PYTHON [CSE3011] LAB MANUAL

Submitted by

Shreyansh Vijayvargiya [21BCE11289]

To

Dr. Bhupendera Panchal

in partial fulfillment of the requirements for the degree of

Bachlore of Engineering and Technology

VIT Bhopal University
Bhopal
Madhya,pradhesh

MAY,24

INDEX

Topics	S. No.	List of Programs
1. Numeric Data Types	1.1	Write a python program to print first n prime numbers.
	1.2	Write a python program to multiply matrices.
	1.3	Write a python program which accepts the radius of a circle from user and computes the area (use math module)
2. Strings	2.1	Write python program to let user enter some data in string and then verify data and print welcome to user
	2.2	Write a Python program that checks whether a passed string is a palindrome or not.
	2.3	Write a Python program to count the occurrences of each word for given string.
	2.4	Write a Python function to reverse a string. Sample String: "1234abcd" Expected Output: "dcba4321"
	2.5	Write a Python program to remove the characters which have odd index values of a given string.
	2.6	Write python program to take command line arguments (word count).
3. Operators	3.1	Write the output of the following Python program: a = 10 b = 4 print (a & b) print (a | b) print (~a) print (a ^ b) print (a >> 2) print (a << 2)
	3.2	Write a program to create a menu with the following options and accepts users input and perform the operation accordingly: i) Addition ii) Subtraction iii) Multiplication iv) Division
4. Conditional and Looping	4.1	Write a python program to print a number is even/odd using if-else.
	4.2	Write a python program to find largest number among three numbers
	4.3	Write a python Program to read a number and display corresponding day using if_elif_else?
	4.4	Write a Python program to get the Fibonacci series between 0 to 50.
	4.5	Write python program to print list of numbers using range and for loop.

	4.6	Write a Python program to construct the following pattern, using a nested for loop. *

5. Lists & Tuples	5.1	Create a list and perform the following methods 1) insert() 2) remove() 3) append() 4) len() 5) pop() 6)clear()
	5.2	Create a tuple and perform the following methods 1) Add items 2) len() 3) check for item in tuple4)Access items
	5.3	Write a Python program to sum all the items in a list.
	5.4	Write a Python function that takes two lists and returns true if they are equal otherwise false.
	5.5	Write python program to store strings in list and then print them
6. Dictionary	6.1	Create a dictionary and apply the following methods 1) Print the dictionary items 2) access items 3) useget() 4)change values 5) use len()
	6.2	Write a Python script to check if a given key already exists in a dictionary.
	6.3	Write a Python script to sort (ascending and descending) a dictionary by value.
7. Functions	7.1	Write python program in which a function is defined and calling that function prints 'Hello World'.
	7.2	Write python program in which a function (with single string parameter) is defined and calling that function prints the string parameters given to function.
	7.3	Write a python program to find factorial of a given number using functions
	7.4	Write a Python function to calculate the factorial of a number (a nonnegative integer). The function accepts the number as an argument.

8. Regular Expression	8.1	Write a Python program to check the validity of passwords input by users using regular expression. Validations are: − At least 1 letter between [a-z] and 1 letter between [A-Z] − At least 1 number between [0-9] − At least 1 character from [$#@] − Minimum length 6 characters − Maximum length 16 characters
9. Class & Object	9.1	Write python program in which a class is defined, then create object of that class and call simple 'print function' defined in class.
	9.2	Write a python Program to call data member and function using classes and objects
	9.3	Write a Python class Employee with attributes like emp_name, emp_id, emp_salary, emp_department and methods like calculate_salary, and print_details. Use 'calculate_salary' method takes two arguments: salary and hours_worked. If the number of hours worked is more than 50, the method computes overtime and adds it to the salary. Overtime is calculated as following formula: overtime = hours_worked – 50 overtime amount = (overtime * (salary / 50)) Use 'print_details' method to print the details of employee. Consider the sample data: "Adams " "E7876" 50000 "Accounting"
		"Jones" "E7499" 45000 "Research" "Martin" "E7900" 50000 "Sales" "Smith" "E7698" 55000 "Operations"
	9.4	Write a python program to demonstrate access specifiers.
	9.5	Write a python program to apply polymorphism.
	9.6	Write a python program to demonstrate inheritance.
	9.7	Write a python program for method overriding.
	9.8	Write a python program to define abstraction.
	9.9	Write a python program to demonstrate Interface.
10. Exception Handling	10.1	Demonstrate a python code to print try, except and finally block statements
	10.2	Demonstrate a python code to implement abnormal termination
	10.3	Write a Python program that prompts the user to input an integer and raises a ValueErrorException if the input is not a valid integer.
	10.4	Write a Python program that executes division and handles an ArithmeticError exception if there is an arithmetic error.
11. File Handling	11.1	Write a python code to perform following operations with a text file: i) create ii) open iii) read iv) write v) append vi) close vii) delete

