**LAB 1**

1. Write Python programs to print strings in the given manner:

a) Hello Everyone !!!

b) Hello

World

c) Hello

World

d) ‘ Rohit’ s date of birth is 12\05\1999’

**Code-:**

*a = "Hello Everyone !!!"*

*b = """Hello*

*World"""*

*c = """Hello*

*World"""*

*d = "'Rohit's date of birth is 12\05\1999'"*

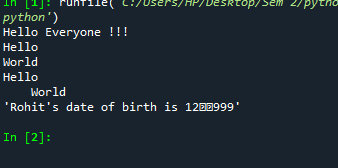
*print(a)*

*print(b)*

*print(c)*

*print(d)*

**Output-:**

****

1. Declare a string variable called x and assign it the value “Hello”.

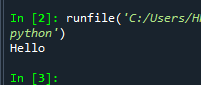
Print out the value of x

**Code-:**

*x = "Hello"*

*print(x)*

**Output-:**



1. Take different data types and print values using print function.

**Code-:**

*x = 12*

*y = "Hello"*

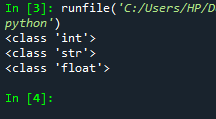
*z = 12.2*

*print(type(x))*

*print(type(y))*

*print(type(z))*

**Output-:**



1. Take two variable a and b. Assign your first name and last name. Print your Name after adding your First name and Last name together.

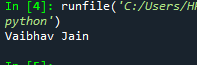
**Code-:**

*a = "Vaibhav "*

*b = "Jain"*

*print(a+b)*

**Output-:**

**

1. Declare three variables, consisting of your first name, your last name and Nickname. Write a program that prints out your first name, then your nickname in parenthesis and then your last name.

Example output : George ( woody ) Washington.

**Code-:**

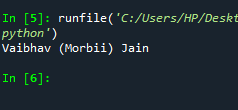
*a = "Vaibhav "*

*b = "(Morbii) "*

*c = "Jain"*

*print(a+b+c)*

**Output-:**

**

1. Declare and assign values to suitable variables and print in the following way :

NAME : NIKUNJ BANSAL

SAP ID : 500069944

DATE OF BIRTH : 13 Oct 1999

ADDRESS : UPES Bidholi Campus

Pincode : 248007

Programme : AI & ML Semester : 2

**Code-:**

*Name = "NIKUNJ BANSAL"*

*Sap = "500069944"*

*DOB = "13 Oct 1999"*

*Address = """UPES*

*Bidholi Campus*

*Pincode : 248007"""*

*Programme = "AI & ML"*

*Sem = "2"*

*print("NAME :",Name)*

*print("SAP ID :",Sap)*

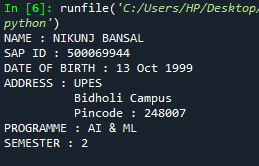
*print("DATE OF BIRTH :",DOB)*

*print("ADDRESS :",Address)*

*print("PROGRAMME :",Programme)*

*print("SEMESTER :",Sem)*

**Output-:**

**

**LAB 2**

1. Declare these variables (x, y and z) as integers. Assign a value of 9 to x, Assign a value of 7 to y, perform addition, multiplication, division and subtraction on these two variables and Print out the result.

**Code-:**

*#Declaring Variables*

*x = 9*

*y = 7*

*add = x + y*

*subtract = x - y*

*divide = x / y*

*multiply = x \* y*

*#Results*

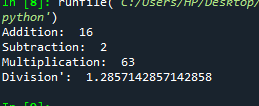
*print("Addition: ",add)*

*print("Subtraction: ",subtract)*

*print("Multiplication: ",multiply)*

*print("Division': ",divide)*

**Output-:**

**

1. Write a Program where the radius is taken as input to compute the area of a circle.

**Code-:**

*#Taking input for the radius*

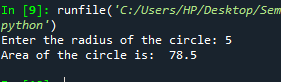
*radius = float(input("Enter the radius of the circle: "))*

*#Area*

*area = 3.14\*radius\*radius*

*print("Area of the circle is: ", area)*

**Output-:**

**

1. Write a Python program to solve (x+y)\*(x+y)

Test data : x = 4 , y = 3

**Code-:**

*x = 4*

*y = 3*

*result = (x+y)\*(x+y)*

*print("Result: ", result)*

**Output-:**

**

1. Write a program to compute the length of the hypotenuse (c) of a right triangle using Pythagoras theorem.

**Code-:**

*import math*

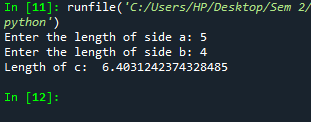
*a = float(input("Enter the length of side a: "))*

*b = float(input("Enter the length of side b: "))*

*c = math.sqrt(a\*a + b\*b)*

*print("Length of c: ", c)*

**Output-:**

**

1. Write a program to find simple interest.

**Code-:**

*#Taking input from the user*

*principal = float(input("Enter the principal amount: "))*

*rate = float(input("Enter the rate of interest: "))*

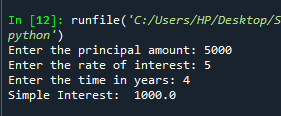
*time = float(input("Enter the time in years: "))*

