Unit – 1

Introduction to Python Programming

1.1 Overview of Python Programming

Python is a high-level programming language known for its simplicity and readability. It was created by Guido van Rossum and first released in 1991. Python is widely used in various domains such as web development, data analysis, artificial intelligence, scientific computing, and more.

➢ **Key Features of Python:**

1. **Easy to Learn and Read:** Python's syntax is designed to be intuitive and easy to understand, making it accessible for beginners.
2. **Interpreted Language:** Python code is executed line by line by the Python interpreter, which means you don't need to compile your code before running it.
3. **Extensive Standard Library:** Python comes with a large collection of modules and libraries that provide ready-to-use functions and classes for tasks like working with files, handling network communication, and processing data.
4. **Dynamic Typing:** Python uses dynamic typing, meaning you don't need to specify variable types explicitly. The interpreter automatically assigns types during execution.
5. **Support for Multiple Platforms:** Python is available for Windows, macOS, Linux, and various other platforms, making it highly versatile.
6. **Large Community Support:** Python has a vast and active community of developers who contribute to libraries, frameworks, and resources that further extend its capabilities.

➢ **Common Uses of Python:**

- **Web Development:** Frameworks like Django and Flask are popular for building web applications.
- **Data Analysis and Visualization:** Libraries such as Pandas, NumPy, and Matplotlib are widely used for data manipulation, analysis, and visualization.
- **Artificial Intelligence and Machine Learning:** Python is the language of choice for many AI and ML applications due to libraries like TensorFlow, PyTorch, and scikit-learn.
- **Scripting:** Python is excellent for writing small scripts to automate repetitive tasks.

- **Game Development:** With libraries like Pygame, Python is used for creating games and simulations.

➢ **Python Syntax**

Syntax in a programming language is a standard way of expressing values or statements which every programming language follows.

To print a statement-
print(“Hello World”)
Output: Hello World

1.2 Variables

Python variables are the reserved memory locations used to store values with in a Python Program. This means that when you create a variable you reserve some space in the memory.

Based on the data type of a variable, Python interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to Python variables, you can store integers, decimals or characters in these variables.

The Python variables are classified in categories –

1. Local Variables
2. Global Variables

Local Variables

A local variable is defined within a specific function or block of code. It can only be accessed by the function or block where it was defined, and it has a limited scope.

Example :

```
def myfunction():

    a = 10

    b = 20

    print("variable a:", a)

    print("variable b:", b)
```

```
        return a+b

        print (myfunction())
```

OUTPUT : variable a: 10
variable b: 20
30

Global Variables

A global variable can be accessed from any part of the program, and it is defined outside any function or block of code. It is not specific to any block or function.

Example :

```
name = 'Python'

students = 50

def myfunction():

 # accessing inside the function

 print("name:", name)

 print("students:", students)

# function call

 myfunction()
```

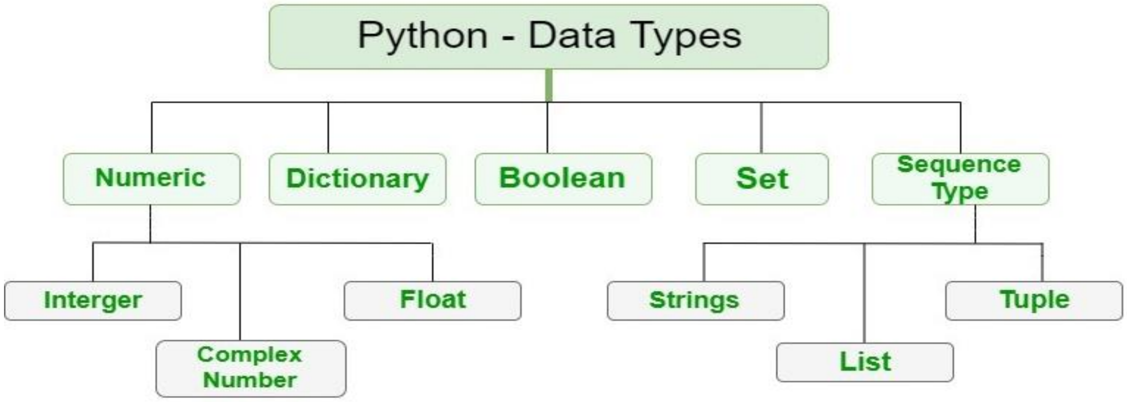
OUTPUT:

name: Python

students : 50

1.3 DataTypes

Python Data Types are used to define the type of a variable. It defines what type of data we are going to store in a variable. The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters.



