## THADOMAL SHAHANI ENGINEERING COLLEGE

#### DEPARTMENT OF INFORMATION TECHNOLOGY

##### Control Structure: LO1

**Aim:** To execute control flow statements in Python

**Theory:** Python control structures allow you to control the flow of execution in your program using conditional statements (like if, elif, and else) and loops (such as for and while). These structures help in making decisions and repeating tasks based on certain conditions.

###### Program with output:

age = int(input("Enter your age: ")) if age >= 18:

print("You are eligible to vote.") elif age == 17:

print("You will be eligible to vote in a year.") else:

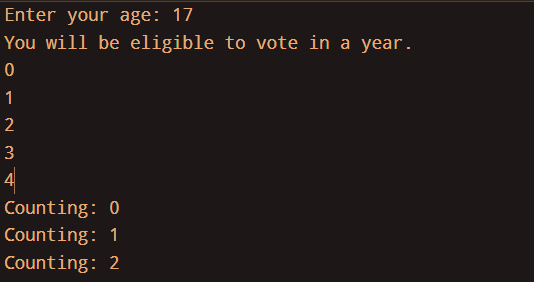
print("You are ineligible to vote.")

for i in range(5): print(i)

count = 0

while count < 3: print("Counting:", count) count += 1

###### Output:

****

**Conclusion:**

In short, Python control structures let your program make decisions and repeat tasks, helping it run smoothly and efficiently.

# THADOMAL SHAHANI ENGINEERING COLLEGE

## DEPARTMENT OF INFORMATION TECHNOLOGY

### Built in Datatypes: LO2

**Aim:** To implement built-in datatypes in Python

**Theory:** These data types include integers, floats, strings, lists, dictionaries, and more. Python provides these built-in data types to simplify the development process and to allow for efficient handling of data

##### Program with output:

**Lists**

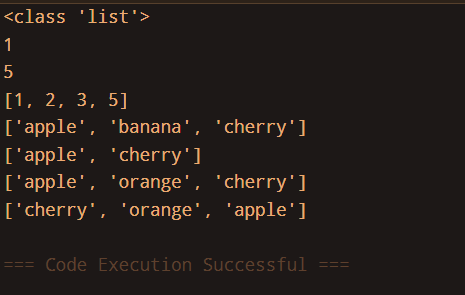
nums = [1, 3, 2, 5]

print(type(nums)) print(min(nums)) print(max(nums)) nums.sort() print(nums)

fruits = ["apple", "banana"] fruits.append("cherry") print(fruits) fruits.remove("banana")

print(fruits)

fruits.insert(1, "orange") print(fruits) fruits.reverse() print(fruits)



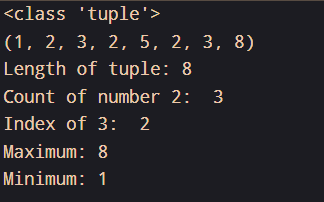
##### Tuples

tuples = (1,2,3,2,5,2,3,8)

print(type(tuples)) print(tuples)

print("Length of tuple:",len(tuples))

print("Count of number 2: ",tuples.count(2)) print("Index of 3: ",tuples.index(3)) print("Maximum:",max(tuples)) print("Minimum:",min(tuples))

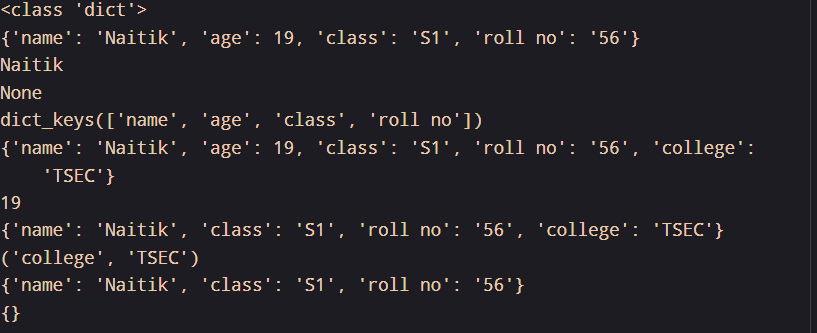


##### Dictionary

dict = {'name': 'Naitik', 'age': 19,'class': 'S1','roll no': '56'} print(type(dict))

print(dict) print(dict.get('name')) print(dict.get('address')) print(dict.keys()) dict.update({'college': 'TSEC'}) print(dict)

print(dict.pop('age')) print(dict) print(dict.popitem()) print(dict) dict.clear() print(dict)



##### Conclusion:

List and dictionaries are mutable datatypes while tuples are not, although tuples can be changed by converting to list and back to tuple after updaton.

List and tuples just require entries with index number as the unique identity automatically assigned to them while dictionary requires a key and its corresponding entries.

## THADOMAL SHAHANI ENGINEERING COLLEGE

#### DEPARTMENT OF INFORMATION TECHNOLOGY

##### Arrays: LO2

###### Aim: To implement arrays in Python

**Theory:** In Python, arrays can hold multiple values in a single variable, but the array's size is fixed. However, Python does not have a built-in array data structure like other languages (e.g., C or Java). Instead, Python provides the **list** type, which can be used similarly to an array in most cases.

###### Program with output:

*import array*

*arr = array.array('i', [1, 2, 3, 4, 5]) print(arr)*

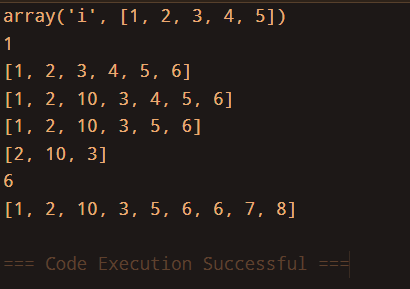
*arr = [1, 2, 3, 4, 5]*

*print(arr[0]) arr.append(6) print(arr) arr.insert(2, 10) print(arr) arr.remove(4) print(arr)*

*sub\_arr = arr[1:4] print(sub\_arr) print(len(arr))*

*arr2 = [6, 7, 8]*

*new\_arr = arr + arr2 print(new\_arr)* **OUTPUT:**



###### Conclusion:

The array module provides fixed-type arrays for more memory-efficient storage. Many update functions are possible in arrays.