	11.2	Write python program to find the most frequent words in a text read from a file.
	11.3	Write a python program to read first n lines of a file.
	11.4	Write a python program to count the number of lines in a text file.
	11.5	Write a python program to count the frequency of words in a file.
	11.6	Write a python program to copy the contents of a file to another file.
	11.7	Write a python program to search for a string in text file
12. Handling csv files	12.1	Write a python code to read a csv file using pandas' module and print the first and last five lines of a file.
	12.2	Write a python program to create, write and read CSV files Into a Dictionary.
13. MySQL	13.1	Write a python program to program to connect with MySQL database.
	13.2	Write a python program to perform following database operations: i) create ii) alter iii) insert iv) update v) drop vi) delete

EXPERIMENT 1.1

AIM: Write a python program to print first n prime numbers.

Code:

```
def is_prime(num):
    if num <= 1:
        return False
    elif num == 2:
        return True
    elif num % 2 == 0:
        return False
    else:
        # Check divisibility by odd numbers up to the square root of num
        for i in range(3, int(num**0.5) + 1, 2):
            if num % i == 0:
                return False
        return True

def print_first_n_primes(n):
    count = 0
    num = 2
    while count < n:
        if is_prime(num):
            print(num, end=" ")
            count += 1
        num += 1

# Test the function
n = int(input("Enter the value of n: "))
print("First", n, "prime numbers are:")
print_first_n_primes(n)
```

OUTPUT :

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS\Desktop\Python Practicals> & C:/Users/
.py"
Enter the value of n: 15
First 15 prime numbers are:
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47
PS C:\Users\ASUS\Desktop\Python Practicals>
```

EXPERIMENT 1.2

Aim : Write a python program to multiply matrices.

Code :

```python
def matrix_multiply(matrix1, matrix2):
    # Check if the matrices can be multiplied
    if len(matrix1[0]) != len(matrix2):
        print("Matrices cannot be multiplied. Number of columns in the
first matrix must equal the number of rows in the second matrix.")
        return None

    # Initialize result matrix with zeros
    result = [[0 for _ in range(len(matrix2[0]))] for _ in
range(len(matrix1))]

    # Perform matrix multiplication
    for i in range(len(matrix1)):
        for j in range(len(matrix2[0])):
            for k in range(len(matrix2)):
                result[i][j] += matrix1[i][k] * matrix2[k][j]

    return result
```

```python
# Test the function
matrix1 = [[1, 2, 3],
           [4, 5, 6]]
matrix2 = [[7, 8],
           [9, 10],
           [11, 12]]

result = matrix_multiply(matrix1, matrix2)

if result:
    print("Result of matrix multiplication:")
    for row in result:
        print(row)
```

OUTPUT :

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS\Desktop\Python Practicals> & C:/Users/ASUS/AppD
ication.py"
Result of matrix multiplication:
[58, 64]
[139, 154]
PS C:\Users\ASUS\Desktop\Python Practicals>
```

EXPERIMENT 1.3

Aim : Write a python program which accepts the radius of a circle from user and computes the area (use math module).

Code :

```python
import math

def calculate_circle_area(radius):
    area = math.pi * radius**2
    return area
```

```python
# Accept radius from the user
radius = float(input("Enter the radius of the circle: "))

# Calculate area
area = calculate_circle_area(radius)

# Print the area
print("The area of the circle with radius", radius, "is:", area)
```

OUTPUT :

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS\Desktop\Python Practicals> & C:/Users/ASUS/AppData/L
py"
Enter the radius of the circle: 4
The area of the circle with radius 4.0 is: 50.26548245743669
PS C:\Users\ASUS\Desktop\Python Practicals>
```

EXPERIMENT 2.1

Aim: Write python program to let user enter some data in string and then verify data and print welcome to user.

Code :

```
def verify_data(data):
    # Check if the data meets certain criteria (for example, it should
not be empty)
    if data.strip():  # Check if the data contains non-space characters
        return True
    else:
        return False

def welcome_user(name):
    print("Welcome,", name, "!")