Basic Data types in Python

1. **Integer (int):**
   - **Description:** Whole numbers without a decimal point.
   - **Examples:** -3, 0, 42
   - **Usage:** To store counts, indices, or any number without a fractional part.
2. **Floating-point (float):**

- **Description:** Numbers that contain a decimal point.
- **Examples:** -2.5, 0.0, 3.14
- **Usage:** To store measurements, precise calculations, or any number with a fractional part.

3. **String (str):**
   Description: A sequence of characters enclosed in single, double, or triple quotes.
   Examples: "hello", '1234', '''abc@'''
   Usage: To store text, names, or any sequence of characters.

4. **Boolean (bool):**

Description: Represents true or false values.

Examples: True, False

Usage: To store conditions, results of comparisons, or any binary state.

Complex Datatypes in Python

1. **List:**
    - **Description:** An ordered collection of values, which can be of different types, enclosed in square brackets.
    - **Examples:** [1, 2, 3], ["apple", "banana", "cherry"]
    - **Usage:** To store collections of items, such as a list of numbers or strings.
2. **Tuple:**

- **Description:** Similar to a list, but immutable (cannot be changed after creation), enclosed in parentheses.
- **Examples:** (1, 2, 3), ("red", "green", "blue")
- **Usage:** To store collections of items that should not change.

3. **Dictionary (dict):**

- **Description:** A collection of key-value pairs, enclosed in curly braces.
- **Examples:** {"name": "Alice", "age": 25}, {"fruit": "apple", "color": "red"}
- **Usage:** To store data that can be accessed by unique keys.

4. **Set:**

- **Description:** An unordered collection of unique values, enclosed in curly braces.
- **Examples:** {1, 2, 3}, {"apple", "banana", "cherry"}
- **Usage:** To store unique items, such as a list of unique words.

1.4 Operators

Operators in general are used to perform operations on values and variables. These are standard symbols used for logical and arithmetic operations.

- Arithmetic operators
- Comparison operators
- Assignment Operators
- Logical Operators
- Bitwise Operators

Arithmetic Operators

Arithmetic operators used between two operands for a particular operation. There are many arithmetic operators. It includes the exponent (**) operator as well as the + (addition), - (subtraction), * (multiplication), / (divide), % (reminder), and // (floor division) operators.

Consider the following table for a detailed explanation of arithmetic operators.

+ (Addition)	It is used to add two operands. For example, if a = 10, b = 10 => a+b = 20
- (Subtraction)	It is used to subtract the second operand from the first operand. If the first operand is less than the second operand, the value results negative. For example, if a = 20, b = 5 => a - b = 15
/ (divide)	It returns the quotient after dividing the first operand by the second operand. For example, if a = 20, b = 10 => a/b = 2.0
*** (Multiplication)**	It is used to multiply one operand with the other. For example, if a = 20, b = 4 => a * b = 80
% (reminder)	It returns the reminder after dividing the first operand by the second operand. For example, if a = 20, b = 10 => a%b = 0
**** (Exponent)**	As it calculates the first operand's power to the second operand, it is an exponent operator.
// (Floor division)	It provides the quotient's floor value, which is obtained by dividing the two operands.

Program :

1. a = 32   # Initialize the value of a
2. b = 6   # Initialize the value of b
3. print('Addition of two numbers:',a+b)
4. print('Subtraction of two numbers:',a-b)
5. print('Multiplication of two numbers:',a*b)
6. print('Division of two numbers:',a/b)

7. print('Reminder of two numbers:',a%b)
8. print('Exponent of two numbers:',a**b)
9. print('Floor division of two numbers:',a//b)

Output:
Addition of two numbers: 38
Subtraction of two numbers: 26
Multiplication of two numbers: 192
Division of two numbers: 5.3333333
Reminder of two numbers: 2
Exponent of two numbers: 1073741824
Floor division of two numbers: 5

Comparison operator

Comparison operators mainly use for comparison purposes. Comparison operators compare the values of the two operands and return a true or false Boolean value in accordance. The example of comparison operators are ==, !=, <=, >=, >, <. In the below table, we explain the works of the operators.