## THADOMAL SHAHANI ENGINEERING COLLEGE

#### DEPARTMENT OF INFORMATION TECHNOLOGY

##### Functions : LO 1,2

###### Aim:Basic of Python Functions Program with output:

import numpy as np rows=int(input("no of rows"))

columns=int(input("no of columns"))

arr=np.empty((rows,columns),dtype=int)

for i in range(rows):

for j in range(columns): arr[i,j]=int(input("enter elements:"))

print("array is:") print(arr) print(arr.dtype) print(arr.size) print(arr.shape) print(arr.ndim) print(arr.itemsize) print(arr[:3])

print(arr[-3:])

print(arr[1:4:2])

a=10

S1-56

b=9

c="Hello world" print("value of a=",a) print("value of b=",b) print("/nArithmatic operator") print("Add a+b=",a+b) print("Sub a-b=",a-b) print("Multi a\*b=",a\*b) print("Div a/b=",a/b) print("")

import numpy as np

rows = int(input("Enter number of rows: ")) columns = int(input("Enter number of columns: ")) arr = np.empty((rows, columns), dtype=int)

for i in range(rows):

for j in range(columns):

arr[i, j] = int(input(f"Enter element at position ({i}, {j}): "))

print("\nArrayis:") print(arr)

print("\nArray Properties:") print(f"Datatype: {arr.dtype}") print(f"Size : {arr.size}") print(f"Shape : {arr.shape}")

print(f"Number of dimensions: {arr.ndim}") print(f"Item size: {arr.itemsize}")

# Demonstrating array slicing print("\nArray Slicing and Indexing:") print(f"First 3 rows:\n{arr[:3]}")

print(f"Last 3 rows:\n{arr[-3:]}")

print(f"Every second element from rows 1 to 4:\n{arr[1:4:2]}")

# Positive and Negative Ending if np.any(arr > 0):

print("\nThere are positive numbers in the array!") else:

print("\nThere are no positive numbers in the array.")

if np.any(arr < 0):

print("There are negative numbers in the array!") else:

print("There are no negative numbers in the array.")

import numpy as np

def demonstrate\_numpy\_functions():

rows = int(input("Enter the number of rows: ")) columns = int(input("Enter the number of columns: "))

# Create np.zeros() array

zeros\_array=np.zeros((rows, columns), dtype=int) print("\nArray filled with zeros:") print(zeros\_array)

# Create np.ones() array

ones\_array=np.ones((rows, columns), dtype=int) print("\nArray filled with ones:") print(ones\_array)

start = int(input("\nEnter the start value for np.arange: ")) stop = int(input("Enter the stop value for np.arange: ")) step = int(input("Enter the step value for np.arange: "))

range\_array = np.arange(start, stop, step).reshape(-1, columns) print("\nArray created with np.arange():")

print(range\_array)

print("\nProperties of the array filled with zeros:") print(f"Data type: {zeros\_array.dtype}")

print(f"Size : {zeros\_array.size}") print(f"Shape : {zeros\_array.shape}")

print(f"Item size : {zeros\_array.itemsize}")

# Checking for positive and negative numbers if np.any(zeros\_array > 0):

print("\nThere are positive numbers in the array filled with zeros!") else:

print("\nThere are no positive numbers in the array filled with zeros.")

if np.any(zeros\_array < 0):

print("There are negative numbers in the array filled with zeros!") else:

print("There are no negative numbers in the array filled with zeros.")

if np.any(range\_array > 0):

print("\nThere are positive numbers in the array created with np.arange()!")

else:

print("\nThere are no positive numbers in the array created with np.arange().")

if np.any(range\_array < 0):

print("There are negative numbers in the array created with np.arange()!") else:

print("There are no negative numbers in the array created with np.arange().")

demonstrate\_numpy\_functions()

import numpy as np

def array\_operations\_and\_manipulation():

rows = int(input("Enter the number of rows: ")) columns = int(input("Enter the number of columns: "))

arr = np.empty((rows, columns), dtype=int)

print("\nEnter elements for the array:") for i in range(rows):

for j in range(columns):

arr[i, j] = int(input(f"Element at position ({i+1}, {j+1}): "))

print("\nArray entered by the user:") print(arr)

print("\nArithmetic Operations:")

print(f"Addition (array + 2):\n{arr + 2}") print(f"Subtraction (array - 2):\n{arr - 2}") print(f"Multiplication (array \* 2):\n{arr \* 2}") print(f"Division (array / 2):\n{arr / 2}") print(f"Modulus (array % 2):\n{arr % 2}")

print(f"Exponentiation(array\*\* 2):\n{arr \*\* 2}") print(f"Floor Division (array // 2):\n{arr // 2}")

print("\nManipulation Functions:")

print(f"Absolute values (abs):\n{np.abs(arr)}")

print(f"Array rounded to nearest integers (round):\n{np.round(arr)}") print(f"Square root of each element (sqrt):\n{np.sqrt(np.abs(arr))}") print(f"Mean of array: {np.mean(arr)}")

print(f"Sum of array: {np.sum(arr)}")

print(f"Maximum value in array: {np.max(arr)}") print(f"Minimum value in array: {np.min(arr)}")

if np.any(arr > 0):

print("\nThere are positive numbers in the array!") else:

print("\nThere are no positive numbers in the array.")

if np.any(arr < 0):

print("There are negative numbers in the array!") else:

print("There are no negative numbers in the array.")

=================================== RESTART:

C:/Users/Lab1004/Desktop/Function.py ===================================

Enter the number of rows: 2 Enter the number of columns: 2

Array filled with zeros:

[[0 0]

[0 0]]

Array filled with ones:

[[1 1]

[1 1]]

Enter the start value for np.arange: -8 Enter the stop value for np.arange: 5 Enter the step value for np.arange: 7

Array created with np.arange():

[[-8 -1]]

Properties of the array filled with zeros:

Data type: int64

Size (number of elements): 4 Shape (rows, columns): (2, 2) Item size (in bytes): 8

There are no positive numbers in the array filled with zeros. There are no negative numbers in the array filled with zeros.

There are no positive numbers in the array created with np.arange(). There are negative numbers in the array created with np.arange()!

