,"to the power of",n,"=",f)
```

#Recursive function to generate fibonacii numbers

```
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-1) + fib(n-2)

n=int(input('How many elements?'))
for i in range(0,n):
    print(fib(i))
```

Advantages of Recursion:

1. A recursive code has a cleaner-looking code.
2. Recursion makes it easier to code, as it breaks a task into smaller ones.
3. It is easier to generate a sequence using recursion than by using nested iteration

- **Scope**

  Defn:Scope defines portion of programs where we can access a particular identifier. The scopes of the variables depend upon the location where the variable is being declared.
  In python, the variables are defined with the two types of scopes.
  1. **Global variables** : The variable defined outside any function is known to have a global scope Global variables can be accessed throughout the program

```
def strfun():
    print (s)

# Global scope
s = "I like Python"
strfun()
print (s)
```

  Output:
  I like Python
  I like Python
  In the above program s is global string variable that can be accessed anywhere, Hence it produces the above output.

  2. **Local variables** : The variable defined inside a function is known to have a local scope. Such variables cannot be accessed outside the function.

```
def strfun():
    s= "i like python"
    print(s)

#main program
strfun()
print(s)
```

Output:
i like python
NameError: name 's' is not defined

In the above program s is local string variable that is defined inside the function. It can be accessed only inside the function. Hence it produces an error.

Modules

Usually we store entire program code in one file. When programs get larger and multiple people work on it, it is more convenient to store different parts of the program in different files. Python modules allow us to easily construct a program from code in multiple files.

- **Defn :** A module is a file containing Python function definitions and statements. It is saved with **.py** extention. Modules in Python provides us the flexibility to organize the code in a logical way. To use the functionality of one module into another, we must have to **import** the specific module.

- **Loading the module in our python code**

We need to load the module in our python code (program) to use its functionality. Python provides two types of import statements as defined below.

1. *The import statement*
2. *The from-import statement*

- **The import statement :**

The import statement is used to import all the functionalities (functions) of one module into another. We can import multiple modules with a single import statement.

Syntax:

import module1,module2,…….. module n

The from-import statement

Instead of importing the whole module into the namespace, python provides the flexibility to import only the specific functions of a module. This can be done by using **from import** statement.

Syntax :

from < module-name> import <name 1>, <name 2>..,<name n>

from calculation import add,sub

We can also import all the functions from a module by using *.

from <module> import *

#python Program to demonstrate modules

#module m1

```
def greeting(name):
    print("hello",name)
```

#Program to call the module

```
import m1
m1.greeting("india")
```

#Write a Python program to create a module Calculation.py that contains functions to perform basic arithmetic operations. Demonstrate using module

#module calculation

```
def add(a,b):
    res = a+b
    return res

def sub(a,b):
    res = a-b
    return res

def prod(a,b):
    res = a*b
    return res

def div(a,b):
    res = a/b
    return res
```

#Program to call the module calculation

```
import Calculation
```

```
a = int(input('Enter first number: '))
b = int(input('Enter second number: '))

res = Calculation.add(a, b)
print(a, '+', b, '=', res)

res = Calculation.sub(a, b)
print(a, '-', b, '=', res)

res = Calculation.prod(a, b)
print(a, '*', b, '=', res)

res = Calculation.div(a, b)
print(a, '/', b, '=', res)
```

Programming Exercises:

1. Write Program to create a module area.py that contains functions to calculate the area of various geometric objects
2. Write Program to create a module that contains functions to test a number for positive or negative, even or odd, finds the sum of digits and reverses a number