# Accept data from the user
data = input("Enter your name: ")

# Verify the data
if verify_data(data):
    # If the data is valid, print a welcome message
    welcome_user(data)
else:
    # If the data is invalid, print an error message
    print("Invalid input. Please enter your name.")
```

OUTPUT :

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS\Desktop\Python Practicals> & C:/Users
y"
Enter your name: Shreyansh
Welcome, Shreyansh !
PS C:\Users\ASUS\Desktop\Python Practicals>
```

EXPERIMENT 2.2

Aim : Write a Python program that checks whether a passed string is a palindrome or not.

Code :

```
def is_palindrome(s):
    # Convert the string to lowercase and remove non-alphanumeric
characters
    s = ''.join(char.lower() for char in s if char.isalnum())
    # Check if the string is equal to its reverse
    return s == s[::-1]

# Test the function
string = input("Enter a string: ")
if is_palindrome(string):
    print("The string is a palindrome.")
else:
    print("The string is not a palindrome.")
```

OUTPUT :

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS\Desktop\Python Practicals> & C:/Users/AS
"
Enter a string: Shreyansh
The string is not a palindrome.
PS C:\Users\ASUS\Desktop\Python Practicals>
```

EXPERIMENT 2.3

AIM : Write a Python program to count the occurrences of each word for given string.

Code :

```python
def count_word_occurrences(string):
    # Split the string into words
    words = string.split()

    # Create a dictionary to store word counts
    word_counts = {}

    # Count occurrences of each word
    for word in words:
        # Remove punctuation from the word
        word = word.strip(",.!?;:'\"").lower()
        # Update word count in the dictionary
        word_counts[word] = word_counts.get(word, 0) + 1

    return word_counts

# Test the function
string = input("Enter a string: ")
word_counts = count_word_occurrences(string)
print("Occurrences of each word:")
for word, count in word_counts.items():
    print(f"'{word}': {count}")
```

OUTPUT :

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS\Desktop\Python Practicals> & C:/Users/ASUS/Ap
rrences.py"
Enter a string: Shreyansh
Occurrences of each word:
'shreyansh': 1
PS C:\Users\ASUS\Desktop\Python Practicals>
```

EXPERIMENT 2.4

Aim : Write a Python function to reverse a string. Sample String: "1234abcd" Expected Output: "dcba4321"

Code :

```python
def reverse_string(string):
    # Use string slicing to reverse the string
    reversed_string = string[::-1]
    return reversed_string

# Test the function
sample_string = "1234abcd"
reversed_output = reverse_string(sample_string)
print("Original string:", sample_string)
print("Reversed string:", reversed_output)
```

OUTPUT :

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS\Desktop\Python Practicals> & C:/Users/ASUS/App
.py"
Original string: 1234abcd
Reversed string: dcba4321
PS C:\Users\ASUS\Desktop\Python Practicals>
```

EXPERIMENT 2.5

Aim: Write a Python program to remove the characters which have odd index values of a given string.

Code :

```python
def remove_odd_index_characters(string):
    # Initialize an empty string to store characters with even indices
    result = ""

    # Iterate through the characters of the string
    for index, char in enumerate(string):
        # Check if the index is even
        if index % 2 == 0:
            # Append the character to the result string
            result += char

    return result

# Test the function
given_string = input("Enter a string: ")
result_string = remove_odd_index_characters(given_string)
print("Result string:", result_string)
```

OUTPUT:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS\Desktop\Python Practicals> & C:/Users/ASUS/Ap
dex_characters.py"
Enter a string: Shreyansh
Result string: Srynh
PS C:\Users\ASUS\Desktop\Python Practicals>
```

EXPERIMENT 2.6

Aim: Write python program to take command line arguments (word count).

Code :

```python
import sys

def word_count(sentence):
    # Split the sentence into words using whitespace as the delimiter
    words = sentence.split()
    # Count the number of words
    return len(words)

# Check if the program is run with the correct number of arguments
if len(sys.argv) != 2:
    print("Usage: python program_name.py 'sentence'")
    sys.exit(1)

# Get the sentence from the command-line argument
sentence = sys.argv[1]

# Calculate and print the word count
count = word_count(sentence)
print("Word count:", count)
```

OUTPUT :

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS\Desktop\Python Practicals> python WordCount.py 'My name is Shreyansh'
>>
Word count: 4
PS C:\Users\ASUS\Desktop\Python Practicals>
```

EXPERIMENT 3.1

Aim: Write the output of the following Python program: a = 10 b = 4 print (a & b) print (a | b) print (~a) print (a ^ b) print (a >> 2) print (a << 2).

Code :