###### #array\_operations\_and\_manipulation( #Prime

def is\_prime(num): if num <= 1:

return False

for i in range(2, int(num \*\* 0.5) + 1): if num % i == 0:

return False return True

def main():

num = int(input("Enter a number to check if it's prime: ")) if is\_prime(num):

print(f"{num} is a prime number.") else:

print(f"{num} is not a prime number.")

if name == " main ": main()

'''

#Palindrome

def is\_palindrome(value): value = str(value)

return value == value[::-1]

def main():

value = input("Enter a string or number to check if it's a palindrome: ")

if is\_palindrome(value): print(f"'{value}' is a palindrome.")

else:

print(f"'{value}' is not a palindrome.")

if name == " main ": main()

===================================RESTART:C:/Users/Lab1004/Desktop/Funt on.py ===================================

Enter a number to check if it's prime: 56 56 is not a prime number.

Enter a string or number to check if it's a palindrome: 34 '34' is not a palindrome.

|  |  |  |
| --- | --- | --- |
|  | **THADOMAL SHAHANI ENGINEERING**  **COLLEGE** |  |
| **DEPARTMENT OF INFORMATION**  **TECHNOLOGY** |  |

1. **Classes: LO3 Aim:** To implement functioning of Classes in Python **Theory:**

###### Program with output:

class Student:

name = "Naitik" age = 19 roll\_no = 5¥

s1 = Student()

s1.name = "Naitik Mehta"

print(f"{s1.name} having Roll\_No. {s1.roll\_no} is {s1.age} years old.")

class Person:

def init (self,name,age): self.name = name self.age = age

def statement(self):

print(self.name," is",self.age)

a = Person("Naitik",19) a.statement()

class Result:

def init (self,name,roll\_no,marks1,marks2,marks3): self.name = name

self.roll\_no = roll\_no self.marks1 = marks1 self.marks2 = marks2 self.marks3 = marks3

def total(self):

self.t = self.marks1 + self.marks2 + self.marks3 print("Total =",self.t)

def percentage(self):

self.p = self.t\*100/300 print("Percentage =",self.p)

def grade(self):

if self.p >=90:

print("Grade: A") elif self.p >= 75:

print("Grade: B")

else:

def call(self):

print("Grade: Less than B")

self.total()

self.percentage() self.grade()

a = Result("Naitik",5¥,80,80,80) a.call()

#Inheritance #Single Inheritance class Shape:

pass

class Circle(Shape):

def init (self,radius):

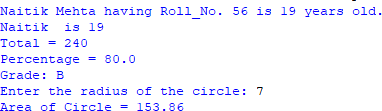
self.radius = radius def area(self):

ar = radius\*radius\*3.14 print("Area of Circle =",ar)

radius = int(input("Enter the radius of the circle: ")) a = Circle(radius)

a.area()

**OUTPUT:**

****

**Conclusion: We studied working of methods in classes, constructors and different ways to create methods in classes and access them in objects.**

## THADOMAL SHAHANI ENGINEERING COLLEGE

#### DEPARTMENT OF INFORMATION TECHNOLOGY

##### Polymorphism : LO3

**Aim:** To show OOP feature of Polymorphism in Python

**Theory:** Polymorphism in Python allows objects of different classes to share the same interface, enabling a single function, method, or operator to work with different data types. It enhances code flexibility and reusability

###### Program with output:

*class Shape: def area(self): pass*

*class Rectangle(Shape): def* ***init****(self, length, breadth): self.length*

*= length self.breadth = breadth*

*def area(self):*

*return self.length \* self.breadth*

*class Square(Shape): def* ***init****(self, side): self.side = side*

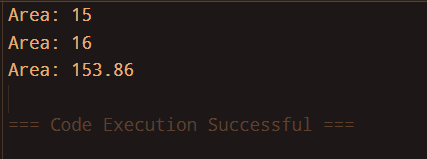
*def area(self):*

*return self.side \*\* 2*

*class Circle(Shape): def* ***init****(self, radius): self.radius = radius def area(self):*

*return 3.14 \* self.radius \*\* 2*

*shapes = [Rectangle(5, 3), Square(4), Circle(7)] for shape in shapes: print(f"Area: {shape.area()}")* **OUTPUT:**



###### Method Overriding

*class Student:*

*def* ***init****(self, name, num\_subjects):*

*self.name = name self.num\_subjects = num\_subjects self.subjects = []*

*self.marks = []*

*def input\_details(self):*

*print(f"Enter details for {self.name}:") for i in range(self.num\_subjects): subject\_name = input(f"Enter name of subject {i+1}: ") marks = float(input(f"Enter marks for {subject\_name}: "))*

*self.subjects.append(subject\_name)*

*self.marks.append(marks)*

*def total\_marks(self):*

*pass*

*def grade(self):*

*pass*

*class ReportCard(Student):*

*def* ***init****(self, name, num\_subjects): super().****init****(name, num\_subjects)*

*def total\_marks(self):*

*return sum(self.marks)*

*def grade(self):*

*total = self.total\_marks() percentage = (total / (self.num\_subjects \* 100)) \* 100*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *if* | *percentage* | | *>=* | | *90:* |
|  | *return* |  |  | | *"A"* |
| *elif* |  | *percentage* | *>=* | | *75:* |
| *elif* | *return* | *percentage* | *>=* | | *"B"*  *50:* |
|  | *return* |  |  | | *"C"* |
| *elif* |  | *percentage* | *>=* | | *35:* |
|  | *return* |  |  | | *"D"* |
| *else:* | | | | | |
| *return* | |  |  | *"F"* | |
| *def* | |  |  | *show\_rc(self):* | |
| *print(f"Name:* | |  |  | *{self.name}")* | |
| *print("Subjects* | |  | *and* | *Marks:")* | |
| *for i* | | *in* |  | *range(self.num\_subjects):* | |

