
**MGM's**

# Jawaharlal Nehru Engineering College Aurangabad

**MGM University, Aurangabad**

## Department of Computer Science & Engineering

# LAB MANUAL

| | |
|---|---|
| Program (UG/PG) : | UG |
| Year : | First Year |
| Semester : | I |
| Course Code : | 20UCC104BL |
| Course Title : | Python Programming |
| Prepared By : | Ms. D. R. Naglot |

Department of Computer Science & Engineering

**2021-22**

# FOREWORD

It is our great pleasure to present this laboratory manual for First Year engineering students for the subject of Computer Programming for Problem Solving

As a student, many of you may be wondering with some of the questions in your mind regarding the subject and exactly what has been tried is to answer through this manual.

As you may be aware that MGM has already been awarded with ISO 9001:2015,140001:2015 certification and it is our endure to technically equip our students taking the advantage of the procedural aspects of ISO Certification.

Faculty members are also advised that covering these aspects in initial stage itself, will greatly relived them in future as much of the load will be taken care by the enthusiasm energies of the students once they are conceptually clear.

Dr. H. H. Shinde
Principal

## LABORATORY MANUAL CONTENTS

This manual is intended for the First year students of Computer Science and Engineering in the subject of Computer Programming for Problem Solving. This manual typically contains practical/Lab Sessions related Computer Programming for Problem Solving covering various aspects related the subject to enhanced understanding.

Students are advised to thoroughly go through this manual rather than only topics mentioned in the syllabus as practical aspects are the key to understanding and conceptual visualization of theoretical aspects covered in the books.

Good Luck for your Enjoyable Laboratory Sessions

Prof. Vijaya B. Musande

Subject Teacher HOD

# LIST OF EXPERIMENTS

| | | |
|---|---|---|
| **Course Code :** 20UCC104BL | **Course Title**<br>***Python Programming Lab*** | **Total Credits** : 01 |
| **Teaching Scheme** | | **Evaluation Scheme:** |
| **Theory:** | | CA : 60 |
| **Tutorial :** -- | | End Sem: 40 |
| **Practical :** 2 Hrs/ Week | | |

| | |
|---|---|
| **Lab Outcomes** | **LO1 :** Demonstrate python program using development environment.<br>**LO2:** Develop logical thinking to solve the problems using programming fundamental concepts.<br>**LO3 :** Construct python program using various data structures.<br>**LO4 :** Apply modularization approach for solving complex problem.<br>**LO5:** Make use of various packages in Python for data science.<br>**LO6 :** Implement different SQL commands in python. |
| **Pre-requisite** | Nil |
| **Course Type** | **Program Core Course** |

**LABORATORY CONTENTS**

| SN | Name of Practical | LO mapping | PO mapping | PSO mapping |
|---|---|---|---|---|
| 1 | **Program to perform input/output operations**<br>1. Write a program to take input (integer, float, string) and print it. | LO 1 | PO1 | PSO1 |

| | | | | |
|---|---|---|---|---|
| 2 | **Program based on operators**<br>1.Write a program to simulate a simple calculator (+ - / * %) that takes two operands as input and displays the result<br>2. Write a program to find area and perimeter of geometric objects.<br>3. The distance between two cities (in km.) is input through the keyboard. Write a program to convert and print this distance in meters, feet, inches and centimeters.<br>4. Write a C Program to interchange two numbers.<br>5. Write a program to compute Fahrenheit from centigrade | LO2 | PO1<br>PO2<br>PO9 | PSO1 |
| 3 | **Programs based on Decision making.**<br>1. Write a program to read marks from keyboard and your program should display equivalent grade according to following table(else-if ladder)<br>Marks Grade<br>100 - 80 Distinction<br>79 - 60 First Class<br>59 - 40 Second Class<br>< 40 Fail<br>2. Write a program to input basic salary of an employee and calculate gross salary according to given conditions.<br>Basic Salary <= 10000 : HRA = 20%, DA = 80%<br>Basic Salary is between 10001 to 20000 : HRA = 25%, DA = 90%<br>Basic Salary >= 20001 : HRA = 30%, DA = 95%<br>3. If the ages of three brothers are input through the keyboard, write a C Program to determine the youngest and oldest of the three.<br>4. Write a program to calculate overtime pay of employee. Overtime is paid at the rate of Rs. 12.00 | LO2 | PO1<br>PO2<br>PO9 | PSO1 |

| | | | | |
|---|---|---|---|---|
| | per hour for every hour worked above 40 hours. Assume that employee do not work for fractional part of an hour.<br>5. Write a program for checking the speed of drivers.<br>If speed is less than 70, it should print "Ok". Otherwise, for every 5km above the speed limit (70), it should give the driver one demerit point and print the total number of demerit points. For example, if the speed is 80, it should print: "Points: 2". If the driver gets more than 12 points, the function should print: "License suspended" | | | |
| 4 | **Programs using while and for loops**<br>1. WAP to find factorial of given number<br>2. WAP to check whether given number is Palindrome or not<br>3. WAP to check whether given number is Armstrong or not<br>4. WAP to print Fibonacci series<br>5. Write a Python program which iterates the integers from 1 to 50. For multiples of three print "Fizz" instead of the number and for the multiples of five print "Buzz". For numbers which are multiples of both three and five print "FizzBuzz".<br>6. WAP to check whether given number is Perfect number or not<br>7. WAP to check whether given number is Prime number or not<br>8. Write C program to print given star and number patterns and reverse it.<br>* 1<br>** 12<br>*** 123<br>**** 1234 | LO2 | PO1<br>PO2<br>PO9 | PSO1 |

| | | | | | |
|---|---|---|---|---|---|
| 5 | **Programs on string**<br>1. Write Python Program to find length of string without using len() function.<br>2. Count all letters, digits, and special symbols from a given string.<br>3. Python Program to Count the Number of Vowels in a String.<br>4. Python Program to Calculate the Number of Upper Case Letters and Lower Case Letters in a String.<br>5. Python Program to Check whether given string is palindrome or not | LO3 | PO1<br>PO2<br>PO9 | PSO1 | |
| 6 | **Programs on List and Tuple**<br>1. Write a Python program to sum all the items in a list.<br>2. Write a Python program to multiply all the items in a list<br>3. Write a Python program to get the largest number from a list.<br>4. Write a Python program to get the smallest number from a list<br>5. Write a Python program to count all elements in list and count Occurrences Of A List Item In Python<br>6. Write a Python program to create a tuple with different data types<br>7. Write a Python program to check whether an element exists within a tuple<br>8. Write a Python program to reverse a tuple<br>9. Write a Python program calculate the product of all the numbers given in tuple. Original Tuple: (2, 4, 8, 8, 3, 2, 9)<br>Product - multiplying all the numbers of the said tuple: 27648 | LO3 | PO1<br>PO2<br>PO9 | PSO1 | |
| 7 | **Programs on set and dictionary**<br>1. Write a Python program to concatenate following dictionaries to create a new one. Sample Dictionary : dic1={1:10, 2:20} dic2={3:30, 4:40} dic3={5:50,6:60}<br>Expected Result : {1: 10, 2: 20, 3: 30, 4: 40, 5: 50, 6: 60}<br>2. Write a Python program to check whether a given | LO3 | PO1<br>PO2<br>PO9 | PSO1 | |

