**Python Lab**

**Practical 1: Iterators**

1) Iter() function and next() function

mytuple = ("apple", "banana", "cherry")

myit = iter(mytuple)

print(next(myit))

print(next(myit))

print(next(myit))

2) Iter() function

mystr = "banana"

myit = iter(mystr)

print(next(myit))

print(next(myit))

print(next(myit))

print(next(myit))

print(next(myit))

print(next(myit))

3) Traversal using for loop

mytuple = ("apple", "banana", "cherry")

for x in mytuple:

print(x)

4) Traversal using for loop

mystr = "banana"

for x in mystr:

print(x)

5) Create an iterator that returns numbers, starting with 1, and each sequence will increase by one

class MyNumbers:

def \_\_iter\_\_(self):

self.a = 1

return self

def \_\_next\_\_(self):

x = self.a

self.a += 1

return x

myclass = MyNumbers()

myiter = iter(myclass)

print(next(myiter))

print(next(myiter))

print(next(myiter))

print(next(myiter))

print(next(myiter))

6) simpleGenerator() function with next() function

def simpleGeneratorFun():

yield 1

yield 2

yield 3

# x is a generator object

x = simpleGeneratorFun()

# Iterating over the generator object using next

print(next(x))

print(next(x))

print(next(x))

7) simpleGenerator() function with for loop

def simpleGeneratorFun():

yield 1

yield 2

yield 3

for value in simpleGeneratorFun():

print(value)

8) Custom generator function

def mygenerator(n):

for i in range(1,n):

yield i\*\*3

9) Custom generator function

def try\_generator(y):

n = y

n += 1

print("Performed addition")

yield n

n \*= 2

print("Performed multiplication")

yield n

result = try\_generator(5)

print(next(result))

print(next(result))

**Practical 2: tuples, lists and arrays using Numpy**

1) create numpy array

import numpy as np

#create numpy array

a = np.array([5, 8, 12])

print(a)

2) create numpy array using arrange() function

import numpy as np

#create numpy array

a = np.arange(5, 14, 2)

print(a)

3) operations on NumPy array

# Python code to perform arithmetic

# operations on NumPy array

import numpy as np

# Initializing the array

arr1 = np.arange(4, dtype = np.float\_).reshape(2, 2)

print('First array:')

print(arr1)

print('\nSecond array:')

arr2 = np.array([12, 12])

print(arr2)

print('\nAdding the two arrays:')

print(np.add(arr1, arr2))

print('\nSubtracting the two arrays:')

print(np.subtract(arr1, arr2))

print('\nMultiplying the two arrays:')

print(np.multiply(arr1, arr2))

print('\nDividing the two arrays:')

print(np.divide(arr1, arr2))

4) Python code to perform power operation on NumPy array

# Python code to perform power operation on NumPy array

import numpy as np

arr = np.array([5, 10, 15])

print('First array is:')

print(arr)

print('\nApplying power function:')

print(np.power(arr, 2))

print('\nSecond array is:')

arr1 = np.array([1, 2, 3])

print(arr1)

print('\nApplying power function again:')

print(np.power(arr, arr1))

5) Determining the datatype of an array  
x = np.array([[1, 2, 3], [4, 5, 6]], np.int32)

type(x)

x.dtype

6) Different types of arrays

import numpy as nmp

X = nmp.array( [ [ 1, 6, 7], [ 5, 9, 2] ] )

print(X) #Array of integers

X = nmp.array( [ [ 1, 6.2, 7], [ 5, 9, 2] ] )

print(X) #Array of floats

X = nmp.array( [ [ 1, 6, 7], [ 5, 9, 2] ], dtype = complex )

print(X)

7)

# Python program to demonstrate functions

def evenOdd( x ):

if (x % 2 == 0):

print("even")

else:

print("odd")

#CALLING FUNCTION

evenOdd(2)

evenOdd(3)

8) Python program to demonstrate Keyword Arguments  
# Python program to demonstrate Keyword Arguments

def student(firstname, lastname):

print(firstname, lastname)

# Keyword arguments

student(firstname ='SNEHA', lastname ='PAKLE')

student(lastname ='PAKLE', firstname ='SNEHA')