```
a = 10
b = 4
print(a & b)     # Bitwise AND
print(a | b)     # Bitwise OR
print(~a)        # Bitwise NOT
print(a ^ b)     # Bitwise XOR
print(a >> 2)    # Right shift
print(a << 2)    # Left shift
```

OUTPUT :

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS\Desktop\Python Practicals> & C:/Users/ASUS
0
14
-11
14
2
40
PS C:\Users\ASUS\Desktop\Python Practicals>
```

EXPERIMENT 3.2

Aim: Write a program to create a menu with the following options and accepts users input and perform the operation accordingly:
i) Addition ii) Subtraction iii) Multiplication iv) Division

Code :

```
def addition(num1, num2):
    return num1 + num2

def subtraction(num1, num2):
    return num1 - num2

def multiplication(num1, num2):
    return num1 * num2

def division(num1, num2):
    if num2 != 0:
        return num1 / num2
    else:
        return "Error: Division by zero"

def menu():
    print("Menu:")
    print("1. Addition")
    print("2. Subtraction")
    print("3. Multiplication")
    print("4. Division")
```

```python
# Get user input for operation choice
choice = input("Enter your choice (1/2/3/4): ")

# Perform operation based on user's choice
if choice in ['1', '2', '3', '4']:
    num1 = float(input("Enter the first number: "))
    num2 = float(input("Enter the second number: "))

    if choice == '1':
        print("Result:", addition(num1, num2))
    elif choice == '2':
        print("Result:", subtraction(num1, num2))
    elif choice == '3':
        print("Result:", multiplication(num1, num2))
    elif choice == '4':
        print("Result:", division(num1, num2))
else:
    print("Invalid choice. Please enter a valid option (1/2/3/4).")
```

OUTPUT :

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS\Desktop\Python Practicals> & C:/Users/ASUS/
"
Enter your choice (1/2/3/4): 2
Enter the first number: 15
Enter the second number: 7
Result: 8.0
PS C:\Users\ASUS\Desktop\Python Practicals>
```

EXPERIMENT 4.1

Aim: Write a python program to print a number is even/odd using if-else.

Code :

```python
def check_even_odd(number):
    if number % 2 == 0:
        print(number, "is even.")
    else:
        print(number, "is odd.")

# Test the function
number = int(input("Enter a number: "))
check_even_odd(number)
```

OUTPUT :

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS\Desktop\Python Practicals> & C:/Users/ASUS/App
Enter a number: 12
12 is even.
PS C:\Users\ASUS\Desktop\Python Practicals>
```

EXPERIMENT 4.2

Aim: Write a python program to find largest number among three numbers

Code :

```python
def find_largest_number(num1, num2, num3):
    if num1 >= num2 and num1 >= num3:
        return num1
    elif num2 >= num1 and num2 >= num3:
        return num2
    else:
        return num3

# Test the function
num1 = float(input("Enter the first number: "))
num2 = float(input("Enter the second number: "))
num3 = float(input("Enter the third number: "))

largest_number = find_largest_number(num1, num2, num3)
print("The largest number among", num1, ",", num2, ", and", num3, "is:",
largest_number)
```

OUTPUT :

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS\Desktop\Python Practicals> & C:/Users/ASUS/AppDat
.py"
Enter the first number: 15
Enter the second number: 60
Enter the third number: 89
The largest number among 15.0 , 60.0 , and 89.0 is: 89.0
PS C:\Users\ASUS\Desktop\Python Practicals>
```

EXPERIMENT 4.3

Aim: Write a python Program to read a number and display corresponding day using if_elif_else?

Code :

```
def display_day(day_number):
    if day_number == 1:
        print("Monday")
    elif day_number == 2:
        print("Tuesday")
    elif day_number == 3:
        print("Wednesday")
    elif day_number == 4:
        print("Thursday")
    elif day_number == 5:
        print("Friday")
    elif day_number == 6:
        print("Saturday")
    elif day_number == 7:
        print("Sunday")
    else:
        print("Invalid day number. Please enter a number between 1 and
7.")

# Test the function
day_number = int(input("Enter a number (1-7) to display the corresponding
day: "))
display_day(day_number)
```

OUTPUT:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS\Desktop\Python Practicals> & C:/Users/ASUS/AppData/Loc
Enter a number (1-7) to display the corresponding day: 6
Saturday
PS C:\Users\ASUS\Desktop\Python Practicals>
```

EXPERIMENT 4.4

Aim: Write a Python program to get the Fibonacci series between 0 to 50.