| | | | | | |
|---|---|---|---|---|---|
| | key already exists in a dictionary<br>3. Write a Python script to generate and print a dictionary that contains a number (between 1 and n) in the form (x, x*x) Sample Dictionary ( n = 5) : Expected Output : {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}<br>4. Write a Python program to merge two Python dictionaries<br>5. Write a Python program to get the maximum and minimum value in a dictionary<br>6. Write a Python program to create set difference, Union and intersection<br>7. Write a Python program to check if two given sets have no elements in common | | | | |
| 8 | **Programs using function**<br>1. Write Functions to calculate your trip's costs:<br>Define a function called hotel_cost with one argument nights as input<br>Define a function called plane_ride_cost that takes a string, city, as input.<br>Define a function called rental_car_cost with an argument called days.<br>Define a function called trip_cost that takes two arguments, city and days. Like the example above, have your function return the sum of calling the rental_car_cost(days), hotel_cost(days), and plane_ride_cost(city) functions.<br>2. Write a program in to check a given number is even or odd using the function.<br>3. Write a function Exchange to interchange the values of two variables, say x and y. illustrate the use of this function in a calling function.<br>4. Write a program to find Sum of natural number using recursion.<br>5. Write a program to print Fibonacci series number using recursion | LO4 | PO1<br>PO2<br>PO9 | PSO1 | |
| 9 | **Program using NumPy, Matplotlib and Pandas library** | LO5 | PO1<br>PO2 | PSO1 | |

| | | | | | |
|---|---|---|---|---|---|
| | 1. Write a program to perform matrix addition, substraction, multiplication.<br>2. Plot all types of graph using Matplotlib.<br>3. Write a program which perform basic operation of Pandas. | | PO9 | | |
| 10 | **Program on SQL Commands**<br>1. Write a program of binary search<br>2. Write a program which perform basic SQL commands | LO6 | PO1<br>PO2<br>PO9 | PSO1 | |

## DOs and DON'Ts in Laboratory:

1. Make entry in the Log Book as soon as you enter the Laboratory.
2. All the students should sit according to their roll numbers starting from their left to right.
3. All the students are supposed to enter the terminal number in the log book.
4. Do not change the terminal on which you are working.
5. All the students are expected to get at least the algorithm of the program/concept to be implemented.
6. Strictly observe the instructions given by the teacher/Lab Instructor.
7. Do not disturb machine Hardware / Software Setup.

## Instruction for Laboratory Teachers:

1. Submission related to whatever lab work has been completed should be done during the next lab session along with signing the index.
2. The promptness of submission should be encouraged by way of marking and evaluation patterns that will benefit the sincere students.
3. Continuous assessment in the prescribed format must be followed.

# Practical 1

**Aim** : Study about input output statements in Python Programming language.

**Tools / Equipments** : Python, Jupyter Notebook or Google Colab, desktop or laptop computer.

**Objective:**

| Sr.No | Objectives | LO | BL |
|---|---|---|---|
| 1 | Learn a basics and structure of python programming language. | 1 | L2 |
| 2 | Interpret good profound knowledge in python programming languageand enable them to write python program, compile and execute it. | 1 | L2,L3 |
| 3 | Able to solve error found in program /error solving | 1 | L3 |

## 1.1 Description

### 1.1.1. Output Operation:

We use the print() function to output data to the standard output device (screen).

```
print('This sentence is output to the screen')
```

**Output**

This sentence is output to the screen

Another example is given below:

```
a = 5
print('The value of a is', a)
```

**Output**

The value of a is 5

### 1.1.2. Taking input in Python

**input ():** This function first takes the input from the user and converts it into a string. The type of the returned object always will be <type 'str'>. It does not evaluate the expression it just returns the complete statement as String. For example, Python provides a built-in function called input which takes the input from the user. When the input function is called it stops the program and waits for the user's input. When the user presses enter, the program resumes and returns what the user typed.

**Syntax:**

```
inp = input('STATEMENT')
```

Example:

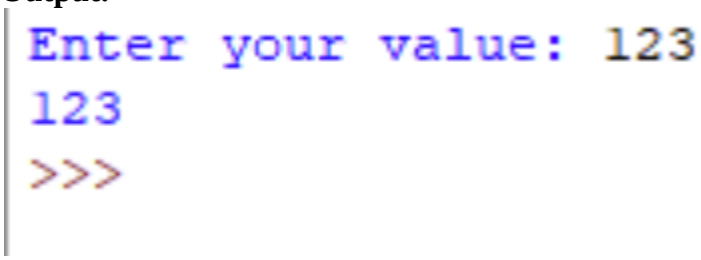
```
>>> name = input('What is your name?\n')     # \n ---> newline  ---> It causes a line break
```

```
>>> What is your name?
 Ram
>>> print(name)
 Ram
```

```
# Python program showing  a use of input()

val = input("Enter your value: ")
print(val)
```

**Output:**




**Taking String as an input:**

```
name = input('What is your name?)
print(name)
```

**Output:**

```
What is your name?
Ram
Ram
```

**How the input function works in Python :**

- When input() function executes program flow will be stopped until the user has given input.
- The text or message displayed on the output screen to ask a user to enter an input value is optional i.e. the prompt, which will be printed on the screen is optional.
- Whatever you enter as input, the input function converts it into a string. if you enter an integer value still input() function converts it into a string. You need to explicitly convert it into an integer in your code using typecasting.

Example:

```
>>> num = input('Enter a number: ')
 Enter a number: 10
>>> num
'10‘
```

- Here, we can see that the entered value 10 is a string, not a number. To convert this into a number we can use int() or float() functions.

```
>>> int('10')
10
 >>> float('10')
```

10.0

- **input() always returns a string. If you want a numeric type, then you need to <u>convert the string</u> to the appropriate type with the int(), float(), or complex() built-in functions:**

```
>>> n = input('Enter a number: ')
Enter a number: 50
>>> print(n + 100)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: must be str, not int

>>> n = int(input('Enter a number: '))
Enter a number: 50
 >>> print(n + 100)
150
```

### 1.2 Programming exercise

| Sr. No | Problem Statement | Level |
|---|---|---|
| 1 | Program to print student bio-data | [Low] |
| 2 | Write a program to take input (int, float, string-your name ) and print it. | [Low] |

**Conclusion :**

This experiment is focused on input and output statement in python programming language.