9) Square of sum of two numbers  
def nsquare(x, y):

return (x\*x + 2\*x\*y + y\*y)

print("The square of the sum of 2 and 3 is : ", nsquare(2, 3))

10) Lambda functions

x = lambda a : a + 10

print(x(5))

11) Lambda functions

x = lambda a, b, c : a + b + c

print(x(5, 6, 2))

12) Lambda functions

str1 = 'HELLO WORLD'

# lambda returns a function object

rev\_upper = lambda string: string.upper()[::-1]

print(rev\_upper(str1))

**Practical 3: Constructors, Static methods, Operator overloading, Method overloading and overriding**

1) Class and Object

class Dog:

attr1 = "dog"

attr2 = "tommy"

def fun(self):

print("I'm a", self.attr1)

print("My name is", self.attr2)

d1 = Dog()

print(d1.attr1)

d1.fun()

2) Constructor

# constructor

class Person:

def \_\_init\_\_ (self, name):

self.name = name

def hello(self):

print('Hello, my name is', self.name)

p = Person('Aaman')

p.hello()

3) Inner Class

# Inner Class

class Color:

def \_\_init\_\_ (self):

self.name = 'Blue'

self.lg = self.Dark\_Blue()

def show(self):

print("Name:", self.name)

class Dark\_Blue:

def \_\_init\_\_ (self):

self.name = 'Dark\_Blue'

self.code = '\*\*\*\*\*\*\*'

def display(self):

print("Name:", self.name)

print("Code:", self.code)

outer = Color()

outer.show()

g = outer.lg

g.display()

4) Simple Inheritance

# Simple Inheritance

class Parent:

def \_\_init\_\_ (self,name):

self.name = name

print(self.name,"parent class")

class Child(Parent):

pass

kid = Child("Aaman")

5) Multilevel Inheritance

# Mutilevel

class Grand:

age\_grand = 95

class Dad(Grand):

age\_dad = 55

class Son(Dad):

age\_son = 19

son = Son()

print(son.age\_grand, son.age\_dad, son.age\_son)

6) Multiple Inheritance

# Multiple

class Base1:

a="class 1"

class Base2:

b="class 2"

class Child1(Base1,Base2):

c="class 3"

kid = Child1()

print(kid.a,kid.b,kid.c)

7) Hierarchical Inheritance

# Hierarchical

class Shape:

a = "this the shape class"

class Circle(Shape):

b = "This is the circle class"

class Rectangle(Shape):

c = "This is the rectangle class"

xyz = Circle()

abc = Rectangle()

print(xyz.a, xyz.b, "\n", abc.a, abc.c)

8) Operator Overloading  
# Overloading

class Abc:

def \_\_init\_\_ (self, a):

self.a = a

# adding two objects

def \_\_add\_\_ (self, o):

return self.a + o.a

ob1 = Abc(1)

ob2 = Abc(2)

ob3 = Abc("Aaman")

ob4 = Abc("Bhowmick")

print(ob1 + ob2)

print(ob3 + ob4)

9) Method Overloading

# Method Overloading

def product(a, b):

p = a \* b

print(p)

def product(a, b, c):

p = a \* b\*c

print(p)

product(4, 5, 5)

10) Method Overriding

# Method Overriding

class Parent():

def \_\_init\_\_ (self):

self.value = "Inside Parent"

def show(self):

print(self.value)

class Child(Parent):

# Constructor

def \_\_init\_\_ (self):

self.value = "Inside Child"

def show(self):

print(self.value)

obj1 = Parent()

obj2 = Child()

obj1.show()

obj2.show()

11) Abstract Class & Method

import abc

class Shape(metaclass=abc.ABCMeta):

@abc.abstractmethod

def area(self):

pass

class Rectangle(Shape):

def \_\_init\_\_ (self, x,y):

self.l = x

self.b=y

def area(self):

return self.l\*self.b

r = Rectangle(10,20)

print ('area: ',r.area())

12) Interface

# Interface

import zope.interface

class MyInterface(zope.interface.Interface):

x = zope.interface.Attribute("abc")

def method1(self, x):

pass

def method2(self):

pass

print(type(MyInterface))

print(MyInterface. module )

print(MyInterface. name )

x = MyInterface['x']

print(x)

print(type(x))