Operator	Description
==	If the value of two operands is equal, then the condition becomes true.
!=	If the value of two operands is not equal, then the condition becomes true.
<=	The condition is met if the first operand is smaller than or equal to the second operand.
>=	The condition is met if the first operand is greater than or equal to the second operand.
>	If the first operand is greater than the second operand, then the condition becomes true.

<	If the first operand is less than the second operand, then the condition becomes true.

Program

```
a = 32      # Initialize the value of a
b = 6       # Initialize the value of b
print('Two numbers are equal or not:',a==b)
print('Two numbers are not equal or not:',a!=b)
print('a is less than or equal to b:',a<=b)
print('a is greater than or equal to b:',a>=b)
print('a is greater b:',a>b)
print('a is less than b:',a<b)
```

Output:

```
Two numbers are equal or not: False
Two numbers are not equal or not: True
a is less than or equal to b: False
a is greater than or equal to b: True
a is greater b: True
a is less than b: False
```

Assignment Operators

Using the assignment operators, the right expression's value is assigned to the left operand. There are some examples of assignment operators like =, +=, -=, *=, %=, **=, //=. In the below table, we explain the works of the operators.

Operator	Description
=	It assigns the value of the right expression to the left operand.

+=	By multiplying the value of the right operand by the value of the left operand, the left operand receives a changed value. For example, if a = 10, b = 20 => a+ = b will be equal to a = a+ b and therefore, a = 30.
-=	It decreases the value of the left operand by the value of the right operand and assigns the modified value back to left operand. For example, if a = 20, b = 10 => a- = b will be equal to a = a- b and therefore, a = 10.

Program

```
a = 32         # Initialize the value of a
b = 6          # Initialize the value of b
print('a=b:', a==b)
print('a+=b:', a+b)
print('a-=b:', a-b)
```

Output:
a=b: False
a+=b: 38
a-=b: 26

Bitwise Operators

The two operands' values are processed bit by bit by the bitwise operators. The examples of Bitwise operators are bitwise OR (|), bitwise AND (&), bitwise XOR (^), negation (~), Left shift (<<), and Right shift (>>).

Operator	Description	Syntax
&	Bitwise AND	x & y
|	Bitwise OR	x | y

Operator	Description	Syntax
~	Bitwise NOT	~x
^	Bitwise XOR	x ^ y
>>	Bitwise right shift	x>>
<<	Bitwise left shift	x<<

Example:

```
a = 10
b = 4
print(a & b)
print(a | b)
print(~a)
print(a ^ b)
print(a >> 2)
print(a << 2)
```

Output

```
0

14

-11

14

2

40
```

Logical Operators

The assessment of expressions to make decisions typically uses logical operators. The examples of logical operators are and, or, and not. In the case of logical AND, if the first one is 0, it does not depend upon the second one. In the case of logical OR, if the first one is 1, it does not depend on the second one. Python supports the following logical operators. In the below table, we explain the works of the logical operators.

Operator	Description	Syntax
and	Logical AND: True if both the operands are true	x and y
or	Logical OR: True if either of the operands is true	x or y
not	Logical NOT: True if the operand is false	not x

Example

```
a = True
b = False
print(a and b)
print(a or b)
print(not a)
```

OUTPUT

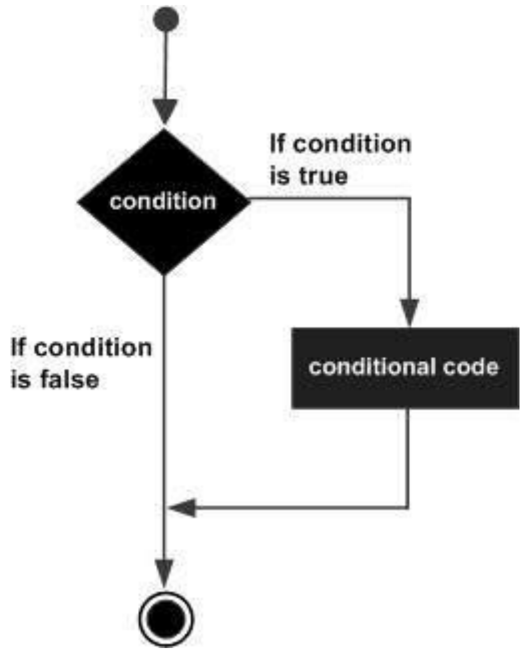
```
False
True
False
```

1.5 Conditional Statements

Conditional Statements are statements in Python that provide a choice for the control flow based on a condition. It means that the control flow of the Python program will be decided based on the outcome of the condition.