Code :

```
def fibonacci_series(n):
    fib_series = [0, 1]  # Initialize the Fibonacci series with first two
terms

    # Generate Fibonacci series up to the nth term
    while fib_series[-1] + fib_series[-2] <= n:
        fib_series.append(fib_series[-1] + fib_series[-2])

    return fib_series

# Test the function
fib_series = fibonacci_series(50)
print("Fibonacci series up to 50:", fib_series)
```

OUTPUT:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS\Desktop\Python Practicals> & C:/Users/ASUS/AppData/

Fibonacci series up to 50: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
PS C:\Users\ASUS\Desktop\Python Practicals>
```

EXPERIMENT 4.5

Aim: Write python program to print list of numbers using range and for loop.

Code :

```python
# Define the range of numbers (start, stop, step)
start = 1
stop = 11
step = 1

# Print the list of numbers using a for loop
print("List of numbers using range and for loop:")
for num in range(start, stop, step):
    print(num)
```

OUTPUT:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

r.py"
List of numbers using range and for loop:
1
2
3
4
5
6
7
8
9
10
PS C:\Users\ASUS\Desktop\Python Practicals>
```

EXPERIMENT 4.6

Aim: Write a Python program to construct the following pattern, using a nested for loop.

```
*
**
***
****
*****
****
***
**
*
```

Code:

```python
# Define the number of rows for the pattern
num_rows = 5

# Upper half of the pattern
for i in range(1, num_rows + 1):
    print('*' * i)

# Lower half of the pattern
for i in range(num_rows - 1, 0, -1):
    print('*' * i)
```

OUTPUT:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS\Desktop\Python Practicals> & C:/Users/ASUS/AppD
*
**
***
****
*****
****
***
**
*
PS C:\Users\ASUS\Desktop\Python Practicals>
```

EXPERIMENT 5.1

Aim: Create a list and perform the following methods 1) insert() 2) remove() 3) append() 4) len() 5) pop() 6)clear().

Code:

```
# Create an empty list
my_list = []

# 1) insert(): Insert an element at a specific index
my_list.insert(0, 'a')  # Insert 'a' at index 0
my_list.insert(1, 'b')  # Insert 'b' at index 1
print("After insertions:", my_list)

# 2) remove(): Remove the first occurrence of a value
my_list.remove('a')  # Remove the first occurrence of 'a'
print("After removing 'a':", my_list)

# 3) append(): Append an element to the end of the list
my_list.append('c')  # Append 'c' to the end of the list
print("After appending 'c':", my_list)

# 4) len(): Get the length of the list
print("Length of the list:", len(my_list))

# 5) pop(): Remove and return the last element of the list
last_element = my_list.pop()  # Remove and return 'c'
print("Popped element:", last_element)
print("List after pop:", my_list)

# 6) clear(): Remove all elements from the list
my_list.clear()  # Clear the list
print("List after clearing:", my_list)
```

OUTPUT:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS\Desktop\Python Practicals> & C:/Users/ASUS/AppDa
After insertions: ['a', 'b']
After removing 'a': ['b']
After appending 'c': ['b', 'c']
Length of the list: 2
Popped element: c
List after pop: ['b']
List after clearing: []
PS C:\Users\ASUS\Desktop\Python Practicals>
```

EXPERIMENT 5.2

Aim: Create a tuple and perform the following methods 1) Add items 2) len() 3) check for item in tuple4)Access items.

Code:

```python
# Create a tuple
my_tuple = (1, 2, 3, 4, 5)

# 1) Add items (Creating a new tuple with added items)
new_tuple = my_tuple + (6, 7)
print("New tuple with added items:", new_tuple)

# 2) len(): Get the length of the tuple
print("Length of the tuple:", len(my_tuple))

# 3) Check for item in tuple
item = 3
if item in my_tuple:
    print(f"{item} is present in the tuple.")
else:
    print(f"{item} is not present in the tuple.")

# 4) Access items
print("First item:", my_tuple[0])
print("Last item:", my_tuple[-1]
```

OUTPUT:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS\Desktop\Python Practicals> & C:/Users/ASUS/AppData/
New tuple with added items: (1, 2, 3, 4, 5, 6, 7)
Length of the tuple: 5
3 is present in the tuple.
First item: 1
Last item: 5
PS C:\Users\ASUS\Desktop\Python Practicals>
```

EXPERIMENT 5.3

Aim: Write a Python program to sum all the items in a list.

Code:

```python
# Define a list of numbers
my_list = [1, 2, 3, 4, 5]

# Sum all the items in the list
total = sum(my_list)

# Print the sum
print("Sum of all the items in the list:", total)
```

OUTPUT:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS\Desktop\Python Practicals> & C:/Users/ASUS/App

Sum of all the items in the list: 15
PS C:\Users\ASUS\Desktop\Python Practicals>
```

EXPERIMENT 5.4

Aim: Write a Python function that takes two lists and returns true if they are equal otherwise false.