**Practical 4: Threading and Modules**

1)Python program to illustrate the concept of threading

#importing the threading module

import threading

def print\_cube(num):

print("Cube: {}" .format(num \* num \* num))

def print\_square(num):

print("Square: {}" .format(num \* num))

#It Allows You to Execute Code When the File Runs as a Script, but Not When It’s Imported as a Module

if \_\_name\_\_ =="\_\_main\_\_":

# creating thread

t1 = threading.Thread(target=print\_square, args=(10,))

t2 = threading.Thread(target=print\_cube, args=(10,))

t1.start()

t2.start()

# wait until thread 1 is completely executed

t1.join()

# wait until thread 2 is completely executed

t2.join()

# both threads completely executed

print("Done!")

2) Threading inside function

from threading import \*

def show():

print("this is child thread")

t=Thread(targert=show())

t.start()

print("parent thread")

3) a custom function that blocks for a moment

def task():

# block for a moment

sleep(1)

# display a message

print('This is from another thread')

4) example of extending the Thread class

from time import sleep

from threading import Thread

# custom thread class

class CustomThread(Thread):

# override the run function

def run(self):

# block for a moment

sleep(1)

# display a message

print('This is coming from another thread')

# create the thread

thread = CustomThread()

# start the thread

thread.start()

# wait for the thread to finish

print('Waiting for the thread to finish')

thread.join()

5) example of assessing whether a thread is alive

from threading import Thread

# create the thread

thread = Thread()

# report the thread is alive

print(thread.is\_alive())

6) Math module

# to import standard module math

import math

print("The value of pi is", math.pi)

7) Math module

# import module by renaming it

import math as m

print("The value of pi is", m.pi)

8) Math module

# import only pi from math module

from math import pi

print("The value of pi is", pi)

9) Math module

# import all names from the standard module math

from math import \*

print("The value of pi is", pi)

10) Math module

import math

math.sin(0.52)

print(math.sin(0.52))

11) Math module

import math

math.log(10)

print(math.log(10))

12) Math module

import math

math.exp(44)

13) Random module  
# ranndint function of the random module returns a random number between a given range, here (20 to 100)

import random

random.randint(20, 100)

print(random.randint(20, 100))

14) os module

# Python program to explain os.getcwd() method

# importing os module

import os

# Get the current working directory (CWD)

cwd = os.getcwd()

# Print the current working directory (CWD)

print("Current working directory-", cwd)

15) os module

# Python program to explain os.getcwd() method

# importing os module

import os

# Get the current working directory (CWD)

cwd = os.getcwd()

# Print the current working directory (CWD)

print("Current working directory-", cwd)

16) os module

# create a directory

import os

dir = os.path.join("....")

if not os.path.exists(dir):

os.mkdir(dir)

18) sys module

import sys

print(sys.version)

19) sys module

import sys

print(sys.path)

20) sys module

# returns name of python modules that current shell has imported

import sys

print(sys.modules)

21) sys module

import sys

sys.maxsize

print(sys.maxsize)

22) collections module

#OrderedDict()

import collections

d1 = collections.OrderedDict()

d1['A'] = 65

d1['C'] = 67

d1['B'] = 66

d1['D'] = 68

for k,v in d1.items():

print (k,v)

23) collections module  
import collections

q=collections.deque([10,20,30,40])

q.appendleft(0)

q

24) collections module

import collections

q=collections.deque([10,20,30,40])

q.append(50)

q

25) collections module

import collections

q=collections.deque([10,20,30,40])

q.pop()

q

26) collections module

import collections

q=collections.deque([10,20,30,40])

q.popleft()

27) re module

# Program to extract numbers from a string (regular exp)

import re

string = 'hello 12 hi 89. Howdy 34'

pattern = '\d+'

result = re.findall(pattern, string)

print(result)

28) re.split

#re.split

import re

string = 'Twelve:12 Eighty nine:89 Nine:9.'

pattern = '\d+'

# maxsplit = 1

# split only at the first occurrence

result = re.split(pattern, string, 1)

print(result)

29) Search for a string pattern using regular expressions

import re

pattern = '^a...s$'

test\_string = 'abyss'

result = re.match(pattern, test\_string)

if result:

print("Search successful.")

else:

print("Search unsuccessful.")