## Practical - 2

**Aim :** Programs based on operators and evaluation of expressions.

**Objective :**

| Sr.No | Objectives | LO | BL |
|---|---|---|---|
| 1 | To be familiar with different data types, Operators and Expressions in Python | 2 | L2 |
| 2 | Recognize and understand the syntax and construction of Python operator. | 2 | L2 |
| 3 | Develop a program for real life examples using operator. | 2 | L3 |

**2.1 Description :**

**Python Operators**

The operator can be defined as a symbol which is responsible for a particular operation between two operands. Operators are the pillars of a program on which the logic is built in a specific programming language. Python provides a variety of operators, which are described as follows.

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

**Arithmetic Operators**

Arithmetic operators are used to perform arithmetic operations between two operands. It includes + (addition), - (subtraction), *(multiplication), /(divide), %(reminder), //(floor division), and exponent (**) operators.

| Operator | Description |
|---|---|
| + (Addition) | It is used to add two operands. For example, if a = 20, b = 10 => a+b = 30 |
| - (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 = 10 => a - b = 10 |
| / (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 = 10 => a * b = 200 |
| % (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) | It is an exponent operator represented as it calculates the first operand power to the second operand. |
| // (Floor division) | It gives the floor value of the quotient produced by dividing the two operands. |

**Comparison operator**

Comparison operators are used to comparing the value of the two operands and returns Boolean true or false accordingly. The comparison operators are described in the following table.

| 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. |
| <= | If the first operand is less than or equal to the second operand, then the condition becomes true. |
| >= | If the first operand is greater than or equal to the second operand, then the condition becomes true. |
| > | 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. |

**Assignment Operators**

The assignment operators are used to assign the value of the right expression to the left operand. The assignment operators are described in the following table.

| Operator | Description |
|---|---|
| = | It assigns the value of the right expression to the left operand. |
| += | It increases 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 = 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. |
| *= | It multiplies the value of the left operand by the value of the right operand and assigns the modified value back to then the left operand. For example, if a = 10, b = 20 => a* = b will be equal to a = a* b and therefore, a = 200. |
| %= | It divides the value of the left operand by the value of the right operand and assigns the reminder back to the left operand. For example, if a = 20, b = 10 => a % = b will be equal to a = a % b and therefore, a = 0. |
| **= | a**=b will be equal to a=a**b, for example, if a = 4, b =2, a**=b will assign 4**2 = 16 to a. |
| //= | A//=b will be equal to a = a// b, for example, if a = 4, b = 3, a//=b will assign 4//3 = 1 to a. |

**Bitwise Operators**

The bitwise operators perform bit by bit operation on the values of the two operands. Consider the following example.

For example,

if a = 7

b = 6

then, binary (a) = 0111

binary (b) = 0110

hence, a & b = 0011

a | b = 0111

a ^ b = 0100

~ a = 1000

| Operator | Description |
|---|---|
| & (binary and) | If both the bits at the same place in two operands are 1, then 1 is copied to the result. Otherwise, 0 is copied. |
| \| (binary or) | The resulting bit will be 0 if both the bits are zero; otherwise, the resulting bit will be 1. |
| ^ (binary | The resulting bit will be 1 if both the bits are different; otherwise, the |

| | |
|---|---|
| xor) | resulting bit will be 0. |
| ~ (negation) | It calculates the negation of each bit of the operand, i.e., if the bit is 0, the resulting bit will be 1 and vice versa. |
| << (left shift) | The left operand value is moved left by the number of bits present in the right operand. |
| >> (right shift) | The left operand is moved right by the number of bits present in the right operand. |

**Logical Operators**

The logical operators are used primarily in the expression evaluation to make a decision. Python supports the following logical operators.

| Operator | Description |
|---|---|
| And | If both the expression are true, then the condition will be true. If a and b are the two expressions, a → true, b → true => a and b → true. |
| Or | If one of the expressions is true, then the condition will be true. If a and b are the two expressions, a → true, b → false => a or b → true. |
| Not | If an expression a is true, then not (a) will be false and vice versa. |

**Membership Operators**

Python membership operators are used to check the membership of value inside a Python data structure. If the value is present in the data structure, then the resulting value is true otherwise it returns false.

| Operator | Description |
|---|---|
| In | It is evaluated to be true if the first operand is found in the second operand (list, tuple, or dictionary). |
| not in | It is evaluated to be true if the first operand is not found in the second operand (list, tuple, or dictionary). |

**Identity Operators**

The identity operators are used to decide whether an element certain class or type.

| Operator | Description |
|---|---|
| Is | It is evaluated to be true if the reference present at both sides point to the same object. |

| | |
|---|---|
| is not | It is evaluated to be true if the reference present at both sides do not point to the same object. |

### 2.2. Programming Exercise

| Sr. No | Problem Statement | Level |
|---|---|---|
| 1 | Write a program to simulate a simple calculator (+ - / * %) that takes two operands as input and displays the Result. | [Low] |
| 2 | Write a program to find area and perimeter of geometric objects. | [Medium] |
| 3 | The distance between two cities (in km.) is input through the keyboard. Write a program to convert and print this distance in meters, feet, inches and centimeters. | [Medium] |
| 4 | Write a Program to interchange two numbers. | [Medium] |
| 5 | Write a program to compute Fahrenheit from centigrade (f=1.8*c +32) | [Low] |

Conclusion :

This experiment implements, the focused objective(s) of python operators and evaluation of expressions.