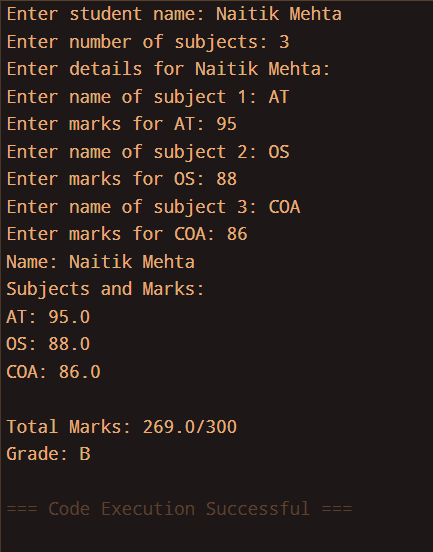
*print(f"{self.subjects[i]}: {self.marks[i]}")*

|  |  |  |
| --- | --- | --- |
| *total* |  | *= self.total\_marks()* |
| *print(f"\nTotal* | *Marks:* | *{total}/{self.num\_subjects \* 100}")* |
| *print(f"Grade:* |  | *{self.grade()}")* |

*name = input("Enter student name: ") num\_subjects = int(input("Enter number of subjects: "))*

*student = ReportCard(name, num\_subjects) student.input\_details() student.show\_rc()*

###### OUTPUT:



**Conclusion:** The first code demonstrates polymorphism by defining a common area() method in the Shape base class, which is overridden by Rectangle, Square, and Circle. The shapes list allows different objects to be processed through a unified interface, enhancing code flexibility and reusability.

The second code demonstrates **polymorphism** by defining total\_marks() and grade() as abstract methods in the Student class, which are overridden in the ReportCard class to provide specific implementations. This allows different student objects to calculate marks and grades dynamically while maintaining a consistent interface.

## THADOMAL SHAHANI ENGINEERING COLLEGE

#### DEPARTMENT OF INFORMATION TECHNOLOGY

1. **Packages : LO4 Aim:** To implement user-defined modules/packages in Python **Theory:**

**Modules**: A module is a single Python file (.py) containing functions, classes, or variables that can be reused. It is imported using import module\_name.

Packages: A package is a collection of modules organized in directories with an init .py file

###### Program with output:

*person1 = {"name":"X","age":20,"city":"Bengaluru"} L1 = [1,2,3,4,5,6,7]*

*x = input("Enter a name: ")*

*def pattern(n): for i in range (n): print() for j in range (i+1): print("\* ",end="")*

*def factorial(x): if x==0 or x==1: return 1 else: return x \* factorial(x-1) def armstrong(a): temp = a sum = 0 while(a!=0): d=a%10 sum = sum + (ddd) a=a//10 if (sum == temp):*

*print(a,"is an armstrong number")*

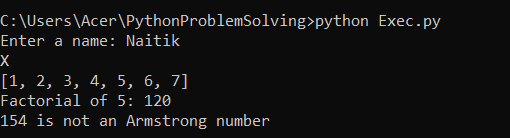
*else:*

*print(a,"is not an armstrong number")*

**NEXT CODE**

*import MyModule as m*

*print(m.person1[name]) print(m.L1) print(m.pattern(5)) print(m.factorial(5)) print(m.armstrong(154))*

**

###### Conclusion:

Using modules and packages promotes code reusability, readability, and maintainability in large projects.

* The import statement allows easy access to built-in, third-party, or custom modules and packages.
* Proper structuring with modules and packages makes Python development efficient, modular, and scalable.

**LO 4 mapped**

## THADOMAL SHAHANI ENGINEERING COLLEGE

#### DEPARTMENT OF INFORMATION TECHNOLOGY

##### Exception Handling: LO1

**Aim:** To implement Exception Handling in Python

###### Theory:

* **Definition**: Exception handling is a mechanism to handle runtime errors, preventing program crashes.
* **Try-Except Block**: Use try to test a block of code and except to handle errors gracefully.
* **Multiple Except Blocks**: Catch different exceptions separately for better debugging.
* **Finally Block**: The finally block executes code regardless of whether an exception occurs.
* **Raising Exceptions**: Use raise to manually trigger an exception when needed.

###### Program with output:

*print("Calculator") try:*

*a = int(input("Enter first number: "))*

*b = int(input("Enter second number: "))*

*print("Choose Opertion: ","1.Add,2.Subtract,3.Multiply,4.Divide") c = int(input())*

*r = 0*

*if c==1:*

*r = a + b print(r)*

*elif c==2:*

*r = a - b print(r)*

*elif c==3:*

*r = a \* b print(r)*

*elif c==4:*

*try:*

*r = a/b print(r)*

*except Exception as e: print(e)*

*else:*

*print("Choose a valid option") except Exception as e:*

*print(e)*

###### OUTPUT:

*print("Temperature") try:*

*print("Enter temperature to convert from: 1. Fahrenheit, 2. Kelvin 3.Celsius")*

*i = int(input())*

*print("Enter temperature to convert to: 1. Fahrenheit, 2. Kelvin 3.Celsius")*

*f = int(input())*

*a = int(input("Enter temperature unit: ")) if i == 1:*

*if f == 1:*

*print(a) elif f == 2:*

*r = ((a-32)\*(5/9))+273*

*print(r) elif f == 3:*

*r = ((a-32)\*(5/9))*

*print(r) elif i == 2:*

*if f == 1:*

*r = 273-((a-32)\*(5/9))*

*print(r) elif f == 2:*

*print(a) elif f == 3:*

*r = a - 273*

*print(r) elif i == 3:*

*if f == 1:*

*r = (a\*(5/9)) + 32*

*print(r) elif f == 2:*

*print(a + 273) elif f == 3:*

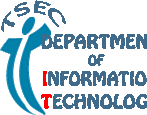
*print(a) except Exception as e:*

*print(e)*

###### OUTPUT:

**Conclusion:** Hence, we demonstrate Exception handling in Python through try-except blocks

##### LO4 mapped

**THADOMAL SHAHANI ENGINEERING COLLEGE**

# DEPARTMENT OF INFORMATION TECHNOLOGY

1. **File Handling : LO1**

**Aim:** To implement different file handling operations in Python

### Theory:

1. **File handling** in Python allows reading, writing, and modifying files using built-in functions.
2. Files are opened using the open(filename, mode) function, where mode can be 'r' (read), 'w' (write), 'a' (append), or 'rb'/'wb' (binary modes).
3. Common methods include read(), readline(), readlines() for reading and write() for writing.
4. Always close a file using close() or use with open(...) as f: to handle files automatically.
5. Exception handling with try-except ensures safe file operations and prevents data corruption.