**Practical 5: Exception Handling and types of errors**

1) Simple try catch

try:

print(x)

except:

print("An exception occurred")

2) Try catch with specified exceptions

try:

print(x)

except NameError:

print("Variable x is not defined")

except:

print("Something else went wrong")

3) Try catch and finally

try:

print(x)

except:

print("Something went wrong")

finally:

print("The 'try except' is finished")

4) Nested try, catch and finally

try:

f = open("demofile.txt")

try:

f.write("Lorum Ipsum")

except:

print("Something went wrong when writing to the file")

finally:

f.close()

except:

print("Something went wrong when opening the file")

5) Custom exceptions

class NoMoneyException(Exception):

pass

class OutOfBudget(Exception):

pass

balance = int(input("Enter a balance: "))

if balance < 1000:

raise NoMoneyException

elif balance > 10000:

raise OutOfBudget

**Type of errors**

1) Compile time errors

#Compile-Time Errors

#Syntax Errors

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

if(n<5):

print(n)

2) Semantic errors

#Semantics Errors

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

sum=0

sum= sum+n

3) Logical errors

x = float(input('Enter a number: '))

y = float(input('Enter a number: '))

z = x+y/2

print ('The result is:',z)

4) Index errors

list\_example= [10,20,30]

print(list\_example[3])

5) Module error

import myModule

list\_example= [10,20,30]

print(list\_example[3])

6) Value error

my\_value= int('abc')

print(my\_value)

7) Division by zero

my\_value= 10/0

print(my\_value)

8) Name error

my\_value= 10/5

print(age)

9) Handling runtime error

a = [1, 2, 3]

try:

print ("Second element = %d" %(a[1]))

# Throws error since there are only 3 elements in array

print ("Fourth element = %d" %(a[3]))

except:

print ("An error occurred")

10) Program to depict else clause with try-except Function which returns a/b

def AbyB(a , b):

try:

c = ((a+b) / (a-b))

except ZeroDivisionError:

print ("a/b result in 0")

else:

print (c)

# Driver program to test above function

AbyB(2.0, 3.0)

AbyB(3.0, 3.0)

11) Python program to demonstrate finally

# No exception Exception raised in try block

try:

k = 5//0 # raises divide by zero exception.

print(k)

# handles zerodivision exception

except ZeroDivisionError:

print("Can't divide by zero")

finally:

# this block is always executed

# regardless of exception generation.

print('This is always executed')

**Practical 6: File handling**

1)# Open function to open the file "MyFile1.txt"

file1 = open("MyFile1.txt","a")

file2 = open(r"D:\Text\MyFile2.txt","w+")

2)# Open function to open the file "MyFile1.txt"

file1 = open("MyFile1.txt","a")

file2 = open(r"D:\Text\MyFile2.txt","w+")

3) Program to show various ways to read and write data in a file.

file1 = open("myfile.txt","w")

L = ["This is Delhi \n","This is Paris \n","This is London \n"]

# \n is placed to indicate EOL (End of Line)

file1.write("Hello \n")

file1.writelines(L)

file1.close() #to change file access modes

file1 = open("myfile.txt","r+")

print("Output of Read function is ")

print(file1.read())

print()

# seek(n) takes the file handle to the nth byte from the beginning.

file1.seek(0)

print( "Output of Readline function is ")

print(file1.readline())

print()

file1.seek(0)

# To show difference between read and readline

print("Output of Read(9) function is ")

print(file1.read(9))

print()

file1.seek(0)

print("Output of Readline(9) function is ")

print(file1.readline(9))

file1.seek(0)

# readlines function

print("Output of Readlines function is ")

print(file1.readlines())

print()

file1.close()

4)# Python program to illustrate Append vs write mode

file1 = open("myfile.txt","w")

L = ["This is Delhi \n","This is Paris \n","This is London \n"]

file1.writelines(L)

file1.close()

# Append-adds at last

file1 = open("myfile.txt","a")#append mode

file1.write("Today \n")

file1.close()

file1 = open("myfile.txt","r")

print("Output of Readlines after appending")

print(file1.readlines())

print()

file1.close()

# Write-Overwrites

file1 = open("myfile.txt","w")#write mode

file1.write("Tomorrow \n")

file1.close()

file1 = open("myfile.txt","r")

print("Output of Readlines after writing")

print(file1.readlines())

print()

file1.close()

5)Python program to copy the contents of a file to another file

# open both files

with open('myfile.txt','r') as firstfile, open('MyFile.txt','w') as secondfile:

line = firstfile.read()

secondfile.write(line)

6) Write a Python program to count the number of each character in a text file.