*#Calculating simple interest*

*SI = (principal\*rate\*time)/100*

*print("Simple Interest: ", SI)*

**Output-:**

**

1. Write a program to find area of triangle when length of sides are given.

**Code-:**

*import math*

*# Input sides of the triangle*

*a = float(input("Enter the length of a: "))*

*b = float(input("Enter the length of b: "))*

*c = float(input("Enter the length of c: "))*

*# Calculate the semi-perimeter*

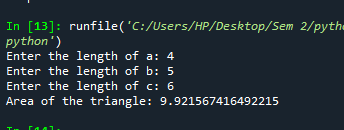
*s = (a + b + c) / 2*

*# Calculate the area*

*area = math.sqrt(s \* (s - a) \* (s - b) \* (s - c))*

*print("Area of the triangle:", area)*

**Output-:**

**

1. Write a program to convert given seconds into hours, minutes and remaining seconds.

**Code-:**

*# Input the seconds*

*sec = int(input("Enter the total seconds: "))*

*# Calculating hours, minutes, and remaining seconds*

*hours = sec // 3600*

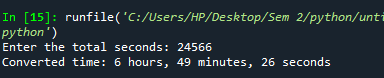
*minutes = (sec % 3600) // 60*

*rem = sec % 60*

*# Print out the result*

*print(f"Converted time: {hours} hours, {minutes} minutes, {rem} seconds")*

**Output-:**

**

1. Write a program to swap two numbers without taking additional variable.

**Code-:**

*#Swap*

*a = float(input("Enter the first number: "))*

*b = float(input("Enter the second number: "))*

*#Swapping*

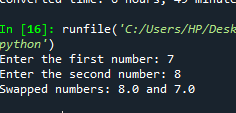
*a = a+b*

*b = a-b*

*a = a-b*

*print(f"Swapped numbers: {a} and {b}")*

**Output-:**

**

1. Write a program to find sum of first n natural numbers.

**Code-:**

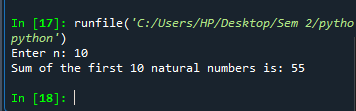
*# Input the value of n*

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

*sum = n \* (n + 1) // 2*

*print(f"Sum of the first {n} natural numbers is: {sum}")*

**Output-:**

****

1. Write a program to print truth table for bitwise operators( & , | and ^ operators)

**Code-:**

*a = 7*

*b = 8*

*#The truth table for bitwise AND (&) operator*

*print("Bitwise AND:")*

*print(f" {a} & {b} = {a & b}")*

*#The truth table for bitwise OR (|) operator*

*print("\nBitwise OR:")*

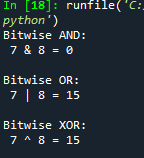
*print(f" {a} | {b} = {a|b}")*

*#The truth table for bitwise XOR (^) operator*

*print("\nBitwise XOR:")*

*print(f" {a} ^ {b} = {a^b}")*

**Output-:**

**

1. Write a program to find left shift and right shift values of a given number.

**Code-:**

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

*# Perform left shift and right shift operations*

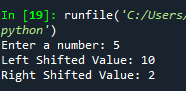
*left = num << 1*

*right = num >> 1*

*print("Left Shifted Value:", left)*

*print("Right Shifted Value:", right)*

**Output-:**

**

1. Using membership operator find whether a given number is in sequence (10,20,56,78,89).

**Code-:**

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

*# Check if the number is in the sequence*

*sequence = [10, 20, 56, 78, 89]*

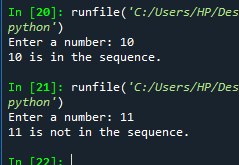
*if num in sequence:*

*print(f"{num} is in the sequence.")*

*else:*

*print(f"{num} is not in the sequence.")*

**Output-:**

**

1. Using membership operator find whether a given character is in a string.

**Code-:**

*char = input("Enter a character: ")*

*# Input a string*

*string = input("Enter a string: ")*

*# Check if the character is in the string*

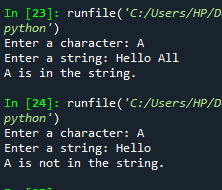
*if char in string:*

*print(f"{char} is in the string.")*

*else:*

*print(f"{char} is not in the string.")*

**Output-:**

**

**LAB 3**

1. Check whether given number is divisible by 3 and 5 both.

**Code-:**

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

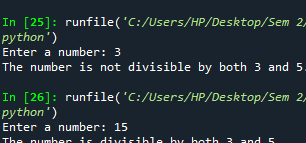
*if num % 3 == 0 and num % 5 == 0:*

*print("The number is divisible by both 3 and 5.")*

*else:*

*print("The number is not divisible by both 3 and 5.")*

**Output-:**

**

1. Check whether a given number is multiple of five or not.

**Code-:**

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

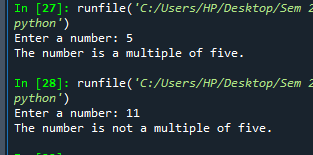
*if number % 5 == 0:*

*print("The number is a multiple of five.")*

*else:*

*print("The number is not a multiple of five.")*

**Output-:**



1. Find the greatest among two numbers. If numbers are equal than print “numbers are equal”.

**Code-:**

*num 1 = float(input("Enter the first number: "))*

*num 2 = float(input("Enter the second number: "))*

*if num1 > num2:*

*print(f"{num1} is the greatest.")*

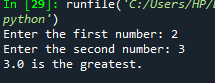
*elif num2 > num1:*

*print(f"{num2} is the greatest.")*

*else:*

*print("Numbers are equal.")*

**Output-:**

**

1. Find the greatest among three numbers assuming no two values are same.

**Code-:**

*a= float(input("Enter the first number: "))*

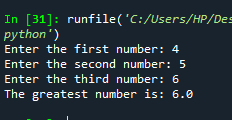
*b= float(input("Enter the second number: "))*

*c = float(input("Enter the third number: "))*

*maxn= max(a, b, c)*

*print(f"The greatest number is: {maxn")*

**Output-:**

**

1. Check whether the quadratic equation has real roots or imaginary roots. Display the roots.
2. Find whether a given year is a leap year or not.