1.5.1 If Statement

If the simple code of block is to be performed if the condition holds then the if statement is used. Here the condition mentioned holds then the code of the block runs otherwise not.



Syntax of If Statement: if condition:

```
    # Statements to execute if
    # condition is true
```

Program :

```
# if statement example

if 10 > 5:

    print("10 greater than 5")
```

Output : 10 greater than 5

1.5.2 If else Statements

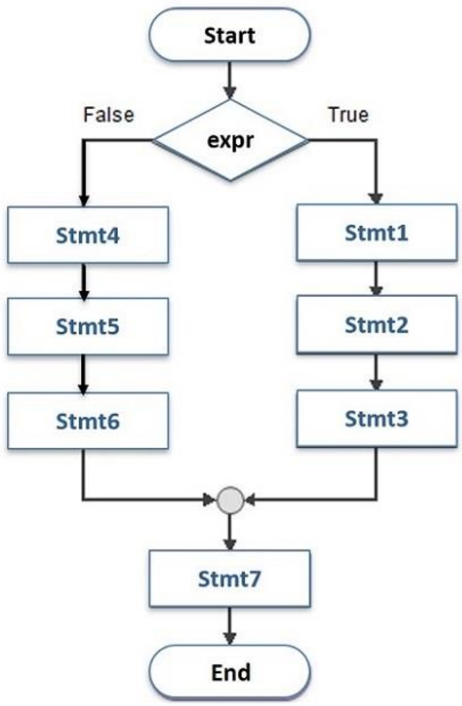
The if-else statement in Python is used to execute a block of code when the condition in the **if statement** is true, and another block of code when the condition is false.

Syntax of if-else Statement

```
if (condition):
    # Executes this block if
    # condition is true
else:
    # Executes this block if
    # condition is false
```

If the boolean expression evaluates to TRUE, then the statement(s) inside the **if block** will be executed otherwise statements of the **else block** will be executed.

Flowchart of if-else Statement



Program :

```
age=25

print ("age: ", age)

if age >=18:

  print ("eligible to vote")

else:

  print ("not eligible to vote")
```

OUTPUT

On executing this code, you will get the following **output** −

age: 25
eligible to vote

To test the **else block**, change the **age** to 12, and run the code again.

age: 12
not eligible to vote

1.5.3 Nested if..else Statements

The nest **if-else** statement within an **if statement**. If the **if condition** is true, the first **if-**

else statement will be executed otherwise, statements inside the **else block** will be executed.

Syntax

```
if expression1:
   statement(s)
   if expression2:
      statement(s)
   else
      statement(s)
else:
   if expression3:
      statement(s)
   else:
      statement(s)
```

Example

```
num=8

print ("num = ",num)

if num%2==0:

   if num%3==0:
```

```
    print ("Divisible by 3 and 2")
  else:
    print ("divisible by 2 not divisible by 3")
else:
  if num%3==0:
    print ("divisible by 3 not divisible by 2")
  else:
    print ("not Divisible by 2 not divisible by 3")
```

When the above code is executed, it produces the following **output** −

```
num = 8
divisible by 2 not divisible by 3
num = 15
divisible by 3 not divisible by 2
num = 12
Divisible by 3 and 2
num = 5
not Divisible by 2 not divisible by 3
```

1.6 Loops

Loops in Python are used to repeatedly execute a block of code as long as a certain condition is met. There are two main types of loops in Python: for loops and while loops. Here’s a detailed explanation of each:

- For loop

  The **for loop** in Python provides the ability to loop over the items of any sequence, such as a list, tuple or a string. It performs the same action on each item of the sequence. This loop starts with the **for** keyword, followed by a variable that represents the current item in the sequence. The **in** keyword links the variable to the sequence you want to iterate over. A **colon (:)** is used at the end of the loop header, and the indented block of code beneath it is executed once for each item in the sequence.