# Practical 3

**Aim**: Programs based on Decision Statements to understand the programming knowledge.

**Objective:**

| Sr. No | Objectives | LO | BL |
|---|---|---|---|
| 1 | Understand the Basic Concepts and syntax of Decision statements ( if, if else, nested if else ,else if ladder etc) | 2 | L2 |
| 2 | Develop program for real life examples using Conditional branching | 2 | L3 |

## 3.1. Description:

**Python Conditions and If statements:**

Python supports the usual logical conditions from mathematics:

- Equals: a == b
- Not Equals: a != b
- Less than: a < b
- Less than or equal to: a <= b
- Greater than: a > b
- Greater than or equal to: a >= b

These conditions can be used in several ways, most commonly in "if statements" and loops.
An "if statement" is written by using the if keyword.
Example
**If statement:**

```
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

In this example we use two variables, a and b, which are used as part of the if statement to test whether b is greater than a. As a is 33, and b is 200, we know that 200 is greater than 33, and so we print to screen that "b is greater than a".

**Indentation**
Python relies on indentation (whitespace at the beginning of a line) to define scope in the code. Other

programming languages often use curly-brackets for this purpose.
Example
If statement, without indentation (will raise an error):

```
a = 33
b = 200
if b > a:
print("b is greater than a") # you will get an error
```

---

**Elif**

The elif keyword is pythons way of saying "if the previous conditions were not true, then try this condition".
Example

```
a = 33
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
```

In this example a is equal to b, so the first condition is not true, but the elif condition is true, so we print to screen that "a and b are equal".

---

**Else**

The else keyword catches anything which isn't caught by the preceding conditions.
Example

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

In this example a is greater than b, so the first condition is not true, also the elif condition is not true, so we go to the else condition and print to screen that "a is greater than b".
You can also have an else without the elif:
Example

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
else:
  print("b is not greater than a")
```

### 3.2. Programming Exercise

| Sr. No | Problem Statement | Level |
|---|---|---|
| **1** | Write a program to read marks for five subjects as an input from keyboard and also calculate percentage to display equivalent grade according to following conditions (else-if ladder and logical operator)<br>**Marks** **Grade**<br>100 - 80 Distinction<br>79 - 60 First Class<br>59 - 40 Second Class<br>< 40 Fail | [High] |
| **2** | Write a program to calculate overtime pay of employee. Overtime is paid at the rate of Rs. 12.00 per hour for every hour worked above 40 hours. Assume that employee do not work for fractional part of an hour. | [Medium] |
| **3** | If the ages of three brothers are input through the keyboard, write a Program to determine the youngest and oldest of the three | [Medium] |
| **4** | Write a c program to prepare pay slip using the following data.<br>· If the Basic Salary is less than or equal to 10000 then HRA = 8% of the basic, and DA = 10% of the basic<br>· Basic Salary is less than or equal to 20000 then HRA = 16% and DA = 20%<br>· Basic Salary is greater than 20000 then HRA = 24% and DA = 30%<br>Gross = basic + Da + Hra | [High] |

| | | |
|---|---|---|
| **5** | Write a program for checking the speed of drivers.<br>If speed is less than 70, it should print "Ok". Otherwise, for every 5km above the speed limit (70), it should give the driver one demerit point and print the total number of demerit points. For example, if the speed is 80, it should print: "Points: 2". If the driver gets more than 12 points, the function should print: "License suspended" | [High] |

Conclusion :

This experiment implements decision making statements

# Practical 4

**Aim**: Development of programs in python using loops (while, for)

**Objective:**

| Sr. No | Objectives | LO | BL |
|---|---|---|---|
| 1 | Understand the Syntax and Iteration of Loops( while and for) | 2 | L2 |
| 2 | Understand the Difference Between while and for loops | 2 | L2 |
| 3 | Develop program for real life examples using loops ,break and continue statement | 2 | L3 |

**4.1.Description:**

**Looping statements in Python**

Looping simplifies complicated problems into smooth ones. It allows programmers to modify the flow of the program so that rather than writing the same code, again and again, programmers are able to repeat the code a finite number of times.

In Python, there are three different types of loops: for loop, while loop, and nested loop.

**For Loop**

The for loop is used in the case where a programmer needs to execute a part of the code until the given condition is satisfied. The for loop is also called a pre-tested loop. It is best to use for loop if the number of iterations is known in advance.

***Syntax:***

```
for variable in sequence:
    statements(s)
```

```
a = 5
for i in range(0, a):
    print(i)
```

```
0
1
2
3
4
```

The for loop runs till the value of i is less than a. As the value of i is 5, the loop ends.

**For loop Using range() function**

The range() function is used to generate the sequence of the numbers,starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

If we pass the range(10), it will generate the numbers from 0 to 9. The syntax of the range() function is given below.

Syntax:

**range(start,stop,step size)**

- The start represents the beginning of the iteration.
- The stop represents that the loop will iterate till stop-1. The range(1,5) will generate numbers 1 to 4 iterations. It is optional.
- The step size is used to skip the specific numbers from the iteration. It is optional to use. By default, the step size is 1. It is optional.

Example:

```
print(range(10))
print(list(range(10)))
print(list(range(2, 8)))
print(list(range(2, 20, 3)))
```

**Output**

```
range(0, 10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
 [2, 3, 4, 5, 6, 7]
[2, 5, 8, 11, 14, 17]
```

**While Loop**

The while loop is to be used in situations where the number of iterations is unknown at first. The block of statements is executed in the while loop until the condition specified in the while loop is satisfied. It is also called a pre-tested loop.

In Python, the while loop executes the statement or group of statements repeatedly while the given condition is True. And when the condition becomes false, the loop ends and moves to the next statement after the loop.

***Syntax:***

```
While condition:

    statement(s)
```

***Input:***

```
count = 0
while (count < 5):
   count = count + 1
   print("Hello")
```

```
Hello

Hello

Hello

Hello

Hello
```

The loop prints ‘Hello’ till the value of count becomes 5 and the condition is False.

**Nested Looping statements in Python**

The Python programming language allows programmers to use one looping statement inside another looping statement.

***Syntax:***

```
#for loop statement
for variable in sequence:
    for variable in sequence:
        statement(s)
        statement(s)

#while loop statement
while condition:
    while condition:
        statement(s)
        statement(s)
```

```
for i in range(1, 7):
    for j in range(i):
        print(i, end=' ')
    print()
```

```
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
6 6 6 6 6 6
```

**4.2. Exercise:**

| Sr. No | Problem Statement | Level |
|---|---|---|
| 1 | WAP to find factorial of given number | [Low] |
| 2 | WAP to check whether given number is Palindrome or not | [Medium] |
| 3 | WAP to check whether given number is Armstrong or not | [Medium] |
| 4 | WAP to print Fibonacci series | [Medium] |
| 5 | Write a Python program which iterates the integers from 1 to 50. For multiples of three print "Fizz" instead of the number and for the multiples of five print "Buzz". For numbers which are multiples of both three and five print "FizzBuzz". | [High] |
| 6 | WAP to check whether given number is Perfect number or not | [High] |
| 7 | WAP to check whether given number is Prime number or not | [High] |
| 8 | WAP to print following patterns<br>*  * * * * *<br>* *  * * * *<br>* * *  * * *<br>* * * *  * *<br>* * * * *  *<br>1  1<br>1 2  2 2<br>1 2 3  3 3 3<br>1 2 3 4  4 4 4 4<br>1 2 3 4 5  5 5 5 5 5 | [High] |

Conclusion :

This experiment implements looping statements

# Practical 5

**Aim**: Implementation of Python programs using string

**Objective:**

| Sr. No | Objectives | LO | BL |
|---|---|---|---|
| 1 | Understand the Syntax of string | 3 | L2 |
| 2 | Make use of inbuilt functions of string | 3 | L3 |
| 3 | Develop program using string | 3 | L3 |

**5.1.Description:**

Strings are amongst the most popular types in Python. We can create them simply by enclosing characters in quotes. Python treats single quotes the same as double quotes. Creating strings is as simple as assigning a value to a variable. For example −

```
var1 = 'Hello World!'
var2 = "Python Programming"
```

Accessing Values in Strings

Python does not support a character type; these are treated as strings of length one, thus also considered a substring.