Code:

```python
def are_lists_equal(list1, list2):
    # Check if the lengths of the lists are equal
    if len(list1) != len(list2):
        return False

    # Compare elements of the lists
    for i in range(len(list1)):
        if list1[i] != list2[i]:
            return False

    # If all elements are equal, return True
    return True

# Test the function
list1 = [1, 2, 3, 4, 5]
list2 = [1, 2, 3, 4, 5]
print("Lists:")
print(list1)
```

```
print(list2)
print("\nAre the lists equal?", are_lists_equal(list1, list2))
```

OUTPUT :

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS\Desktop\Python Practicals> & C:/Users/ASUS/AppData/Loc

Lists:
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5]

Are the lists equal? True
PS C:\Users\ASUS\Desktop\Python Practicals>
```

EXPERIMENT 5.5

Aim: Write python program to store strings in list and then print them

Code:

```
# Create a list to store strings
string_list = []

# Add strings to the list
string_list.append("Hello")
string_list.append("World")
string_list.append("Python")
string_list.append("Programming")

# Print the strings in the list
print("Strings in the list:")
for string in string_list:
    print(string)
```

OUTPUT:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS\Desktop\Python Practicals> & C:/Users/ASUS/Ap

Strings in the list:
Hello
World
Python
Programming
PS C:\Users\ASUS\Desktop\Python Practicals>
```

EXPERIMENT 6.1

Aim: Create a dictionary and apply the following methods 1) Print the dictionary items 2) access items 3) useget() 4)change values 5) use len().

Code:

```
# Create a dictionary
my_dict = {'name': 'John', 'age': 30, 'city': 'New York'}

# 1) Print the dictionary items
print("Dictionary items:")
for key, value in my_dict.items():
    print(key, ":", value)

# 2) Access items
print("\nAccess items:")
print("Name:", my_dict['name'])
print("Age:", my_dict['age'])
print("City:", my_dict['city'])

# 3) Use get()
print("\nUse get():")
print("Name:", my_dict.get('name'))
print("Country:", my_dict.get('country', 'Not found'))

# 4) Change values
print("\nChange values:")
my_dict['age'] = 35
print("Updated age:", my_dict['age'])

# 5) Use len()
print("\nLength of the dictionary:", len(my_dict))
```

OUTPUT:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS\Desktop\Python Practicals> & C:/Users/ASUS/
"
Dictionary items:
name : John
age : 30
city : New York

Access items:
Name: John
Age: 30
City: New York

Use get():
Name: John
Country: Not found

Change values:
Updated age: 35

Length of the dictionary: 3
PS C:\Users\ASUS\Desktop\Python Practicals>
```

EXPERIMENT 6.2

Aim: Write a Python script to check if a given key already exists in a dictionary.

Code:

```python
def key_exists(dictionary, key):
    # Check if the key exists using the 'in' operator
    if key in dictionary:
```

```
        return True, dictionary[key]  # Return True and corresponding
value
    else:
        return False, None  # Return False and None for the value

# Test the function
my_dict = {'name': 'John', 'age': 30, 'city': 'New York'}

key_to_check = input("Enter the key to check: ")

exists, value = key_exists(my_dict, key_to_check)

if exists:
    print(f"The key '{key_to_check}' exists in the dictionary with value
'{value}'.")
else:
    print(f"The key '{key_to_check}' does not exist in the dictionary.")
```

OUTPUT :

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS\Desktop\Python Practicals> & C:/Users/ASUS/AppData/Loc

Enter the key to check: age
The key 'age' exists in the dictionary with value '30'.
PS C:\Users\ASUS\Desktop\Python Practicals>
```

EXPERIMENT 6.3

Aim: Write a Python script to sort (ascending and descending) a dictionary by value.