Syntax of Python for Loop
for iterating_var in sequence:

statement(s)

Example:

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

Output :

apple
banana
cherry

Example:

```
for x in "banana":

  print(x)
```

Output :

```
b
a
n
a
n
a
```

- **While Loop**
  A **while loop** in Python programming language repeatedly executes a target statement as long as the specified boolean expression is true. This loop starts with **while keyword** followed by a boolean expression and colon symbol (:). Then, an indented block of statements starts.

Here, statement(s) may be a single statement or a block of statements with uniform indent. The condition may be any expression, and true is any non-zero value. As soon as the expression becomes false, the program control passes to the line immediately following the loop.

If it fails to turn false, the loop continues to run, and doesn't stop unless forcefully stopped. Such a loop is called infinite loop, which is undesired in a computer program.

Syntax of while Loop

while expression:

statement(s)

Example:

```
t = 0

while (t < 3):
    t = t + 1
    print("Hello")
```

Output :

```
Hello

Hello

Hello
```

➢ Continue Statement

Python **continue statement** is used to skip the execution of the program block and returns the control to the beginning of the current loop to start the next iteration. When encountered, the loop starts next iteration without executing the remaining statements in the current iteration.

The **continue statement** is just the opposite to that of break. It skips the remaining statements in the current loop and starts the next iteration.

Syntax

```
looping statement:
  condition check:
    continue
```

Example:

```
for letter in 'Python':
  if letter == 'h':
    continue
  print ('Current Letter :', letter)
```

Output :

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : o
Current Letter : n
```

➢ Break Statement

B**reak statement** is used to terminate the current loop and resumes execution at the next statement, just like the traditional break statement in C.

The most common use for Python break statement is when some external condition is triggered requiring a sudden exit from a loop. The **break statement** can be used in both Python while and for loops.

If you are using nested loops in Python, the break statement stops the execution of the innermost loop and start executing the next line of code after the block.

Syntax

looping statement:

condition check:

break

Example:

```
for letter in 'Python':
  if letter == 'h':
    break
  print ("Current Letter :", letter)
```

Output :

```
Current Letter : P
Current Letter : y
Current Letter : t
```

➢ Pass Statement

P**ass statement** is used when a statement is required syntactically but you do not want any command or code to execute. It is a null which means nothing happens when it executes. This is also useful in places where piece of code will be added later, but a placeholder is required to ensure the program runs without errors. For instance, in a function or class definition where the implementation is yet to be written, **pass statement** can be used to avoid the **SyntaxError**. Additionally, it can also serve as a placeholder in control flow statements like for and while loops.

Syntax : pass

Example:

```
for letter in 'Python':
  if letter == 'h':
     pass
     print ('This is pass block')
  print ('Current Letter :', letter)
```

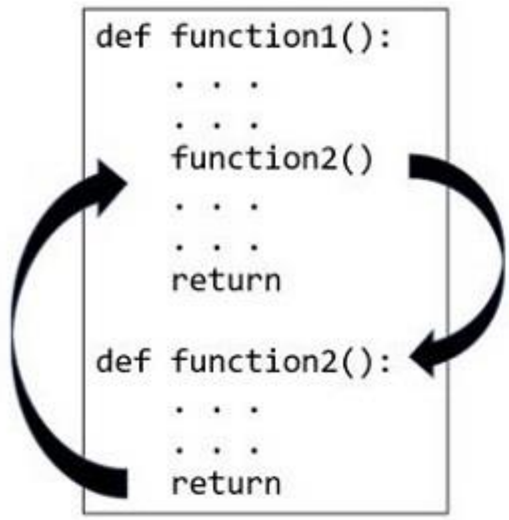
Output :