### Program with output:

*f = open("n1.txt","a") f.write("Built in a fashion") f.close()*

*f = open("n1.txt","r")*

*print(f.read()) f.close()*

*f = open("n2.txt","x") f.close()*

*f = open("n2.txt","w")*

*f.write("White birds across the red city, the theme look so dreamy, yet why...") f.close()*

*f = open("n2.txt","r") print(f.read()) f.close()*

*f = open("cities.txt","r") k = f.readlines() k.sort()*

*f.close()*

*f = open("sorted\_cities.txt","a") f.writelines(k)*

*f.close()*

*f = open("sorted\_cities.txt","r") print(f.read())*

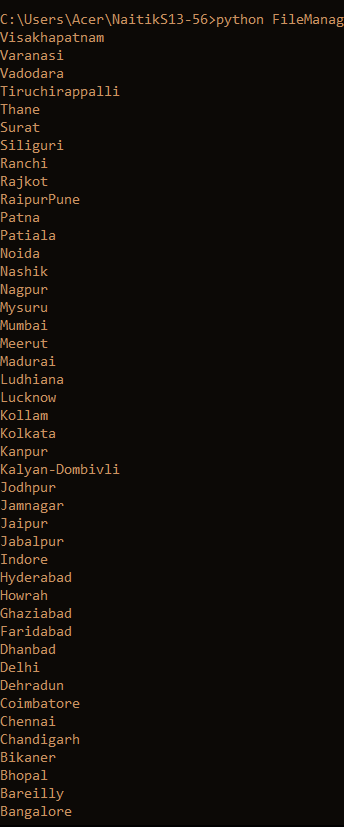
*f = open("cities.txt","r") k = f.readlines()*

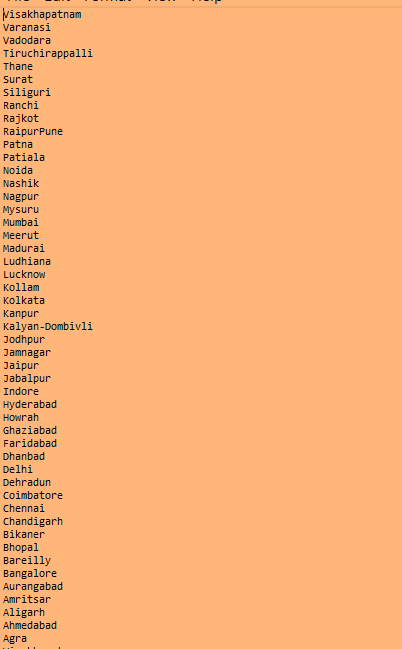
*k.sort(reverse = "true") f.close()*

*f = open("desc\_cities.txt","a") f.writelines(k)*

*f.close()*

*f = open("desc\_cities.txt","r") print(f.read())*

**



**Conclusion:** So, through various examples, we see how we can create, edit, sort and do other management of files directly through Python, with the import os module.

# THADOMAL SHAHANI ENGINEERING COLLEGE

## DEPARTMENT OF INFORMATION TECHNOLOGY

1. **GUI-Application - LO5 Aim:** To implement GUI driven application using Tkinter **Theory:**

**Graphical User Interface (GUI)** development in Python is commonly done using **Tkinter**, which is Python’s built-in GUI toolkit. A GUI allows users to interact with applications using visual elements like **buttons, text fields, labels, and menus** instead of command-line inputs.

Key Features of Tkinter:

1. **Widgets** – Basic building blocks like Button, Label, Entry, Frame, etc.
2. **Event Handling** – Detects user actions (clicks, key presses) and responds accordingly.
3. **Layout Management** – Organizes widgets using grid(), pack(), or

place().

1. **Styling & Themes** – Customization using colors, fonts, and relief styles.

How Tkinter Works:

* + **Mainloop**: The root.mainloop() function keeps the GUI running in an infinite loop, waiting for user interactions.
  + **Widget Placement**: Elements like buttons and text fields are positioned using layout managers (grid, pack, place).
  + **Callbacks**: Functions are assigned to buttons and other interactive elements to perform actions.

##### Program with output:

*import tkinter as tk*

*def on\_click(button\_text): current = entry\_var.get() if button\_text == "=": try: entry\_var.set(eval(current)) except: entry\_var.set("Error") elif button\_text == "C": entry\_var.set("") else: entry\_var.set(current + button\_text)*

*root = tk.Tk() root.title("Calculator") root.geometry("c00x700") root.resizable(False, False) root.configure(bg="black")*

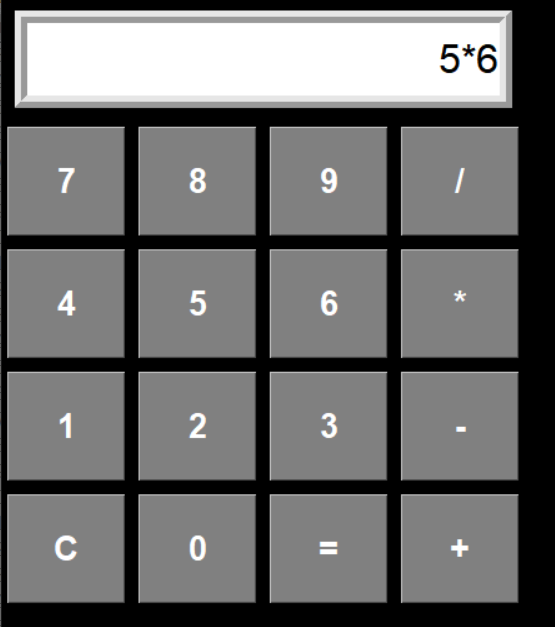
*entry\_var = tk.StringVar() entry = tk.Entry(root, textvariable=entry\_var, font=("Arial", 24), bd=10, relief="ridge", justify="right", bg="white", fg="black") entry.grid(row=0, column=0, columnspan=4, ipadx=8, ipady=10, pady=10)*

*buttons = [ ("7", 1, 0), ("8", 1, 1), ("S", 1, 2), ("/", 1, 3), ("4", 2, 0), ("5", 2, 1), ("c", 2, 2), ("\*", 2,*

*3), ("1", 3, 0), ("2", 3, 1), ("3", 3, 2), ("-", 3, 3), ("C", 4, 0), ("0", 4, 1), ("=", 4, 2), ("+", 4, 3) ]*

*for (text, row, col) in buttons: tk.Button(root, text=text, font=("Arial", 20, "bold"), fg="white", bg="gray", relief="raised", width=5, height=2, command=lambda t=text: on\_click(t)).grid(row=row, column=col, padx=5, pady=5)*

*root.mainloop()*



##### Conclusion:

Tkinter is a simple yet powerful tool for building GUI applications. It provides a platform-independent way to create interactive programs with minimal effort.