**Code-:**

y*ear = int(input("Enter the year:"))*

*if year % 4 == 0:*

*if year % 100 == 0:*

*if year % 400 == 0:*

*print(year,"is a leap year.")*

*else:*

*print(year,"is not a leap year.")*

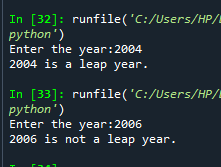
*else:*

*print(year,"is a leap year.")*

*else:*

*print(year,"is not a leap year.")*

**Output-:**

**

1. Print the grade sheet of a student for the given range of cgpa.

Scan marks of five subjects and calculate the percentage.

CGPA=percentage/10

CGPA range:

0 to 3.4 -> F

3.5 to 5.0->C+

5.1 to 6->B

6.1 to 7-> B+

7.1 to 8-> A

8.1 to 9->A+

9.1 to 10-> O (Outstanding)

**Code-:**

*# Input student details*

*name = input("Name: ")*

*roll = input("Roll Number: ")*

*SAP = input("SAPID: ")*

*sem = input("Sem: ")*

*course = input("Course: ")*

*# Input marks for five subjects*

*marks = {*

*"PDS": float(input("PDS: ")),*

*"Python": float(input("Python: ")),*

*"Chemistry": float(input("Chemistry: ")),*

*"English": float(input("English: ")),*

*"Physics": float(input("Physics: "))*

*}*

*#percentage*

*total = sum(marks.values())*

*percentage = (total / (len(marks) \* 100)) \* 100*

*# Calculate CGPA*

*cgpa = percentage / 10*

*if cgpa >= 9.1:*

*grade = "O (Outstanding)"*

*elif 8.1 <= cgpa < 9.1:*

*grade = "A+"*

*elif 7.1 <= cgpa < 8.1:*

*grade = "A"*

*elif 6.1 <= cgpa < 7.1:*

*grade = "B+"*

*elif 5.1 <= cgpa < 6.1:*

*grade = "B"*

*elif 3.5 <= cgpa < 5.1:*

*grade = "C+"*

*else:*

*grade = "F"*

*print("\nGradesheet")*

*print(f"Name: {name}")*

*print(f"Roll Number: {roll} SAPID: {SAP}")*

*print(f"Sem: {sem} Course: {course}")*

*print("Subject name: Marks")*

*for subject, marks in marks.items():*

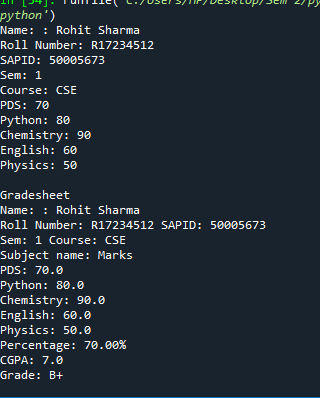
*print(f"{subject}: {marks}")*

*print(f"Percentage: {percentage:.2f}%")*

*print(f"CGPA: {cgpa:.1f}")*

*print(f"Grade: {grade}")*

**Output-:**



**LAB 4**

1. Find a factorial of given number.

**Code-:**

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

*factorial = 1*

*if num < 0:*

*print("Factorial is not defined for negative numbers.")*

*elif num == 0 or num == 1:*

*print("The factorial of", num, "is 1")*

*else:*

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

*factorial \*= i*

*print("The factorial of", num, "is:", factorial)*

1. Find whether the given number is Armstrong number.

**Code-:**

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

*temp = num*

*order = len(str(num))*

*sum = 0*

*while temp > 0:*

*digits = temp%10*

*sum += digits\*\*order*

*temp //= 10*

*if num == sum:*

*print(f"{num} is an Armstrong number.")*

*else:*

*print(f"{num} is not an Armstrong number")*

1. Print Fibonacci series up to given term.

**Code-:**

*num = int(input("Enter the number of terms: "))*

*a, b = 0, 1*

*print("Fibonnaci Series:")*

*for \_ in range(num):*

*print(a, end=" ")*

*a, b = b, a+b*

1. Write a program to find if given number is prime number or not.

**Code-:**

*num = int(input("Enter the number to check: "))*

*if num > 1:*

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

*if num % i == 0:*

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

*break*

*else:*

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

*else:*

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

1. Check whether given number is palindrome or not.

**Code-:**

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

*temp = num*

*reverse = 0*

*while num > 0:*

*digit = num % 10*

*reverse = reverse \* 10 + digit*

*num = num // 10*

*if temp == reverse:*

*print(f"{temp} is a palindrome.")*

*else:*

*print(f"{temp} is not a palindrome.")*

1. Write a program to print sum of digits.

**Code-:**

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

*sum = 0*

*while num > 0:*

*digit = num % 10*

*sum += digit*

*num = num // 10*

*print(f"The sum of digits is: {sum}")*

1. Count and print all numbers divisible by 5 or 7 between 1 to 100.

**Code-:**

*count = 0*

*print("Numbers divisible by 5 or 7 between 1 to 100:")*

*for num in range(1, 101):*

*if num % 5 == 0 or num % 7 == 0:*

*count += 1*

*print(num)*

*print("Total count:", count)*

1. Convert all lower cases to upper case in a string.

**Code-:**

*string = input("Enter a string:")*

*uppercase = string.upper()*

*print("Uppercase string:", uppercase)*

1. Print all prime numbers between 1 and 100.

**Code-:**

*print("Prime numbers between 1 and 100 are:")*

*for num in range(1,101):*

*if num > 1:*

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

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

*break*

*else:*

*print(num)*

1. Print the table for a given number:

5 \* 1 = 5

5 \* 2 = 10………..