To access substrings, use the square brackets for slicing along with the index or indices to obtain your substring. For example −

```
#!/usr/bin/python

var1 = 'Hello World!'
var2 = "Python Programming"

print "var1[0]: ", var1[0]
print "var2[1:5]: ", var2[1:5]
```

When the above code is executed, it produces the following result −

```
var1[0]:  H
var2[1:5]:  ytho
```

## Built-in String Methods

Python includes the following built-in methods to manipulate strings −

| Sr.No. | Methods with Description |
|---|---|
| 1 | capitalize()<br>Capitalizes first letter of string |
| 2 | center(width, fillchar)<br>Returns a space-padded string with the original string centered to a total of width columns. |
| 3 | count(str, beg= 0,end=len(string))<br>Counts how many times str occurs in string or in a substring of string if starting index beg and ending index end are given. |
| 4 | decode(encoding='UTF-8',errors='strict')<br>Decodes the string using the codec registered for encoding. encoding defaults to the default string encoding. |
| 5 | encode(encoding='UTF-8',errors='strict')<br>Returns encoded string version of string; on error, default is to raise a ValueError unless errors is given with 'ignore' or 'replace'. |
| 6 | endswith(suffix, beg=0, end=len(string))<br>Determines if string or a substring of string (if starting index beg and ending index end are given) ends with suffix; returns true if so and false otherwise. |
| 7 | expandtabs(tabsize=8)<br>Expands tabs in string to multiple spaces; defaults to 8 spaces per tab if tabsize not provided. |
| 8 | find(str, beg=0 end=len(string))<br>Determine if str occurs in string or in a substring of string if starting index beg and ending index end are given returns index if found and -1 otherwise. |
| 9 | index(str, beg=0, end=len(string))<br>Same as find(), but raises an exception if str not found. |
| 10 | isalnum()<br>Returns true if string has at least 1 character and all characters are alphanumeric and |

| | |
|---|---|
| | false otherwise. |
| 11 | isalpha()<br>Returns true if string has at least 1 character and all characters are alphabetic and false otherwise. |
| 12 | isdigit()<br>Returns true if string contains only digits and false otherwise. |
| 13 | islower()<br>Returns true if string has at least 1 cased character and all cased characters are in lowercase and false otherwise. |
| 14 | isnumeric()<br>Returns true if a unicode string contains only numeric characters and false otherwise. |
| 15 | isspace()<br>Returns true if string contains only whitespace characters and false otherwise. |
| 16 | istitle()<br>Returns true if string is properly "titlecased" and false otherwise. |
| 17 | isupper()<br>Returns true if string has at least one cased character and all cased characters are in uppercase and false otherwise. |
| 18 | join(seq)<br>Merges (concatenates) the string representations of elements in sequence seq into a string, with separator string. |
| 19 | len(string)<br>Returns the length of the string |
| 20 | ljust(width[, fillchar])<br>Returns a space-padded string with the original string left-justified to a total of width columns. |

| | |
|---|---|
| 21 | lower()<br>Converts all uppercase letters in string to lowercase. |
| 22 | lstrip()<br>Removes all leading whitespace in string. |
| 23 | maketrans()<br>Returns a translation table to be used in translate function. |
| 24 | max(str)<br>Returns the max alphabetical character from the string str. |
| 25 | min(str)<br>Returns the min alphabetical character from the string str. |
| 26 | replace(old, new [, max])<br>Replaces all occurrences of old in string with new or at most max occurrences if max given. |
| 27 | rfind(str, beg=0,end=len(string))<br>Same as find(), but search backwards in string. |
| 28 | rindex( str, beg=0, end=len(string))<br>Same as index(), but search backwards in string. |
| 29 | rjust(width,[, fillchar])<br>Returns a space-padded string with the original string right-justified to a total of width columns. |
| 30 | rstrip()<br>Removes all trailing whitespace of string. |
| 31 | split(str="", num=string.count(str))<br>Splits string according to delimiter str (space if not provided) and returns list of substrings; split into at most num substrings if given. |
| 32 | splitlines( num=string.count('\n')) |

| | |
|---|---|
| | Splits string at all (or num) NEWLINEs and returns a list of each line with NEWLINEs removed. |
| 33 | startswith(str, beg=0,end=len(string))<br>Determines if string or a substring of string (if starting index beg and ending index end are given) starts with substring str; returns true if so and false otherwise. |
| 34 | strip([chars])<br>Performs both lstrip() and rstrip() on string. |
| 35 | swapcase()<br>Inverts case for all letters in string. |
| 36 | title()<br>Returns "titlecased" version of string, that is, all words begin with uppercase and the rest are lowercase. |
| 37 | translate(table, deletechars="")<br>Translates string according to translation table str(256 chars), removing those in the del string. |
| 38 | upper()<br>Converts lowercase letters in string to uppercase. |
| 39 | zfill (width)<br>Returns original string leftpadded with zeros to a total of width characters; intended for numbers, zfill() retains any sign given (less one zero). |
| 40 | isdecimal()<br>Returns true if a unicode string contains only decimal characters and false otherwise. |

**5.2.Programming Exercise:**

| Sr. No | Problem Statement | Level |
|---|---|---|
| 1 | Write Python Program to find length of string without using len() | [Medium] |

| | | |
|---|---|---|
| | function. | |
| 2 | Count all letters, digits, and special symbols from a given string. | [Medium] |
| 3 | Python Program to Count the Number of Vowels in a String. | [Medium] |
| 4 | Python Program to Calculate the Number of Upper Case Letters and Lower Case Letters in a String. | [Medium] |
| 5 | Python Program to Check whether given string is palindrome or not | [Medium] |

Conclusion :

This experiment implements programs using string

# Practical 6

**Aim**: **Programs using List and Tuple**

**Objective:**

| Sr.No | Objectives | LO | BL |
|---|---|---|---|
| 1 | Understand Data types in Python | 3 | L3 |
| 2 | Able to create list and tuple. | 3 | L3,L4 |
| 3 | Make use of list and tuple to s olve the real-life examples | 3 | L3,L4 |