**LO 5 mapped**

# THADOMAL SHAHANI ENGINEERING COLLEGE

## DEPARTMENT OF INFORMATION TECHNOLOGY

### SQLlite - LO5

**Aim:** To implement CRUD operations using SQLite3 in Python

##### Theory:

* In Python, the sqlite3 module provides a lightweight, built-in way to interact with SQLite databases — which are self-contained, serverless, and great for local or small-scale apps.
* *Core Concepts:*
* Database Connection:

You use sqlite3.connect() to connect to (or create) a database file.

* Cursor Object:

This is used to execute SQL commands using methods like .execute(), .fetchall(), etc.

* Parameterized Queries:

Queries use ? placeholders to safely insert values, preventing SQL injection.

* Transactions C Commit:

After operations that change data (INSERT, UPDATE, DELETE), .commit() saves the changes.

* Autoincremented IDs:

Primary keys often use AUTOINCREMENT to uniquely identify each row.

CRUD operations is:

Create, Read, Update, Delete

##### Program with output:

import sqlite3

conn = sqlite3.connect("pylab\_56.db") cursor = conn.cursor()

cursor.execute(""" CREATE TABLE IF NOT EXISTS users ( id INTEGER PRIMARY KEY AUTOINCREMENT, name TEXT, age INTEGER ) """)

def create\_user(name, age): cursor.execute("INSERT INTO users (name, age) VALUES (?, ?)", (name, age)) conn.commit()

def read\_users(): cursor.execute("SELECT \* FROM users") return cursor.fetchall()

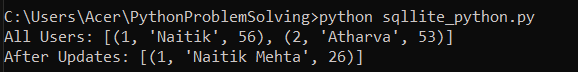
def update\_user(user\_id, name, age): cursor.execute("UPDATE users SET name = ?, age

= ? WHERE id = ?", (name, age, user\_id)) conn.commit()

def delete\_user(user\_id): cursor.execute("DELETE FROM users WHERE id = ?", (user\_id,)) conn.commit()

create\_user("Naitik", 56) create\_user("Atharva", 53) print("All Users:", read\_users())

update\_user(1, "Naitik Mehta", 26) delete\_user(2)

print("After Updates:", read\_users()) conn.close()

##### Conclusion:

* Using Python with SQLite is ideal for quick, local storage needs — no server setup required.
* The structure is modular and reusable, meaning you can expand this into GUI apps (with Tkinter), web apps (with Flask/Django), or even APIs.

**LO 5 mapped**

# THADOMAL SHAHANI ENGINEERING COLLEGE

## DEPARTMENT OF INFORMATION TECHNOLOGY

### Matplotlib : LO1

**Aim:** To implement different types of plots using matplotlib in Python

**Theory:** Matplotlib is a Python library used for creating static, animated, and interactive visualizations. It is one of the most widely used plotting libraries in Python, providing a MATLAB-like interface while being more Pythonic.

1. Figures and Axes

Matplotlib follows an object-oriented approach where:

A Figure (Figure) is the overall container for a plot.

Inside a Figure, there are one or more Axes (Axes), where the actual data is plotted.

1. Pyplot Module

The matplotlib.pyplot module provides a state-based interface similar to MATLAB.

It allows users to quickly generate plots using simple functions like plot(), xlabel(), ylabel(), etc.

1. Backend System

Matplotlib has multiple backends, which control how plots are displayed.

Interactive backends (e.g., TkAgg, Qt5Agg) allow dynamic updates in Jupyter Notebooks and GUI applications.

Non-interactive backends (e.g., Agg) generate static images for reports.

##### Program with output:

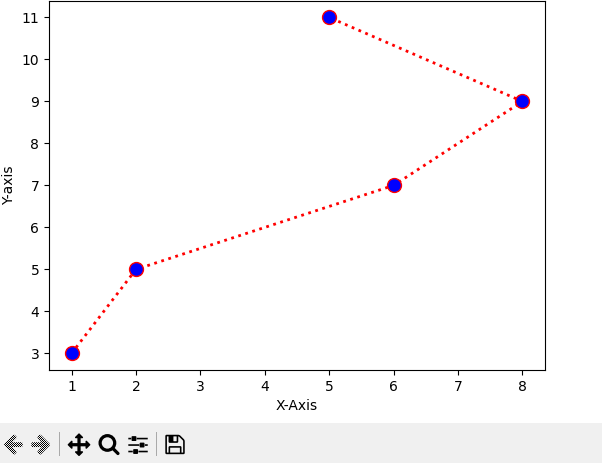
*import pandas as pd*

*import matplotlib.pyplot as mpl import numpy as np*

*xpoints = np.array([1,2,6,8,5]) ypoints = np.array([3,5,7,9,11])*

*mpl.plot(xpoints,ypoints,marker = 'o',ms = 10,mec = 'r', mfc = 'b', linestyle = 'dotted',linewidth =2,color = 'red')*