**Code-:**

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

*print(f"Multiplication table for {num}:")*

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

*result = num \* i*

*print(f"{num} \* {i} = {result}")*

**LAB 5**

1. Write a program to count and display the number of capital letters in a given string.

**Code-:**

*def count\_capital\_letters(input\_string):*

*count = 0*

*for char in input\_string:*

*if char.isupper():*

*count += 1*

*return count*

*def main():*

*input\_string = input("Enter a string: ")*

*capital\_count = count\_capital\_letters(input\_string)*

*print("Number of capital letters:", capital\_count)*

1. Count total number of vowels in a given string.

**Code-:**

*def count\_vowels(input\_string):*

*vowels = 'aeiouAEIOU'*

*vowel\_count = 0*

*for char in input\_string:*

*if char in vowels:*

*vowel\_count += 1*

*return vowel\_count*

*def main():*

*input\_string = input("Enter a string: ")*

*vowel\_count = count\_vowels(input\_string)*

*print("Total number of vowels:", vowel\_count)*

1. Input a sentence and print words in separate lines.

**Code-:**

*def print\_words\_separate\_lines(sentence):*

*words = sentence.split()*

*for word in words:*

*print(word)*

*def main():*

*input\_sentence = input("Enter a sentence: ")*

*print("Words in separate lines:")*

*print\_words\_separate\_lines(input\_sentence)*

1. WAP to enter a string and a substring.

You have to print the number of times that the substring occurs in the given string.

String traversal will take place from left to right, not from right to left.

Sample Input ABCDCDC

CDC

Sample Output 2

1. Given a string containing both upper and lower case alphabets. Write a Python program to count the number of occurrences of each alphabet (case insensitive) and display the same.

Sample Input ABaBCbGc

Sample Output

2A

3B

2C

1G

1. Program to count number of unique words in a given sentence using sets.

**Code-:**

*def count\_unique\_words(sentence):*

*# Split the sentence into words*

*words = sentence.split()*

*# Create a set to store unique words*

*unique\_words = set(words)*

*# Return the count of unique words*

*return len(unique\_words)*

*def main():*

*input\_sentence = input("Enter a sentence: ")*

*unique\_word\_count = count\_unique\_words(input\_sentence)*

*print("Number of unique words:", unique\_word\_count)*

1. Create 2 sets s1 and s2 of n fruits each by taking input from user and find:
   1. Fruits which are in both sets s1 and s2
   2. Fruits only in s1 but not in s2
   3. Count of all fruits from s1 and s2

**Code-:**

*def main():*

*n = int(input("Enter the number of fruits for each set: "))*

*# Input for set s1*

*print("\nEnter fruits for set s1:")*

*s1 = set()*

*for i in range(n):*

*fruit = input(f"Enter fruit {i+1}: ")*

*s1.add(fruit.lower()) # Convert to lowercase to make it case-insensitive*

*# Input for set s2*

*print("\nEnter fruits for set s2:")*

*s2 = set()*

*for i in range(n):*

*fruit = input(f"Enter fruit {i+1}: ")*

*s2.add(fruit.lower()) # Convert to lowercase to make it case-insensitive*

*# Fruits which are in both sets s1 and s2*

*common\_fruits = s1.intersection(s2)*

*print("\nFruits which are in both sets s1 and s2:")*

*print(common\_fruits)*

*# Fruits only in s1 but not in s2*

*unique\_to\_s1 = s1.difference(s2)*

*print("\nFruits only in set s1 but not in set s2:")*

*print(unique\_to\_s1)*

*# Count of all fruits from s1 and s2*

*total\_fruits = len(s1.union(s2))*

*print("\nCount of all fruits from s1 and s2:", total\_fruits)*

1. Take two sets and apply various set operations on them :

S1 = {Red ,yellow, orange , blue }

S2 = {violet, blue , purple}

**Code-:**

*def main():*

*S1 = {"Red", "Yellow", "Orange", "Blue"}*

*S2 = {"Violet", "Blue", "Purple"}*

*# Union*

*print("Union of S1 and S2:", S1.union(S2))*

*# Intersection*

*print("Intersection of S1 and S2:", S1.intersection(S2))*

*# Difference (elements in S1 but not in S2)*

*print("Difference of S1 and S2:", S1.difference(S2))*

*# Subset*

*print("Is S1 a subset of S2?", S1.issubset(S2))*

*# Superset*

*print("Is S1 a superset of S2?", S1.issuperset(S2))*

*# Disjoint*

*print("Are S1 and S2 disjoint?", S1.isdisjoint(S2))*

**LAB 6**

1. Create a tuple to store n numeric values and find average of all values.

**Code-:**

*def calculate\_average(numeric\_values):*

*# Calculate the sum of all values in the tuple*

*total\_sum = sum(numeric\_values)*

*# Calculate the average*

*average = total\_sum / len(numeric\_values)*

*return average*

*def main():*

*n = int(input("Enter the number of values: "))*

*# Input the numeric values into a tuple*

*numeric\_values = tuple(float(input(f"Enter value {i+1}: ")) for i in range(n))*

*# Calculate the average of all values*

*average = calculate\_average(numeric\_values)*

*print("Average of all values:", average)*

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

*main()*

1. WAP to input a list of scores for N students in a list data type. Find the score of the

runner-up and print the output.

Sample Input

N = 5

Scores= 2 3 6 6 5

Sample output

5

Note: Given list is [2, 3, 6, 6, 5]. The maximum score is 6, second maximum is 5.

Hence, we print 5 as the runner-up score.