#open file in read mode

file = open("myfile.txt", "r")

#read the content of file

data = file.read()

#get the length of the data

number\_of\_characters = len(data)

print('Number of characters in text file :', number\_of\_characters)

7) Count Characters in a Text File excluding spaces

#open file in read mode

file = open("myfile.txt", "r")

#read the content of file and replace spaces with nothing

data = file.read().replace(" ","")

#get the length of the data

number\_of\_characters = len(data)

print('Number of characters in text file :', number\_of\_characters)

8) Python program to print each line of a file in reverse order

# Open the file in write mode

f1 = open("MyFile1.txt", "w")

with open("MyFile1.txt", "r") as myfile:

data = myfile.read()

# For Full Reversing we will store the value of data into new variable data\_1 in a reverse order using [start: end: step],

#where step when passed -1 will reverse the string

data\_1 = data[::-1]

# Now we will write the fully reverse data in the output1 file using following command

f1.write(data\_1)

f1.close()

**Practical 7: GUI using Tkinter, PyQt**

**Tkinter**

import tkinter as tk

# create the window

root = tk.Tk()

root.title("Classroom Attendance System")

# create the labels

name\_label = tk.Label(root, text="Name")

date\_label = tk.Label(root, text="Date")

subject\_label = tk.Label(root, text="Subject")

time\_label = tk.Label(root, text="Time")

# create the entry boxes

name\_entry = tk.Entry(root)

date\_entry = tk.Entry(root)

subject\_entry = tk.Entry(root)

time\_entry = tk.Entry(root)

# create the button

submit\_button = tk.Button(root, text="SUBMIT")

# create the display box

display\_box = tk.Text(root, height=5, width=40)

# create the listbox

listbox = tk.Listbox(root, height=5, width=40)

# lay out the widgets using grid

name\_label.grid(row=0, column=0)

date\_label.grid(row=1, column=0)

subject\_label.grid(row=2, column=0)

time\_label.grid(row=3, column=0)

name\_entry.grid(row=0, column=1)

date\_entry.grid(row=1, column=1)

subject\_entry.grid(row=2, column=1)

time\_entry.grid(row=3, column=1)

submit\_button.grid(row=4, column=0)

display\_box.grid(row=5, column=1)

listbox.grid(row=5, column=0)

# create the functionality

def submit\_attendance():

# get the input from the entry boxes

name\_value = name\_entry.get()

date\_value = date\_entry.get()

subject\_value = subject\_entry.get()

time\_value = time\_entry.get()

# insert the values into the display box

display\_box.insert(tk.END, f"{name\_value} attended {subject\_value} class on {date\_value} at {time\_value}\n")

# insert the values into the listbox

listbox.insert(tk.END, f"{name\_value}")

# bind the button to the function

submit\_button.config(command=submit\_attendance)

# run the window

root.mainloop()

**PyQt**

**Practical 8: Matplotlib**

1) matplotlib markers

import matplotlib.pyplot as plt

import numpy as np

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

plt.plot(ypoints, marker = 'o')

plt.show()

2) Bar plot

# importing matplotlib module

from matplotlib import pyplot as plt

# x-axis values

x = [5, 2, 9, 4, 7]

# Y-axis values

y = [10, 5, 8, 4, 2]

# Function to plot the bar

plt.bar(x,y)

# function to show the plot

plt.show()

3) line plot

from matplotlib import pyplot as plt

# x-axis values

x = [5, 2, 9, 4, 7]

# Y-axis values

y = [10, 5, 8, 4, 2]

# Function to plot

plt.plot(x,y)

# function to show the plot

plt.show()