*mpl.xlabel("X-axis") mpl.ylabel("Y-axis") mpl.plot() mpl.show()*



### Subplot

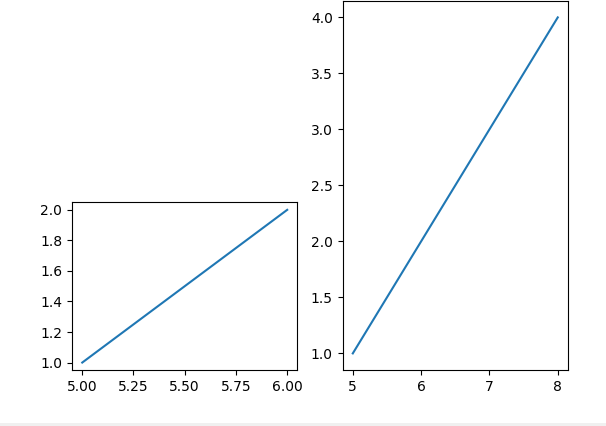
*x = np.array([1,2,3,4])*

*y = np.array([5,6,7,8]) mpl.subplot(1,2,2) mpl.plot(y,x)*

*x = np.array([1,2])*

*y = np.array([5,6]) mpl.subplot(2,2,3) mpl.plot(y,x)*

*mpl.show()*

**

##### Horizontal Bar Graph

*import pandas as pd*

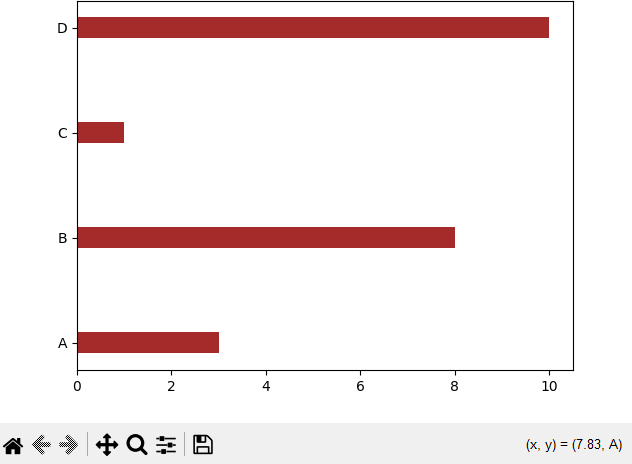
*import matplotlib.pyplot as plt*

*import numpy as np*

*x = np.array(["A","B","C","D"])*

*y = np.array([3,8,1,10])*

*plt.barh(x,y,color = "brown", height = 0.2) plt.show()*

**

##### Pie Chart

*import pandas as pd*

*import matplotlib.pyplot as plt import numpy as np*

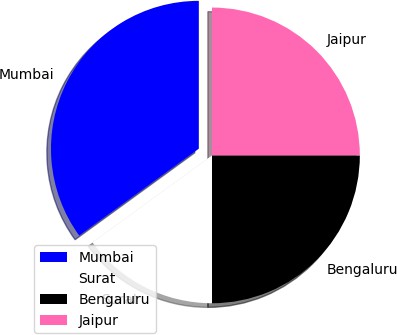
*x = np.array([35,15,25,25])*

*mylabels = ["Mumbai","Surat","Bengaluru","Jaipur"] myexplode = [0.1,0,0,0] myshadow = [0.9,0.2,0.1,0]*

*mycolors = ["blue","white","black","hotpink"] plt.pie(x, labels = mylabels,startangle=90,*

*explode = myexplode, shadow = myshadow, colors = mycolors) plt.legend(loc = "lower left")*

*#plt.legend(bbox\_ to\_anchor=(1,3,1)) plt.show()*

**

**Conclusion:** So we have seen different types of plots using Python’s matplotlib library

**LO 6 mapped.**

## THADOMAL SHAHANI ENGINEERING COLLEGE

**DEPARTMENT OF INFORMATION TECHNOLOGY**

### Pandas : LO6

##### Aim: Theory:

Pandas is a powerful **Python library** used for **data analysis and manipulation**. It provides flexible data structures, primarily **Series** and **DataFrame**, to handle structured data efficiently.

Key Data Structures in Pandas

* **Series**: A one-dimensional labeled array, similar to a list or column in Excel.
* **DataFrame**: A two-dimensional table-like data structure with labeled rows and columns.

##### Program with output:

**Pandas**

*import pandas as pd*

*ds = {*

*'id': [1,2,3],*

*'name': ['Mumbai','Pune','Surat'], 'age': [200,150,300]*

*}*

*print(ds) print()*

*df = pd.DataFrame(ds) print(df)*

*print() print(df.head(2)) print() print(df.tail(1)) print() print(df.shape) print(df.size)*

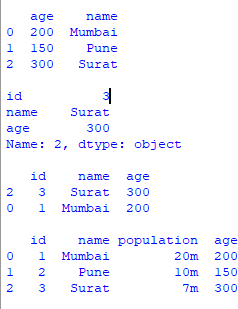
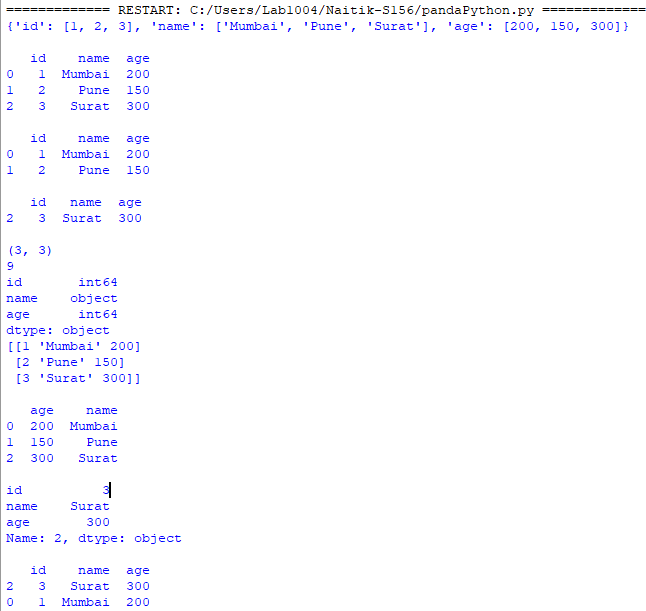
*print(df.dtypes) print(df.values)*