Code:

```python
# Define a dictionary
my_dict = {'apple': 30, 'banana': 20, 'orange': 25, 'mango': 15}

# Sort the dictionary by values in ascending order
sorted_dict_asc = dict(sorted(my_dict.items(), key=lambda item: item[1]))

# Sort the dictionary by values in descending order
sorted_dict_desc = dict(sorted(my_dict.items(), key=lambda item: item[1],
reverse=True))

# Print the sorted dictionaries
print("Dictionary sorted by values in ascending order:")
for key, value in sorted_dict_asc.items():
    print(key, ":", value)

print("\nDictionary sorted by values in descending order:")
for key, value in sorted_dict_desc.items():
    print(key, ":", value)
```

OUTPUT :

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS\Desktop\Python Practicals> & C:/Users/ASUS/AppData/Lo
"
Dictionary sorted by values in ascending order:
mango : 15
banana : 20
orange : 25
apple : 30

Dictionary sorted by values in descending order:
apple : 30
orange : 25
banana : 20
mango : 15
PS C:\Users\ASUS\Desktop\Python Practicals>
```

EXPERIMENT 7.1

Aim: Write python program in which a function is defined and calling that function prints ‘Hello World’.

Code :

```python
def print_hello_world():
    print("Hello World")

# Calling the function
print_hello_world()
```

OUTPUT:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS\Desktop\Python Practicals> & C:/Users/
Hello World
PS C:\Users\ASUS\Desktop\Python Practicals>
```

EXPERIMENT 7.2

Aim: Write python program in which a function (with single string parameter) is defined and calling that function prints the string parameters given to function.

Code:

```
def print_string(parameter):
    print(parameter)

# Calling the function with a string parameter
print_string("Hello, I'm a string parameter.")
```

OUTPUT :

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS\Desktop\Python Practicals> & C:/Users/ASUS/A
ter.py"
Hello, I'm a string parameter.
PS C:\Users\ASUS\Desktop\Python Practicals>
```

EXPERIMENT 7.3

Aim: Write a python program to find factorial of a given number using functions.

Code:

```
def factorial(num):
    if num == 0 or num == 1:
        return 1
    else:
        return num * factorial(num - 1)

# Get input from the user
number = int(input("Enter a number to find its factorial: "))

# Call the function to find factorial
```

```
result = factorial(number)

# Print the result
print("Factorial of", number, "is:", result)
```

OUTPUT :

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS\Desktop\Python Practicals> & C:/Users/ASUS/Ap

Enter a number to find its factorial: 5
Factorial of 5 is: 120
PS C:\Users\ASUS\Desktop\Python Practicals>
```

EXPERIMENT 7.4

Aim: Write a Python function to calculate the factorial of a number (a non-negative integer). The function accepts the number as an argument.

Code:

```
def factorial(num):
    # Check if the number is non-negative
    if num < 0:
        return "Factorial is not defined for negative numbers."
    # Calculate factorial
    result = 1
    for i in range(1, num + 1):
        result *= i
    return result

# Test the function
number = int(input("Enter a non-negative integer to calculate its
factorial: "))
```

```
print("Factorial of", number, "is:", factorial(number))
```

OUTPUT :

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS\Desktop\Python Practicals> & C:/Users/ASUS/AppData/
rial.py"
Enter a non-negative integer to calculate its factorial: 5
Factorial of 5 is: 120
PS C:\Users\ASUS\Desktop\Python Practicals>
```

EXPERIMENT 8.1

Aim:

Write a Python program to check the validity of passwords input by users using regular expression. Validations are:
At least 1 letter between [a-z] and 1 letter between [A-Z]
At least 1 number between [0-9]
At least 1 character from [$#@]
Minimum length 6 characters
Maximum length 16 characters

Code:

```python
import re

def validate_password(password):
    # Define the regular expressions for each criteria
    regex_lowercase = re.compile(r'[a-z]')
    regex_uppercase = re.compile(r'[A-Z]')
    regex_digit = re.compile(r'[0-9]')
    regex_special = re.compile(r'[$#@]')

    # Check the length of the password
    if len(password) < 6 or len(password) > 16:
        return False, "Password length should be between 6 and 16
characters."

    # Check if the password meets all criteria using regular expressions
    if (not regex_lowercase.search(password) or
        not regex_uppercase.search(password) or
        not regex_digit.search(password) or
        not regex_special.search(password)):
        return False, ("Password must contain at least one lowercase
letter, one uppercase letter, "
                       "one digit, and one special character ($#@).")