4) histogram

from matplotlib import pyplot as plt

# Y-axis values

y = [10, 5, 8, 4, 2]

# Function to plot histogram

plt.hist(y)

# Function to show the plot

plt.show()

5) scatter plot

import matplotlib.pyplot as plt

x =[5, 7, 8, 7, 2, 17, 2, 9,

4, 11, 12, 9, 6]

y =[99, 86, 87, 88, 100, 86,

103, 87, 94, 78, 77, 85, 86]

plt.scatter(x, y, c ="black")

# To show the plot

plt.show()

6) Scatter plot with different shape and colour for two datasets.

import matplotlib.pyplot as plt

# dataset-1

x1 = [89, 43, 36, 36, 95, 10,

66, 34, 38, 20]

y1 = [21, 46, 3, 35, 67, 95,

53, 72, 58, 10]

# dataset2

x2 = [26, 29, 48, 64, 6, 5,

36, 66, 72, 40]

y2 = [26, 34, 90, 33, 38,

20, 56, 2, 47, 15]

plt.scatter(x1, y1, c ="pink",

linewidths = 2,

marker ="s",

edgecolor ="green",

s = 50)

plt.scatter(x2, y2, c ="yellow",

linewidths = 2,

marker ="^",

edgecolor ="red",

s = 200)

plt.xlabel("X-axis")

plt.ylabel("Y-axis")

plt.show()

7)scatter diagram of comparing two subject marks of Mathematics and Science. Use marks of 10 students.

import matplotlib.pyplot as plt

import pandas as pd

math\_marks = [88, 92, 80, 89, 100, 80, 60, 100, 80, 34]

science\_marks = [35, 79, 79, 48, 100, 88, 32, 45, 20, 30]

marks\_range = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

plt.scatter(marks\_range, math\_marks, label='Math marks', color='r')

plt.scatter(marks\_range, science\_marks, label='Science marks', color='g')

plt.title('Scatter Plot')

plt.xlabel('Marks Range')

plt.ylabel('Marks Scored')

plt.legend()

plt.show()

8) plots the points in the order they appear in the dataframe.

import plotly.express as px

df = px.data.iris()

fig = px.area(df, x="sepal\_width", y="sepal\_length",

color="species",

hover\_data=['petal\_width'],)

fig.show()

**Practical 9: Pandas and Scipy**

1) Printing an array in series using pandas

import pandas as pd

import numpy as np

info = np.array(['P','a','n','d','a','s'])

a = pd.Series(info)

print(a)

2) Creating data frame using pandas

import pandas as pd

# a list of strings

x = ['Python', 'Pandas']

# Calling DataFrame constructor on list

df = pd.DataFrame(x)

print(df)

3) Create an Empty Series

import pandas as pd

x = pd.Series()

print (x)

4) Create a Series from dict

#import the pandas library

import pandas as pd

import numpy as np

info = {'x' : 0., 'y' : 1., 'z' : 2.}

a = pd.Series(info)

print (a)

5) Accessing data from series with Position

import pandas as pd

x = pd.Series([1,2,3],index = ['a','b','c'])

#retrieve the first element

print (x[0])

6) Retrieving Shape

#The shape of the Series object defines total number of elements including missing or empty values(NaN)

import numpy as np

import pandas as pd

a=pd.Series(data=[1,2,3,4])

b=pd.Series(data=[4.9,8.2,5.6],index=['x','y','z'])

print(a.shape)

print(b.shape)

7) Convert Pandas DataFrame to CSV

import pandas as pd

data = {'Name': ['Smith', 'Parker'], 'ID': [101, 102], 'Language': ['Python', 'JavaScript']}

info = pd.DataFrame(data)

print('DataFrame Values:\n', info)

# default CSV

csv\_data = info.to\_csv()

print('\nCSV String Values:\n', csv\_data)

8) Solve linear equations with the help of scipy

#7x + 2y = 8

#4x + 5y = 10

from scipy import linalg

import numpy as np

# The function takes two arrays

a = np.array([[7, 2], [4, 5]])

b = np.array([8, 10])

# Solving the linear equations

res = linalg.solve(a, b)

print(res)