**Code-:**

*def find\_runner\_up\_score(scores):*

*# Sort the list of scores in descending order*

*sorted\_scores = sorted(scores, reverse=True)*

*# Find the second highest score*

*runner\_up\_score = None*

*for score in sorted\_scores:*

*if score < sorted\_scores[0]:*

*runner\_up\_score = score*

*break*

*return runner\_up\_score*

*def main():*

*N = int(input("Enter the number of students: "))*

*scores = list(map(int, input("Enter the scores separated by space: ").split()))*

*if len(scores) != N:*

*print("Number of scores entered does not match the specified number of students.")*

*return*

*runner\_up\_score = find\_runner\_up\_score(scores)*

*if runner\_up\_score is not None:*

*print("Runner-up score:", runner\_up\_score)*

*else:*

*print("There is no runner-up score.")*

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

*main()*

1. Scan n values in range 0-3 and print the number of times each value has occurred.

**Code-:**

*def count\_occurrences(n):*

*occurrences = {0: 0, 1: 0, 2: 0, 3: 0} # Initialize dictionary to count occurrences*

*# Input values and count occurrences*

*for \_ in range(n):*

*value = int(input("Enter a value between 0 and 3: "))*

*if value in occurrences:*

*occurrences[value] += 1*

*else:*

*print("Invalid value entered. Please enter a value between 0 and 3.")*

*return occurrences*

*def main():*

*n = int(input("Enter the number of values: "))*

*if n <= 0:*

*print("Number of values must be greater than zero.")*

*return*

*occurrences = count\_occurrences(n)*

*# Print the number of times each value has occurred*

*print("\nOccurrences of each value:")*

*for value, count in occurrences.items():*

*print(f"{value}: {count} times")*

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

*main()*

**LAB 7**

1. Write a Python function to find the maximum and minimum numbers from a sequence of numbers.

(Note: Do not use built-in functions.)

**Code-:**

*#Find maximum and minimum numbers from the sequence of number*

*def max\_min(data):*

*l\_num = data[0]*

*s\_num = data[0]*

*for num in data:*

*if num > l\_num:*

*l\_num = num*

*elif num < s\_num:*

*s\_num = num*

*return l\_num, s\_num*

*print(max\_min([0, 10, 15, 40, -5, 42, 17, 28, 75]))*

1. Write a Python function that takes a positive integer and returns the sum of the cube of all the positive integers smaller than the specified number.

**Code-:**

#*sum of the cube of the all positive integers smaller by the specified number*

*def sum\_of\_cubes(n):*

*n = n-1*

*total = 0*

*while n > 0:*

*total = total + (n \* n \* n)*

*n = n- 1*

*return total*

*print("Sum of cubes:", sum\_of\_cubes(4))*

1. Write a Python function to print 1 to n using recursion.

(Note: Do not use loop)

**Code-:**

*# To print 1 to n using recursion*

*def printNums(n):*

*if n > 0:*

*printNums(n - 1)*

*print(n, end=' ')*

*n = 50*

*printNums(n)*

1. Write a recursive function to print Fibonacci series upto n terms.

**Code-:**

*# To print Fibonacci series upto n terms.*

*def fibonacci(n):*

*if n <= 1:*

*return n*

*else:*

*return(fibonacci(n-1) + fibonacci(n-2))*

*n = 10*

*print("Fibonacci sequence:")*

*for i in range(n):*

*print(fibonacci(i))*

1. Write a lambda function to find volume of cone.

**Code-:**

*# To find volume of cone.*

*import math*

*cone\_volume = lambda radius, height: (1/3) \* math.pi \* radius\*\*2 \* height*

*print(cone\_volume(2,3))*

*# lambda function which gives tuple of max and min from a list*

*my\_list = [10, 5, 20, 8, 15]*

*a=[]*

*min\_value = lambda lst: min(lst)*

*max\_value = lambda lst: max(lst)*

*max\_result = max\_value(my\_list)*

*min\_result = min\_value(my\_list)*

*a.append(min\_result)*

*a.append(max\_result)*

*b=tuple(a)*

*print(b)*

1. Write a lambda function which gives tuple of max and min from a list.

Sample input: [10, 6, 8, 90, 12, 56]

Sample output: (90,6)

**Code-:**

*# lambda function which gives tuple of max and min from a list*

*my\_list = [10, 5, 20, 8, 15]*

*a=[]*

*min\_value = lambda lst: min(lst)*

*max\_value = lambda lst: max(lst)*

*max\_result = max\_value(my\_list)*

*min\_result = min\_value(my\_list)*

*a.append(min\_result)*

*a.append(max\_result)*

*b=tuple(a)*

*print(b)*

1. Write functions to explain mentioned concepts:
   1. Keyword argument
   2. Default argument
   3. Variable length argument

**Code-:**

*#Write functions to explain mentioned concepts:*

*#a. Keyword argument*

*def greet(name, message):*

*print(f"Hello, {name}! {message}")*

*greet(name="Alice", message="How are you?")*

*#b. Default argument*

*def greet(name, message="How are you?"):*

*print(f"Hello, {name}! {message}")*

*greet("Bob")*

*greet("Alice", "Nice to see you!")*

*#c. Variable length argument*

*def calculate\_sum(\*args):*

*total = 0*

*for num in args:*

*total += num*

*return total*

*print(calculate\_sum(1, 2, 3))*

*print(calculate\_sum(5, 10, 15, 20))*

**LAB 8**

1. Add few names, one name in each row, in “name.txt file”.
   1. Count no of names
   2. Count all names starting with vowel
   3. Find longest name

**Code-:**

*#1. Add few names, one name in each row, in “name.txt file”.*

*# a. Count no of names*

*# b. Count all names starting with vowel*

*# c. Find longest name*

*def count\_names(filename):*

*with open(filename, 'r') as file:*

*names = file.readlines()*

*return len(names)*

*def count\_names\_starting\_with\_vowel(filename):*

*vowels = "aeiouAEIOU"*

*with open(filename, 'r') as file:*

*names = file.readlines()*

*count = sum(1 for name in names if name.strip()[0] in vowels)*

*return count*

*def find\_longest\_name(filename):*

*with open(filename, 'r') as file:*

*names = file.readlines()*

*longest\_name = max(names, key=lambda x: len(x.strip()))*

*return longest\_name.strip()*

*def main():*

*filename = "myfile.txt"*

*total\_names = count\_names(filename)*

*print("Total number of names:", total\_names)*

*names\_starting\_with\_vowel = count\_names\_starting\_with\_vowel(filename)*

*print("Number of names starting with a vowel:", names\_starting\_with\_vowel)*

*longest\_name = find\_longest\_name(filename)*

*print("Longest name:", longest\_name)*

*if name == "main":*

*main()*

1. Store integers in a file.
   1. Find the max number
   2. Find average of all numbers
   3. Count number of numbers greater than 100