```
Current Letter : P
Current Letter : y
Current Letter : t
This is pass block
Current Letter : h
Current Letter : o
Current Letter : n
```

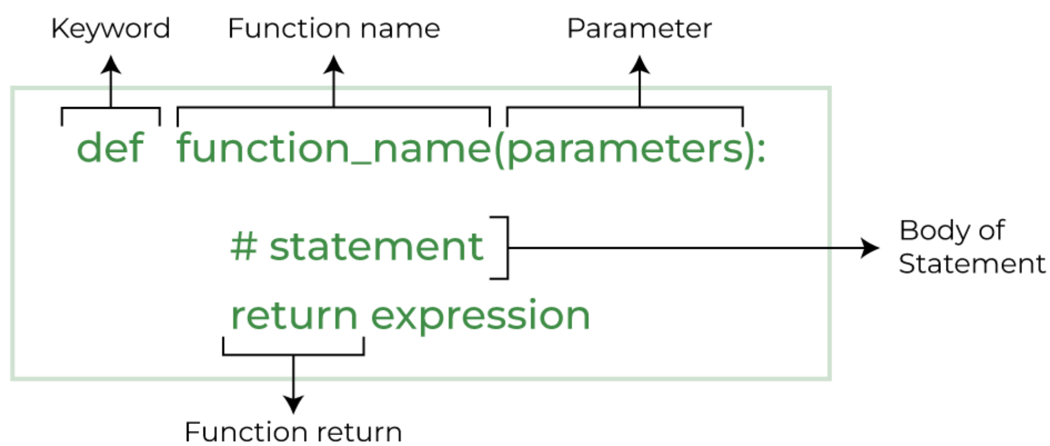
1.7 Functions

A Python function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

A top-to-down approach towards building the processing logic involves defining blocks of independent reusable functions. A Python function may be invoked from any other function by passing required data (called **parameters** or **arguments**). The called function returns its result back to the calling environment.



Syntax/ Declaration



Types of Functions

- **Built-in library function:** These are Standard functions in Python that are available to use.
- **User-defined function:** We can create our own functions based on our requirements.

Python's standard library includes number of built-in functions. Some of Python's built-in functions are print(), int(), len(), sum(), etc. These functions are always available, as they are loaded into computer's memory as soon as you start Python interpreter.

The standard library also bundles a number of modules. Each module defines a group of functions. These functions are not readily available. You need to import them into the memory from their respective modules.

In addition to the built-in functions and functions in the built-in modules, you can also create your own functions. These functions are called user-defined *functions*.

How to define a function:

You define a function using the def keyword, followed by the function name and parentheses. Inside the parentheses, you can specify parameters that the function can accept. The function body contains the code that will be executed when the function is called.

Syntax:

```
def function_name(parameters):

    # Function body

    # Code to be executed

    return value  # Optional
```

Example:

```
def fun():
    print("Welcome to class")
```

Output:

```
Welcome to class
```

Example:

Here's a simple example of a function that adds two numbers:

```
def add_numbers(a, b):
    result = a + b
    return result
```

Explanation:

- def add_numbers(a, b): defines a function named add_numbers that takes two parameters, a and b.
- Inside the function, result = a + b calculates the sum of a and b.
- return result returns the calculated sum to the caller.

Calling a function:

To use (or "call") a function, you write its name followed by parentheses. If the function takes parameters, you provide the arguments inside the parentheses.

```
sum = add_numbers(3, 5)
print(sum)  # Output: 8
```

Explanation:

- add_numbers(3, 5) calls the add_numbers function with 3 and 5 as arguments.
- The function returns the sum, which is then stored in the variable sum.
- print(sum) prints the result, which is 8.

Functions without parameters:

A function doesn't have to take parameters. Here's an example of a function without parameters:

```
def greet():
    print("Hello, world!")
```

1.8 Modules

A module is a file containing definition of functions, classes, variables, constants or any other Python object. Contents of this file can be made available to any other program. Python has the **import** keyword for this purpose.

Example:

```
import math

print ("Square root of 100:", math.sqrt(100))
```

Output :

```
Square root of 100: 10.0
```

➢ The import Statement

In Python, the **import** keyword has been provided to load a Python object from one

module. The object may be a function, class, a variable etc. If a module contains multiple definitions, all of them will be loaded in the namespace.

Let us save the following code having three functions as **mymodule.py.**

```
def sum(x,y):
    return x+y
def average(x,y):
    return (x+y)/2
def power(x,y):
    return x**y
```

The **import mymodule** statement loads all the functions in this module in the current namespace. Each function in the imported module is an attribute of this module object.

To call any function, use the module object's reference. For example, mymodule.sum().

```
import mymodule
print ("sum:",mymodule.sum(10,20))
print ("average:",mymodule.average(10,20))
print ("power:",mymodule.power(10, 2))
```

It will produce the following **output** −

```
sum:30
average:15.0
power:100
```