9) Find the determinant of a matrix using scipy

from scipy import linalg

import numpy as np

# Initializing the matrix A

A = np.array([[9 , 6] , [4 , 5]])

# Finding the determinant of matrix A

D = linalg.det(A)

print(D)

10) Finding eigen values and eigen vectors for a given matrix

# Importing the required libraries

from scipy import linalg

import numpy as np

# Initializing the matrix M

M = np.array([[9 , 3] , [2 , 4]])

# Passing the values to the eigen function

val , vect = linalg.eig(M)

# Display the Eigen values and Eigen vectors

print(val)

print(vect)

**Practical 10: Create a basic flask application**

from flask import Flask, render\_template\_string, request

app = Flask(\_\_name\_\_)

# HTML template with JavaScript code for the to-do list

todo\_template = '''

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>To-Do List</title>

<script>

function add\_item() {

var item\_text = document.getElementById("item-text").value;

if (item\_text != "") {

var item = document.createElement("li");

item.innerHTML = item\_text;

item.onclick = function() {

this.parentNode.removeChild(this);

}

document.getElementById("item-list").appendChild(item);

document.getElementById("item-text").value = "";

}

}

</script>

</head>

<body>

<h1>To-Do List</h1>

<ul id="item-list"></ul>

<input type="text" id="item-text">

<button onclick="add\_item()">Add Item</button>

</body>

</html>

'''

@app.route('/')

def todo():

return render\_template\_string(todo\_template)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