**6.1. Description:**

**Python List:**

- List is an ordered sequence of items
- It is one of the most used data type in Python and is very flexible
- All the items in a list do not need to be of the same type.
- In list items separated by commas are enclosed within brackets [ ]
- The index starts from 0 in Python

  a = [1, 2.2, 'python']

- If you add new items to a list, the new items will be placed at the end of the list
- Lists are mutable, meaning, the value of elements of a list can be altered
- We can use the slicing operator [ ] to extract an item or a range of items from a list

```
a = [5,10,15,20,25,30,35,40]

# a[2] = 15
print("a[2] = ", a[2])

# a[0:3] = [5, 10, 15]
print("a[0:3] = ", a[0:3])

# a[5:] = [30, 35, 40]
print("a[5:] = ", a[5:])
```

**Tuple**

- A tuple in Python is similar to a list
- The difference between the two is that we cannot change the elements of a tuple once it is assigned whereas we can change the elements of a list
- A tuple is created by placing all the items (elements) inside parentheses (), separated by commas, the parentheses are optional
- To create a tuple with only one item, you have to add a comma after the item, otherwise Python will not recognize it as a tuple
- A tuple can have any number of items and they may be of different types (integer, float, list, string, etc.)

```
# Empty tuple
my_tuple = ()
print(my_tuple)

# Tuple having integers
my_tuple = (1, 2, 3)
print(my_tuple)

# Tuple having String
my_tuple = ("apple", "banana", "cherry")
print(my_tuple)

#tuple with one element
my_tuple=("apple",)

# Not a tuple
my_tuple=("apple")

# tuple with mixed datatypes
my_tuple = (1, "Hello", 3.4)
print(my_tuple)

# nested tuple
my_tuple = ("mouse", [8, 4, 6], (1, 2, 3))
print(my_tuple)

print(len(my_tuple))
```

## 6.2. Programming Exercise

| Sr. No | Problem Statement | Level |
|---|---|---|
| 1 | Write a Python program to sum all the items in a list. | [Medium] |
| **2** | Write a Python program to multiply all the items in a list | [Medium] |
| **3** | Write a Python program to get the largest number from a list. | [Medium] |
| **4** | Write a Python program to get the smallest number from a list | [Medium] |

| | | |
|---|---|---|
| **5** | Write a Python program to count all elements in list and count Occurrences Of A List Item In Python | [Medium] |
| **6** | Write a Python program to create a tuple with different data types | [Medium] |
| **7** | Write a Python program to check whether an element exists within a tuple | [Medium] |
| **8** | Write a Python program to reverse a tuple | [Medium] |
| **9** | Write a Python program calculate the product of all the numbers given in tuple. Original Tuple: (2, 4, 8, 8, 3, 2, 9)<br>Product - multiplying all the numbers of the said tuple: 27648 | [High] |

Conclusion :

This experiment implements program using list and tuple

# Practical 7

**Aim**: **Programs using Dictionary and Set**

**Objective:**

| Sr.No | Objectives | LO | BL |
|---|---|---|---|
| 1 | Understand Data types in Python | 3 | L3 |
| 2 | Able to create Dictionary and set. | 3 | L3,L4 |
| 3 | Make use of list and tuple to s olve the real-life examples | 3 | L3,L4 |

**6.1. Description:**

Python Dictionary

- Python dictionary is an unordered collection of items
- Each item of a dictionary has a key/value pair
- An item has a key and a corresponding value that is expressed as a pair (*key: value*)
- Dictionaries are optimized to retrieve values when the *key* is known
- we can also create a dictionary using the built-in *dict()* function
- Accessing Elements from Dictionary
- Keys can be used either inside square brackets [] or with the get() method
- In Brackets KeyError is raised in case a key is not found in the dictionary

```
# empty dictionary
my_dict = {}

# dictionary with integer keys
my_dict = {1: 'apple', 2: 'ball'}

# dictionary with mixed keys
my_dict = {'name': 'John', 1: [2, 4, 3]}

# using dict()
my_dict = dict({1:'apple', 2:'ball'})

# get and [] for retrieving elements
my_dict = {'name': 'Jack', 'age': 26}

# Output: Jack
print(my_dict['name'])

# Output: 26
print(my_dict.get('age'))
```

**Python Set**

- A set is created by placing all the items (elements) inside curly braces {}, separated by comma, or by using the built-in set() function
- It can have any number of items and they may be of different types (integer, float, tuple, string etc.). But a set cannot have mutable elements like lists, sets or dictionaries as its elements
- Once a set is created, you cannot change its items, but you can add new items
- Sets are unordered, so you cannot be sure in which order the items will appear
- Duplicates are not allow in set
- Empty curly braces {} will make an empty dictionary in Python. To make a set without any elements, we use the set() function without any argument

```
# Different types of sets in Python
# set of integers
my_set = {1, 2, 3}
print(my_set)

# set of mixed datatypes
my_set = {1.0, "Hello", (1, 2, 3)}
print(my_set)

# note the double round-brackets
thisset = set(("apple", "banana", "cherry"))
print(thisset)

# set cannot have duplicates
# Output: {1, 2, 3, 4}
my_set = {1, 2, 3, 4, 3, 2}
print(my_set)

# set cannot have mutable items
# here [3, 4] is a mutable list
# this will cause an error.
my_set = {1, 2, [3, 4]}
```

### 6.2. Programming Exercise

| Sr. No | Problem Statement | Level |
|---|---|---|
| 1 | Write a Python program to concatenate following dictionaries to create a new one. Sample Dictionary : dic1={1:10, 2:20} dic2={3:30, 4:40} dic3={5:50,6:60}<br>Expected Result : {1: 10, 2: 20, 3: 30, 4: 40, 5: 50, 6: 60} | [Medium] |
| **2** | Write a Python program to check whether a given key already exists in a dictionary | [Medium] |
| **3** | Write a Python script to generate and print a dictionary that contains a number (between 1 and n) in the form (x, x*x)<br>Sample Dictionary ( n = 5) : | [Medium] |

| | | |
|---|---|---|
| | Expected Output : {1: 1, 2: 4, 3: 9, 4: 16, 5: 25} | |
| **4** | Write a Python program to merge two Python dictionaries | [Medium] |
| **5** | Write a Python program to get the maximum and minimum value in a dictionary | [Medium] |
| **6** | Write a Python program to create set difference, Union and intersection | [Medium] |
| **7** | Write a Python program to check if two given sets have no elements in common | [Medium] |

Conclusion :

This experiment implements program using dictionary and set.