**Code-:**

*#2. Store integers in a file.*

*# a. Find the max number*

*# b. Find average of all numbers*

*# c. Count number of numbers greater than 100*

*def read\_integers(filename):*

*with open(filename, 'r') as file:*

*integers = [int(line.strip()) for line in file]*

*return integers*

*def find\_max\_number(integers):*

*return max(integers)*

*def calculate\_average(integers):*

*return sum(integers) / len(integers)*

*def count\_numbers\_greater\_than\_100(integers):*

*return sum(1 for num in integers if num > 100)*

*def main():*

*filename = 'myfile.txt'*

*integers = read\_integers(filename)*

*if integers:*

*print("Max number:", find\_max\_number(integers))*

*print("Average of all numbers:", calculate\_average(integers))*

*print("Count of numbers greater than 100:", count\_numbers\_greater\_than\_100(integers))*

*else:*

*print("No integers found in the file.")*

*if name == "main":*

*main()*

1. Assume a file city.txt with details of 5 cities in given format (cityname population(in lakhs) area(in sq KM) ): Example: Dehradun 5.78 308.20 Delhi 190 1484 ……………

Open file city.txt and read to:

* 1. Display details of all cities
  2. Display city names with population more than 10Lakhs
  3. Display sum of areas of all cities

**Code-:**

*3. Assume a file city.txt with details of 5 cities in given format (cityname population(in lakhs) area(in sq KM) ):*

*# Example:*

*# Dehradun 5.78 308.20*

*# Delhi 190 1484*

*# ……………*

*# Open file city.txt and read to:*

*# a. Display details of all cities*

*# b. Display city names with population more than 10Lakhs*

*# c. Display sum of areas of all cities*

*def details(filename):*

*cities = []*

*with open(filename, 'r') as file:*

*for line in file:*

*city\_details = line.strip().split()*

*city\_name = city\_details[0]*

*population = float(city\_details[1])*

*area = float(city\_details[2])*

*cities.append((city\_name, population, area))*

*return cities*

*def display\_all\_cities(cities):*

*print("City Details:")*

*for city in cities:*

*print("City:", city[0])*

*print("Population (in lakhs):", city[1])*

*print("Area (in sq KM):", city[2])*

*print()*

*def display\_cities\_population\_more\_than\_10\_lakhs(cities):*

*print("Cities with population more than 10 Lakhs:")*

*for city in cities:*

*if city[1] > 10:*

*print(city[0])*

*def calculate\_sum\_of\_areas(cities):*

*total\_area = sum(city[2] for city in cities)*

*print("Sum of areas of all cities:", total\_area)*

*def main():*

*filename = 'city.txt' # Change this to the name of your file*

*cities = details(filename)*

*if cities:*

*display\_all\_cities(cities)*

*display\_cities\_population\_more\_than\_10\_lakhs(cities)*

*calculate\_sum\_of\_areas(cities)*

*else:*

*print("No city details found in the file.")*

*if name == "main":*

*main()*

1. Input two values from user where the first line contains N, the number of test cases. The next N lines contain the space separated values of a and b. Perform integer division and print a/b. Handle exception in case of ZeroDivisionError or ValueError.

Sample input 1 0 2 $ 3 1

Sample Output : Error Code: integer division or modulo by zero Error Code: invalid literal for int() with base 10: '$' 3

**Code-:**

*#4. Input two values from user where the first line contains N, the number of test cases. The next N lines contain the*

*# space separated values of a and b. Perform integer division and print a/b. Handle exception in case of*

*# ZeroDivisionError or ValueError.*

*def perform\_division(a, b):*

*try:*

*result = int(a) // int(b)*

*print(result)*

*except ValueError as ve:*

*print("Error Code:", ve)*

*except ZeroDivisionError as zde:*

*print("Error Code:", zde)*

*if name == "main":*

*N = int(input("Enter the number of test cases: "))*

*for \_ in range(N):*

*a, b = input().split()*

*perform\_division(a, b)*

*#5. Create multiple suitable exceptions for a file handling program.*

*def read(filename):*

*try:*

*with open(filename, 'r') as file:*

*content = file.read()*

*print(content)*

*except FileNotFoundError:*

*print(f"Error: File '{filename}' not found.")*

*except PermissionError:*

*print(f"Error: Permission denied to open '{filename}'.")*

*except IsADirectoryError:*

*print(f"Error: '{filename}' is a directory, not a file.")*

*except UnicodeDecodeError:*

*print(f"Error: Unable to decode file '{filename}'. It may not be a text file.")*

*except Exception as e:*

*print(f"An unexpected error occurred: {e}")*

*except ValueError as ve:*

*print(f"Error code:",ve)*

*if name == "main":*

*filename = input("Enter the name of the file to read: ")*

*read(filename)*

1. Create multiple suitable exceptions for a file handling program.

**Code-:**

*#4. Input two values from user where the first line contains N, the number of test cases. The next N lines contain the*

*# space separated values of a and b. Perform integer division and print a/b. Handle exception in case of*