*df[df["age"]>170] print()*

*print(df[["age", "name"]])*

*print() print(df.loc[2]) print() print(df.loc[[2,0]])*

*print()*

**

**Conclusion:** So this is how we do data manipulation as Data-Frames using Pandas

**LO6 mapped.**

# THADOMAL SHAHANI ENGINEERING COLLEGE

## DEPARTMENT OF INFORMATION TECHNOLOGY

### SciPy - LO6

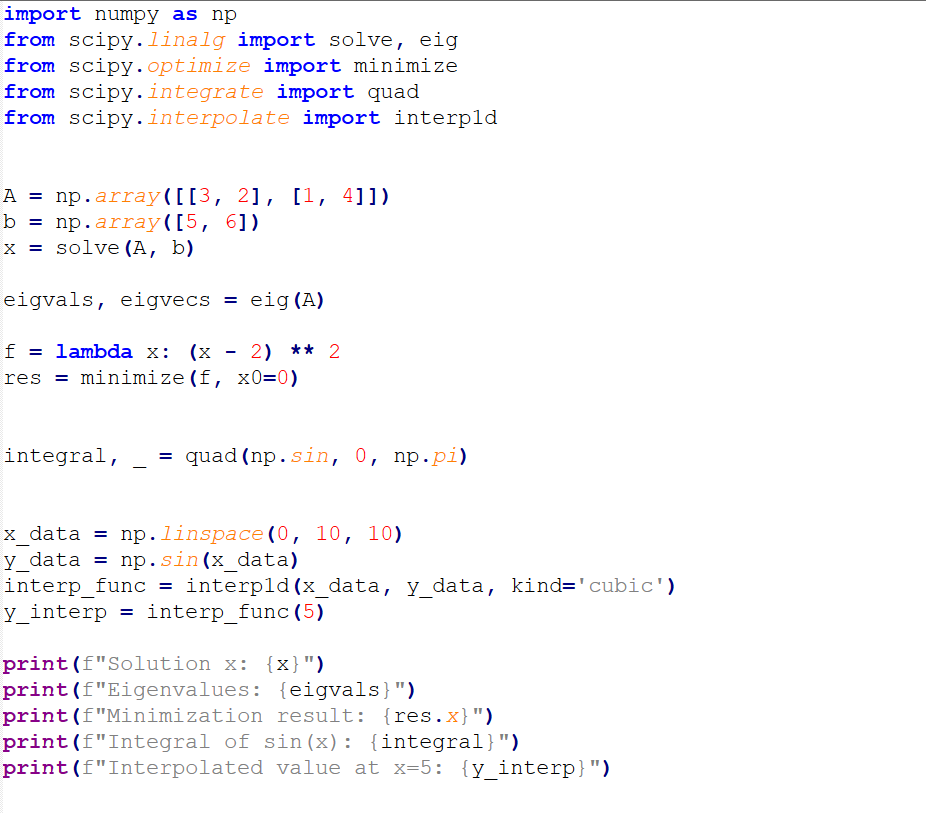
**Aim:** To implement basic operations using SciPy library and linear algebra in Python

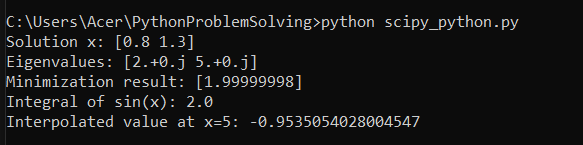
##### Theory:

SciPy is a powerful scientific computing library in Python that provides tools for numerical computation. The script demonstrates several key functionalities of SciPy:

1. Linear Algebra (scipy.linalg)
   1. Used to solve systems of linear equations: *Ax=bAx = b*Ax=b.
   2. Also used to compute eigenvalues and eigenvectors of a matrix.
2. Optimization (scipy.optimize)
   1. The minimize function finds the minimum of a given function using numerical methods.
3. Integration (scipy.integrate)
   1. The quad function is used for numerical integration of functions over a given interval.
4. Interpolation (scipy.interpolate)
   1. The interp1d function constructs an interpolating function based on given data points.

##### Program with output:



****

**Conclusion:**

SciPy provides efficient tools for numerical computation, making it ideal for scientific applications and simplifies complex mathematical problems and is essential for scientific and engineering applications.

**LO 6 mapped**

# THADOMAL SHAHANI ENGINEERING COLLEGE

## DEPARTMENT OF INFORMATION TECHNOLOGY

### Flask - LO6

**Aim:** To implement CRUD operations using SQLite3 in Python

##### Theory:

* **Flask** is a lightweight Python web framework used for building web apps.
* It’s part of the **microframework** family, meaning it doesn’t come with built-in form handling, database abstraction, or authentication — but you can add what you need.
* Flask is based on **Werkzeug (for the WSGI server)** and **Jinja2 (for templating)**.

***Key Concepts:***

###### Concept Description

Flask The main application object

@app.route() URL routing decorator to bind a function to a URL

render\_templat e()

Loads and returns HTML files using Jinja templates

request Access data from incoming requests (GET, POST, etc.)

##### Program with output:

from flask import Flask, request app = Flask(name)

@app.route("/") def home(): return """ Welcome to Flask

Flask is a lightweight Python web framework used for building web apps.

Click Say Hello """

@app.route("/hello/") def hello(name): return f""" Hello, {name}!

Back to Home """

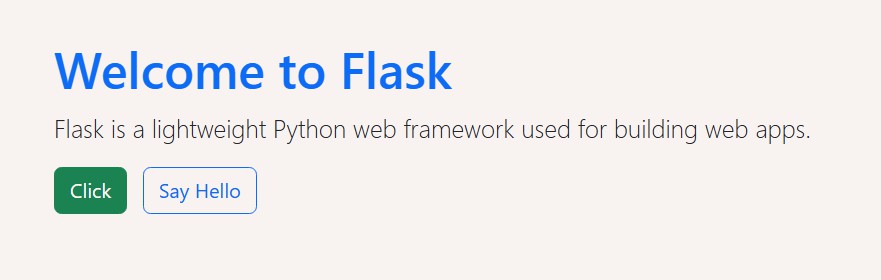
@app.route("/form", methods=["GET", "POST"]) def form(): if request.method == "POST": name = request.form.get("name") return f"""

Thank you, {name}!

Back to Home """ return """ Enter Your Name Name:

Submit Back to Home """

if name == "main": app.run(debug=True)



##### Conclusion:

* **Flask is ideal** for beginners and professionals to build web applications with Python.
* Its simplicity and flexibility make it perfect for rapid development and learning core web principles.

**LO 6 mapped**