# Practical 8

**Aim**: **Programs using Function**

**Objective:**

| Sr. No | Objectives | LO | BL |
|---|---|---|---|
| 1 | Understand the Syntax and types of functions | 4 | L2 |
| 2 | Develop modular programs using function, parameters with using types of variables (local and global) | 4 | L3 |
| 3 | Develop program for Mathematical recursion and Iterations | 4 | L3 |

**8.1 Description:**

**What are Python Functions?**

A function is a collection of related assertions that performs a mathematical, analytical, or evaluative operation. Python functions are simple to define and essential to intermediate-level programming. The exact criteria hold to function names as they do to variable names. The goal is to group up certain often performed actions and define a function. Rather than rewriting the same code block over and over for varied input variables, we may call the function and repurpose the code included within it with different variables.

The functions are broad of two types, user-defined and built-in functions. It aids in keeping the software succinct, non-repetitive, and well-organized.

Advantages of Functions in Python

Python functions have the following benefits.

- By including functions, we can prevent repeating the same code block repeatedly in a program.
- Python functions, once defined, can be called many times and from anywhere in a program.
- If our Python program is large, it can be separated into numerous functions which is simple to track.
- The key accomplishment of Python functions is we can return as many outputs as we want with different arguments.

However, calling functions has always been overhead in a Python program.

Syntax of Python Function

**Code**

```
def name_of_function( parameters ):
    """This is a docstring"""
    # code block
```

The following elements make up define a function, as seen above.

- The beginning of a function header is indicated by a keyword called def.
- name_of_function is the function's name that we can use to separate it from others. We will use this name to call the function later in the program. The same criteria apply to naming functions as to naming variables in Python.

- We pass arguments to the defined function using parameters. They are optional, though.
- The function header is terminated by a colon (:).
- We can use a documentation string called docstring in the short form to explain the purpose of the function.
- The body of the function is made up of several valid Python statements. The indentation depth of the whole code block must be the same (usually 4 spaces).
- We can use a return expression to return a value from a defined function.

**Example of a User-Defined Function**

We will define a function that when called will return the square of the number passed to it as an argument.

**Code**

```
def square( num ):
    """
    This function computes the square of the number.
    """
    return num**2
object_ = square(9)
print( "The square of the number is: ", object_ )
```

**Output:**

```
The square of the number is:  81
```

**Calling a Function**

A function is defined by using the def keyword and giving it a name, specifying the arguments that must be passed to the function, and structuring the code block.

After a function's fundamental framework is complete, we can call it from anywhere in the program. The following is an example of how to use the a_function function.

**Code**

```
# Defining a function
def a_function( string ):
    "This prints the value of length of string"
    return len(string)

# Calling the function we defined
print( "Length of the string Functions is: ", a_function( "Functions" ) )
print( "Length of the string Python is: ", a_function( "Python" ) )
```

**Output:**

```
Length of the string Functions is:  9
Length of the string Python is:  6
```

**8.2 Programming Exercise:**

| Sr. No | Problem Statement | Level |
|---|---|---|
| 1 | Write Functions to calculate your trip's costs:<br>Define a function called hotel_cost with one argument nights as input<br>Define a function called plane_ride_cost that takes a string, city, as input.<br>Define a function called rental_car_cost with an argument called days.<br>Define a function called trip_cost that takes two arguments, city and days. Like the example above, have your function return the sum of calling the rental_car_cost(days), hotel_cost(days), and plane_ride_cost(city) functions. | [High] |
| 2 | Write a program in to check a given number is even or odd using the function. | [Medium] |
| 3 | Write a function Exchange to interchange the values of two variables, say x and y. illustrate the use of this function in a calling function. | [Medium] |
| 4 | Write a program to find Sum of natural number using recursion. | [Medium] |
| 5 | Write a program to print Fibonacci series number using recursion | [Medium] |

\

Conclusion :

This experiment implements program using function.

# Practical 9

**Aim: Program using NumPy, Matplotlib and Pandas library**

**Objective:**

| Sr.No | Objectives | LO | BL |
|---|---|---|---|
| 1 | Develop a program using NumPy library | 5 | L3 |
| 2 | Plot different types of graphs using Matplotlib | 5 | L3 |
| 3 | Develop a program using Pandas library | 5 | L3 |

**9.1 Description :**

**NumPy**

NumPy is a Python package. It stands for 'Numerical Python'. It is a library consisting of multidimensional array objects and a collection of routines for processing of array.

Numeric, the ancestor of NumPy, was developed by Jim Hugunin. Another package Numarray was also developed, having some additional functionalities. In 2005, Travis Oliphant created NumPy package by incorporating the features of Numarray into Numeric package. There are many contributors to this open source project.

Using NumPy, a developer can perform the following operations −

- Mathematical and logical operations on arrays.
- Fourier transforms and routines for shape manipulation.
- Operations related to linear algebra. NumPy has in-built functions for linear algebra and random number generation.

**1-Dimensional Array:**

The array that has Zero Dimensional arrays as its elements is a uni-dimensional or 1-D array.

The code below creates a 1-D array,

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(arr)
```

**Output:**

```
[1 2 3 4 5]
```

**Two Dimensional Arrays:**

2-D Arrays are the ones that have 1-D arrays as its element. The following code will create a 2-D array with 1,2,3 and 4,5,6 as its values.

```
import numpy as np
arr1 = np.array([[1, 2, 3], [4, 5, 6]])
```

**print(arr1)**
**Output:**
**[[1 2 3]**
**[4 5 6]]**

**Three Dimensional Arrays:**
Let us see an example of creating a 3-D array with two 2-D arrays:
**import numpy as np**
**arr1 = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]]) print(arr1)**
**Output:**
**[[[1 2 3]**
**[4 5 6]]**
**[[1 2 3]**
**[4 5 6]]]**

To identify the dimensions of the array, we can use ndim as shown below:
**import numpy as np**
**a = np.array(36)**
**d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])**
**print(a.ndim)**
**print(d.ndim)**
**Output:**
**0**
**3**

## Matplotlib

Matplotlib is a python library used to create 2D graphs and plots by using python scripts. It has a module named pyplot which makes things easy for plotting by providing feature to control line styles, font properties, formatting axes etc. It supports a very wide variety of graphs and plots namely - histogram, bar charts, power spectra, error charts etc. It is used along with NumPy to provide an environment that is an effective open source alternative for MatLab. It can also be used with graphics toolkits like PyQt and wxPython.

**Pandas**

**Pandas** is built on top of the **Numpy** package, means **Numpy** is required for operating the Pandas. Before Pandas, Python was capable for data preparation, but it only provided limited support for data analysis. So, Pandas came into the picture and enhanced the capabilities of data analysis. It can perform five significant steps required for processing and analysis of data irrespective of the origin of the data, i.e., **load, manipulate, prepare, model, and analyze**.

**Key Features of Pandas**

- It has a fast and efficient DataFrame object with the default and customized indexing.
- Used for reshaping and pivoting of the data sets.
- Group by data for aggregations and transformations.
- It is used for data alignment and integration of the missing data.
- Provide the functionality of Time Series.

- Process a variety of data sets in different formats like matrix data, tabular heterogeneous, time series.
- Handle multiple operations of the data sets such as subsetting, slicing, filtering, groupBy, re-ordering, and re-shaping.
- It integrates with the other libraries such as SciPy, and scikit-learn.
- Provides fast performance, and If you want to speed it, even more, you can use the **Cython**.

**Benefits of Pandas**

The benefits of pandas over using other language are as follows:

- **Data Representation:** It represents the data in a form that is suited for data analysis through its DataFrame and Series.
- **Clear code:** The clear API of the Pandas allows you to focus on the core part of the code. So, it provides clear and concise code for the user.

Python Pandas Data Structure

The Pandas provides two data structures for processing the data, i.e., **Series** and **DataFrame**, which are discussed below:

## 1) Series

It is defined as a one-dimensional array that is capable of storing various data types. The row labels of series are called the **index**. We can easily convert the list, tuple, and dictionary into series using "series' method. A Series cannot contain multiple columns. It has one parameter:

**Data:** It can be any list, dictionary, or scalar value.

**Creating Series from Array:**

Before creating a Series, Firstly, we have to import the numpy module and then use array() function in the program.