**QB**

**1)Python Program to print Armstrong Number on given specific range**

**def isArmstrong(num):**

**cpy = num**

**sum = 0**

**while num!=0:**

**last = num%10**

**sum = sum+(last\*\*3)**

**num = num//10**

**if sum == cpy:**

**return True**

**else:**

**return False**

**num = int(input("Enter a range\n"))**

**for i in range(1, num+1):**

**if isArmstrong(i):**

**print(i)**

**3)Program to check prime number**

**def check\_prime(num):**

**prime = True**

**for i in range(2,num):**

**if num%i == 0:**

**prime= False**

**if prime:**

**print(f'{num} is prime')**

**else:**

**print(f'{num} is not prime')**

**check\_prime(69)**

**def is\_prime(*num*):**

**if *num* <= 1:**

**return False**

**elif *num* == 2:**

**return True**

**elif *num* % 2 == 0:**

**return False**

**else:**

**for i in *range*(3, *int*(*num*\*\*0.5) + 1, 2):**

**if *num* % i == 0:**

**return False**

**return True**

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

**if is\_prime(n):**

***print*("Number is prime")**

**else:**

***print*("Number is not prime")**

**5)Program to implement Generator and Decorator**

***# generator***

**def fibonacci(*n*):**

**a, b = 0, 1**

**for i in *range*(*n*):**

**yield a**

**a, b = b, a + b**

**for num in fibonacci(10):**

***print*(num)**

***# Decorator***

**import functools TV**

**def log(*func*):**

**@functools.wraps(*func*)**

**def wrapper(*\*args*, *\*\*kwargs*):**

***print*(f"Calling function {*func*.*\_\_name\_\_*} with args {*args*} and kwargs {*kwargs*}")**

**return *func*(\**args*, \*\**kwargs*)**

**return wrapper**

**@log**

**def my\_function(*x*, *y*):**

**return *x* + *y***

**result = my\_function(3, 4)**

***print*(result)**

**5)Importing Numpy perform various operations on Arrays**

**import numpy as np**

**arr1 = np.array([1,2,4])**

**arr2 = np.array([2, 1, 3])**

**print("Array 1: \n", arr1)**

**print("Array 2: \n", arr2)**

**print("Array 1 + Array2: \n", np.add(arr1, arr2))**

**print("Array 1 - Array2: \n", np.subtract(arr1, arr2))**

**print("Array 1 \* Array2: \n", np.multiply(arr1, arr2))**

**print("Array 1 / Array2: \n", np.divide(arr1, arr2))**

**print("Array 1 ^ Array2: \n", np.power(arr1, arr2))**

**6)Python Program to Shuffle Deck of Cards implement using Functions**

**# Python program to shuffle a deck of card**

**# importing modules**

**import itertools, random**

**# make a deck of cards**

**deck = list(itertools.product(range(1,14),['Spade','Heart','Diamond','Club']))**

**# shuffle the cards**

**random.shuffle(deck)**

**# draw five cards**

**print("You got:")**

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

**print(deck[i][0], "of", deck[i][1])**

**7)Python Program to Implement Hybrid Inheritance**

**# Python program to demonstrate**

**# hybrid inheritance**

**class School:**

**def func1(self):**

**print("This function is in school.")**

**class Student1(School):**

**def func2(self):**

**print("This function is in student 1. ")**

**class Student2(School):**

**def func3(self):**

**print("This function is in student 2.")**

**class Student3(Student1, School):**

**def func4(self):**

**print("This function is in student 3.")**

**# Driver's code**

**object = Student3()**

**object.func1()**

**object.func2()**

**8) Method Overloading and overriding**

**#overloading**

**from multipledispatch import dispatch**

**@dispatch(int, int)**

**def sum(num1, num2):**

**print(num1+num2)**

**@dispatch(int, int, int)**

**def sum(num1, num2, num3):**

**print(num1+num2+num3)**

**@dispatch(float, float)**

**def sum(num1, num2):**

**print(num1+num2)**

**sum(3,4)**

**sum(5,7,8)**

**sum(57.43, 42.57)**

**#Overriding**

**class vehicle():**

**def \_\_init\_\_(self):**

**self.type = 'vehicle'**

**def print\_type(self):**

**print('My type is', self.type)**

**class car():**

**def \_\_init\_\_(self):**

**self.type = 'Car'**

**def print\_type(self):**

**print('My type is', self.type)**

**v1 = vehicle()**

**c1 = car()**

**v1.print\_type()**

**c1.print\_type()**

**9)Multithreading**

**from threading import Thread**

**def fun1():**

**for i in range(1,11):**

**if i%2 == 0:**

**continue**

**print(i)**

**def fun2():**

**for i in range(1,11):**

**if i%2 == 0:**

**print(i)**

**t1 =Thread(target=fun1())**

**t2 =Thread(target=fun2())**

**t1.start**

**t2.start**

**11)Implement Operations -> Create new file, write content, read, append , delete over text file using Python**

**f1 = open("temp.txt", "w")**

**f1.write("Hi \n My name is Piyush \n Bye \n")**

**f1 = open("temp.txt", "r")**

**data = f1.read()**

**print("Contents of the file: \n",data)**

**f1 = open("temp.txt", "a")**

**f1.write("Hi again \n Bye again")**

**f1 = open("temp.txt", "r")**

**data = f1.read()**

**print("Contents of the file: \n",data)**

**12)Different types of plots using Numpy and Matplotlib**

***# 1) matplotlib markers***

**import matplotlib.pyplot as plt**

**import numpy as np**

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

**plt.plot(ypoints, *marker* = 'o')**

**plt.show()**

***# 2) Bar plot***

***# importing matplotlib module***

**from matplotlib import pyplot as plt**

***# x-axis values***

**x = [5, 2, 9, 4, 7]**

***# Y-axis values***

**y = [10, 5, 8, 4, 2]**

***# Function to plot the bar***

**plt.bar(x,y)**

***# function to show the plot***

**plt.show()**

***# 3) line plot***

**from matplotlib import pyplot as plt**

***# x-axis values***

**x = [5, 2, 9, 4, 7]**

***# Y-axis values***

**y = [10, 5, 8, 4, 2]**

***# Function to plot***

**plt.plot(x,y)**

***# function to show the plot***

**plt.show()**

***# 4) histogram***

**from matplotlib import pyplot as plt**

***# Y-axis values***

**y = [10, 5, 8, 4, 2]**

***# Function to plot histogram***

**plt.hist(y)**

***# Function to show the plot***

**plt.show()**

***# 5) scatter plot***

**import matplotlib.pyplot as plt**

**x =[5, 7, 8, 7, 2, 17, 2, 9,**

**4, 11, 12, 9, 6]**

**y =[99, 86, 87, 88, 100, 86,**

**103, 87, 94, 78, 77, 85, 86]**

**plt.scatter(x, y, *c* ="black")**

***# To show the plot***

**plt.show()**