*# ZeroDivisionError or ValueError.*

*def perform\_division(a, b):*

*try:*

*result = int(a) // int(b)*

*print(result)*

*except ValueError as ve:*

*print("Error Code:", ve)*

*except ZeroDivisionError as zde:*

*print("Error Code:", zde)*

*if name == "main":*

*N = int(input("Enter the number of test cases: "))*

*for \_ in range(N):*

*a, b = input().split()*

*perform\_division(a, b)*

*#5. Create multiple suitable exceptions for a file handling program.*

*def read(filename):*

*try:*

*with open(filename, 'r') as file:*

*content = file.read()*

*print(content)*

*except FileNotFoundError:*

*print(f"Error: File '{filename}' not found.")*

*except PermissionError:*

*print(f"Error: Permission denied to open '{filename}'.")*

*except IsADirectoryError:*

*print(f"Error: '{filename}' is a directory, not a file.")*

*except UnicodeDecodeError:*

*print(f"Error: Unable to decode file '{filename}'. It may not be a text file.")*

*except Exception as e:*

*print(f"An unexpected error occurred: {e}")*

*except ValueError as ve:*

*print(f"Error code:",ve)*

*if name == "main":*

*filename = input("Enter the name of the file to read: ")*

*read(filename)*

**LAB 9**

1. Create a class of student (name, sap id, marks[phy,chem,maths] ). Create 3 objects by taking inputs from the user and display details of all students.
2. Add constructor in the above class to initialize student details of n students and implement following methods:
   1. Display() student details
   2. Find Marks\_percentage() of each student
   3. Display result() [Note: if marks in each subject >40% than Pass else Fail*]*

**Code-:**

*class Student:*

*def init(self, name, sap\_id, marks):*

*self.name = name*

*self.sap\_id = sap\_id*

*self.marks = marks*

*def display\_details(self):*

*print("Name:", self.name)*

*print("SAP ID:", self.sap\_id)*

*print("Physics Marks:", self.marks['physics'])*

*print("Chemistry Marks:", self.marks['chemistry'])*

*print("Mathematics Marks:", self.marks['mathematics'])*

*print()*

*if name == "main":*

*students = []*

*for i in range(3):*

*name = input("Enter student name: ")*

*sap\_id = input("Enter SAP ID: ")*

*physics\_marks = float(input("Enter Physics marks: "))*

*chemistry\_marks = float(input("Enter Chemistry marks: "))*

*mathematics\_marks = float(input("Enter Mathematics marks: "))*

*marks = {'physics': physics\_marks,'chemistry': chemistry\_marks,'mathematics': mathematics\_marks }*

*student = Student(name, sap\_id, marks)*

*students.append(student)*

*print("\nDetails of all students:")*

*for student in students:*

*student.display\_details()*

1. Create programs to implement different types of inheritances.

Single Inheritance

**Code-:**

*class Parent:*

*def parent\_method(self):*

*print("Parent's method")*

*class Child(Parent):*

*def child\_method(self):*

*print("Child's method")*

Multiple Inheritance

*class Parent1:*

*def method1(self):*

*print("Parent 1's method")*

*class Parent2:*

*def method2(self):*

*print("Parent 2's method")*

*class ChildMultiple(Parent1, Parent2):*

*def child\_method(self):*

*print("Child's method")*

Multilevel Inheritance

*class Grandparent:*

*def grandparent\_method(self):*

*print("Grandparent's method")*

*class ParentMultilevel(Grandparent):*

*def parent\_method(self):*

*print("Parent's method")*

*class ChildMultilevel(ParentMultilevel):*

*def child\_method(self):*

*print("Child's method")*

Hierarchical Inheritance

*class ParentHierarchical:*

*def parent\_method(self):*

*print("Parent's method")*

*class Child1Hierarchical(ParentHierarchical):*

*def child1\_method(self):*

*print("Child1's method")*

*class Child2Hierarchical(ParentHierarchical):*

*def child2\_method(self):*

*print("Child2's method")*

# Single Inheritance

*print("Single Inheritance:")*

*child\_obj = Child()*

*child\_obj.parent\_method()*

*child\_obj.child\_method()*

# Multiple Inheritance

*print("\nMultiple Inheritance:")*

*child\_multiple\_obj = ChildMultiple()*

*child\_multiple\_obj.method1()*

*child\_multiple\_obj.method2()*

*child\_multiple\_obj.child\_method()*

# Multilevel Inheritance

*print("\nMultilevel Inheritance:")*

*child\_multilevel\_obj = ChildMultilevel()*

*child\_multilevel\_obj.grandparent\_method()*

*child\_multilevel\_obj.parent\_method()*

*child\_multilevel\_obj.child\_method()*

# Hierarchical Inheritance

*print("\nHierarchical Inheritance:")*

*child1\_hierarchical\_obj = Child1Hierarchical()*

*child2\_hierarchical\_obj = Child2Hierarchical()*

*child1\_hierarchical\_obj.parent\_method()*

*child1\_hierarchical\_obj.child1\_method()*

*child2\_hierarchical\_obj.parent\_method()*

*child2\_hierarchical\_obj.child2\_method()*

1. Create a class to implement method Overriding.

**Code-:**

*class Parent:*

*def method(self):*

*print("Parent's method")*

*class Child(Parent):*

*def method(self):*

*print("Child's overridden method")*

*par\_obj = Parent()*

*child\_obj = Child()*

*par\_obj.method()*

*child\_obj.method()*

5. Create a class for operator overloading which adds two Point Objects where Point has x & y values

e.g. if

P1(x=10,y=20)

P2(x=12,y=15)

P3=P1+P2 => P3(x=22,y=35)

**Code-:**

*class Point:*

*def init(self, x, y):*

*self.x = x*

*self.y = y*

*def add(self, other):*

*if isinstance(other, Point): #subclass*

*new\_x = self.x + other.x*

*new\_y = self.y + other.y*

*return Point(new\_x, new\_y)*

*else:*

*raise TypeError("Unsupported operand type(s) for +: 'Point' and '{}'".format(type(other)))*