```
import pandas as pd
import numpy as np
info = np.array(['P','a','n','d','a','s'])
a = pd.Series(info)
print(a)
```

**Output**

```
0   P
1   a
2   n
3   d
4   a
5   s
dtype: object
```

**Explanation:** In this code, firstly, we have imported the **pandas** and **numpy** library with the **pd** and **np** alias. Then, we have taken a variable named "info" that consist of an array of some values. We have called the **info** variable through a **Series** method and defined it in an "**a**" variable. The Series has printed by calling the **print(a)** method.

**Python Pandas DataFrame**

It is a widely used data structure of pandas and works with a two-dimensional array with labeled axes (rows and columns). DataFrame is defined as a standard way to store data and has two different indexes, i.e., row index and column index. It consists of the following properties:

- The columns can be heterogeneous types like int, bool, and so on.
- It can be seen as a dictionary of Series structure where both the rows and columns are indexed. It is denoted as "columns" in case of columns and "index" in case of rows.

**Create a DataFrame using List:**

We can easily create a DataFrame in Pandas using list.

```
import pandas as pd
# a list of strings
x = ['Python', 'Pandas']

# Calling DataFrame constructor on list
df = pd.DataFrame(x)
print(df)
```

**Output**

```
0   Python
1   Pandas
```

**Explanation:** In this code, we have defined a variable named "x" that consist of string values. The DataFrame constructor is being called on a list to print the values.

**9.2 Programming Exercise**

| Sr. No | Problem Statement | Level |
|---|---|---|
| **1** | Write a program to perform matrix addition, substraction, multiplication | [Medium] |
| **2** | Plot all types of graph using Matplotlib. | [Medium] |
| **3** | Write a program which perform basic operation of Pandas. | [High] |

Conclusion :

This experiment implements NumPy, Matplotlib and Pandas Library

# Practical 10

**Aim**: **Program on SQL Commands**

**Objective:**

| Sr.No | Objectives | LO | BL |
|---|---|---|---|
| 1 | To understand the basic concept of binary search and SQL | 6 | L2 |
| 2 | Execute SQL query like create, insert, update and delete | 6 | L3 |

**10.1 Description:**

**Python SQLite3** module is used to integrate the SQLite database with Python. It is a standardized Python DBI API 2.0 and provides a straightforward and simple-to-use interface for interacting with SQLite databases. There is no need to install this module separately as it comes along with Python after the 2.5x version.

**Connecting to SQLite Database**

- To use SQLite, we must import **sqlite3**.
  import sqlite3
- Then create a connection using connect() method and pass the name of the database you want to access if there is a file with that name, it will open that file. Otherwise, Python will create a file with the given name.
  sqliteConnection = sqlite3.connect('gfg.db')
- After this, a cursor object is called to be capable to send commands to the SQL.
- cursor = sqliteConnection.cursor()

C**ursor object**

- The cursor object is used to make the connection for executing SQL queries.
- It acts as middleware between SQLite database connection and SQL query. It is created after giving connection to SQLite database.
- The cursor is a control structure used to traverse and fetch the records of the database. All the commands will be executed using cursor object only.

In [4]:
```
import sqlite3

# connecting to the database
connection = sqlite3.connect("gfg.db")

# cursor
crsr = connection.cursor()

# print statement will execute if there
# are no errors
print("Connected to the database")

# close the connection
connection.close()
```

```
Connected to the database
```

**Creating Tables**

- After connecting to the database and creating the cursor object let's see how to execute the queries.
- To execute a query in the database, create an object and write the SQL command in it with being commented.

  Example:- sql_comm = "SQL statement"
- And executing the command is very easy. Call the cursor method execute() and pass the name of the sql command as a parameter in it. Save a number of commands as the sql_comm and execute them. After you perform all your activities, save the changes in the file by committing those changes and then lose the connection.

In [2]:
```
import sqlite3

# connecting to the database
connection = sqlite3.connect("gfg.db")

# cursor
crsr = connection.cursor()

# SQL command to create a table in the database
sql_command = """CREATE TABLE emp (staff_number INTEGER PRIMARY KEY,fname VARCHAR(20),
lname VARCHAR(30),gender CHAR(1),joining DATE);"""

# execute the statement
crsr.execute(sql_command)

# close the connection
connection.close()
```

### Inserting data

In [6]:
```
# Python code to demonstrate table creation and insertions with SQL
#importing module
import sqlite3
# connecting to the database
connection = sqlite3.connect("gfg.db")
# cursor
crsr = connection.cursor()
# SQL command to insert the data in the table
sql_command = """INSERT INTO emp VALUES (23, "Rishabh","Bansal", "M", "2014-03-28");"""
crsr.execute(sql_command)
# another SQL command to insert the data in the table
sql_command = """INSERT INTO emp VALUES (1, "Bill", "Gates","M", "1980-10-28");"""
crsr.execute(sql_command)
# To save the changes in the files. Never skip this.
# If we skip this, nothing will be saved in the database.
connection.commit()
# close the connection
connection.close()
```

### 10.2 Programming Exercise

| Sr. No | Problem Statement | Level |
|---|---|---|
| 1 | Write a program of binary search | [High] |
| 2 | Write a program which perform basic SQL commands | [High] |

**Conclusion:**

This experiment implements various query of SQL