*def str(self):*

*return "Point(x={}, y={})".format(self.x, self.y)*

*P1 = Point(25, 60)*

*P2 = Point(92, 10)*

*P3 = P1 + P2*

*print("P1:", P1)*

*print("P2:", P2)*

*print("P3:",P3)*

**LAB 10**

1. Create numpy array to find sum of all elements in an array.

**Code-:**

*import numpy as np*

*arr = np.array([1, 2, 3, 4, 5])*

*sum\_of\_elements = np.sum(arr)*

*print(f"Sum of all elements: {sum\_of\_elements}")*

**Output-:**

**

1. Create numpy array of (3,3) dimension. Now find sum of all rows & columns individually. Also find 2nd maximum element in the array

**Code-:**

*import numpy as np*

*matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])*

*sum\_of\_rows = np.sum(matrix, axis=1)*

*sum\_of\_columns = np.sum(matrix, axis=0)*

*second\_max\_element = np.partition(matrix.flatten(), -2)[-2]*

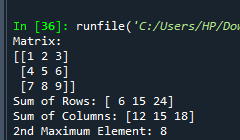
*print(f"Matrix:\n{matrix}")*

*print(f"Sum of Rows: {sum\_of\_rows}")*

*print(f"Sum of Columns: {sum\_of\_columns}")*

*print(f"2nd Maximum Element: {second\_max\_element}")*

**Output-:**

****

1. Perform Matrix multiplication of any 2 n\*n matrices.

**Code-:**

*import numpy as np*

*matrix1 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])*

*matrix2 = np.array([[9, 8, 7], [6, 5, 4], [3, 2, 1]])*

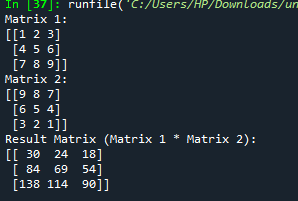
*result\_matrix = np.dot(matrix1, matrix2)*

*print(f"Matrix 1:\n{matrix1}")*

*print(f"Matrix 2:\n{matrix2}")*

*print(f"Result Matrix (Matrix 1 \* Matrix 2):\n{result\_matrix}")*

**Output-:**

**

1. Write a Pandas program to get the powers of an array values element-wise.

**Code-:**

*import pandas as pd*

*data = {'X': [78, 85, 96, 80, 86], 'Y': [84, 94, 89, 83, 86], 'Z': [86, 97, 96, 72, 83]}*

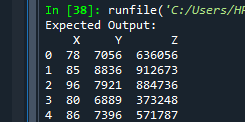
*df = pd.DataFrame(data)*

*powers\_df = df.pow([1, 2, 3])*

*print("Expected Output:")*

*print(powers\_df)*

**Output-:**

****

1. Write a Pandas program to get the first 3 rows of a given DataFrame.

**Code-:**

*import pandas as pd*

*import numpy as np*

*exam\_data = {'name': ['Anastasia', 'Dima', 'Katherine', 'James', 'Emily', 'Michael', 'Matthew', 'Laura', 'Kevin', 'Jonas'],*

*'score': [12.5, 9, 16.5, np.nan, 9, 20, 14.5, np.nan, 8, 19],*

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

*'qualify': ['yes', 'no', 'yes', 'no', 'no', 'yes', 'yes', 'no', 'no', 'yes']}*

*labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']*

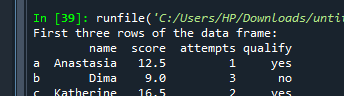
*df = pd.DataFrame(exam\_data, index=labels)*

*first\_three\_rows = df.head(3)*

*print("First three rows of the data frame:")*

*print(first\_three\_rows)*

**Output-:**

**

1. Write a Pandas program to find and replace the missing values in a given DataFrame which do not have any valuable information.

**Code-:**

*import pandas as pd*

*import numpy as np*

*data = {'Name': ['John', 'Alice', 'Bob', np.nan, 'Eve'],*

*'Age': [25, np.nan, 30, np.nan, 22],*

*'Salary': [50000, 60000, np.nan, np.nan, 55000]}*

*df = pd.DataFrame(data)*

*df\_filled = df.fillna(-1)*

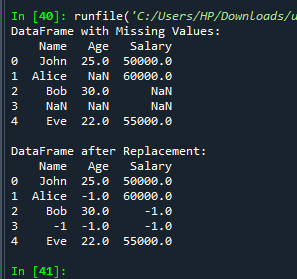
*print("DataFrame with Missing Values:")*

*print(df)*

*print("\nDataFrame after Replacement:")*

*print(df\_filled)*

**Output-:**

****

1. Create a program to demonstrate different visual forms using Matplotlib.

**Code-:**

*import matplotlib.pyplot as plt*

*import numpy as np*

*x = np.linspace(0, 10, 100)*

*y1 = np.sin(x)*

*y2 = np.cos(x)*

*plt.figure(figsize=(8, 4))*

*plt.subplot(2, 2, 1)*

*plt.plot(x, y1, label='sin(x)')*

*plt.plot(x, y2, label='cos(x)')*

*plt.title('Line Plot')*

*plt.legend()*

*plt.subplot(2, 2, 2)*

*plt.scatter(x, y1, label='sin(x)')*

*plt.scatter(x, y2, label='cos(x)')*

*plt.title('Scatter Plot')*

*plt.legend()*

*categories = ['Category A', 'Category B', 'Category C']*

*values = [20, 35, 25]*

*plt.subplot(2, 2, 3)*

*plt.bar(categories, values)*

*plt.title('Bar Chart')*

*data = np.random.randn(1000)*

*plt.subplot(2, 2, 4)*

*plt.hist(data, bins=30, edgecolor='black')*

*plt.title('Histogram')*

*plt.tight\_layout()*

*plt.show()*

**Output-:**