    # Password meets all criteria
    return True, "Password is valid."

# Test the function
password = input("Enter a password to validate: ")
is_valid, message = validate_password(password)
print(message)
```

OUTPUT:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS                                    Pytho

PS C:\Users\ASUS\Desktop\Python Practicals> & C:/Users/ASUS/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/ASUS/Des
Enter a password to validate: Shreyansh
Password must contain at least one lowercase letter, one uppercase letter, one digit, and one special character ($#@).
PS C:\Users\ASUS\Desktop\Python Practicals> & C:/Users/ASUS/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/ASUS/Des
Enter a password to validate: Shreyansh#7
Password is valid.
PS C:\Users\ASUS\Desktop\Python Practicals>
```

EXPERIMENT 9.1

Aim: Write python program in which a class is defined, then create object of that class and call simple 'print function' defined in class.

Code :

```python
class MyClass:
    def print_message(self):
        print("Hello from MyClass!")

# Create an object of the class
obj = MyClass()

# Call the print function defined in the class
obj.print_message()
```

OUTPUT:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS\Desktop\Python Practicals> & C:/Users/ASUS/
Hello from MyClass!
PS C:\Users\ASUS\Desktop\Python Practicals>
```

EXPERIMENT 9.2

Aim: Write a python Program to call data member and function using classes and objects

Code:

```python
class MyClass:
    # Data member
    my_variable = "Hello, I'm a data member."

    # Function
    def my_function(self):
        return "Hello, I'm a function."

# Create an object of the class
obj = MyClass()

# Access the data member using the object
print("Data member:", obj.my_variable)

# Call the function using the object
print("Function:", obj.my_function())
```

OUTPUT :

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS\Desktop\Python Practicals> & C:/Users/ASUS/AppData/L
Data member: Hello, I'm a data member.
Function: Hello, I'm a function.
PS C:\Users\ASUS\Desktop\Python Practicals>
```

EXPERIMENT 9.3

Aim: Write a Python class Employee with attributes like emp_name, emp_id, emp_salary, emp_department and methods like calculate_salary, and print_details. Use 'calculate_salary' method takes two arguments: salary and hours_worked. If the number of hours worked is more than 50, the method computes overtime and adds it to the salary. Overtime is calculated as following formula:

overtime = hours_worked – 50
overtime amount = (overtime * (salary / 50))
Use 'print_details' method to print the details of employee. Consider the sample data:
"Adams " "E7876" 50000 "Accounting"
"Jones" "E7499" 45000 "Research"
"Martin" "E7900" 50000 "Sales"
"Smith" "E7698" 55000 "Operations"

Code :

```
class Employee:
    def __init__(self, emp_name, emp_id, emp_salary, emp_department):
        self.emp_name = emp_name
        self.emp_id = emp_id
        self.emp_salary = emp_salary
        self.emp_department = emp_department

    def calculate_salary(self, salary, hours_worked):
        if hours_worked > 50:
            overtime = hours_worked - 50
            overtime_amount = overtime * (salary / 50)
            total_salary = salary + overtime_amount
            return total_salary
        else:
            return salary

    def print_details(self):
        print("Employee Name:", self.emp_name)
        print("Employee ID:", self.emp_id)
        print("Employee Salary:", self.emp_salary)
        print("Employee Department:", self.emp_department)


# Sample data
employee1 = Employee("Adams", "E7876", 50000, "Accounting")
employee2 = Employee("Jones", "E7499", 45000, "Research")
employee3 = Employee("Martin", "E7900", 50000, "Sales")
employee4 = Employee("Smith", "E7698", 55000, "Operations")

# Calculate and print details
employees = [employee1, employee2, employee3, employee4]
for employee in employees:
    print("\n")
    print("Employee Details:")
    employee.print_details()
```

```
    print("Total Salary:", employee.calculate_salary(employee.emp_salary,
55))
```

OUTPUT:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS\Desktop\Python Practicals> & C:/Users/ASUS/AppD
ls.py"


Employee Details:
Employee Name: Adams
Employee ID: E7876
Employee Salary: 50000
Employee Department: Accounting
Total Salary: 55000.0


Employee Details:
Employee Name: Jones
Employee ID: E7499
Employee Salary: 45000
Employee Department: Research
Total Salary: 49500.0


Employee Details:
Employee Name: Martin
Employee ID: E7900
Employee Salary: 50000
Employee Department: Sales
Total Salary: 55000.0
```

EXPERIMENT 9.4

Aim: Write a python program to demonstrate access specifiers.

Code:

```
class MyClass:
    def __init__(self):
        # Public attribute
        self.public_attr = "I am a public attribute"
        # Protected attribute
        self._protected_attr = "I am a protected attribute"
        # Private attribute
        self.__private_attr = "I am a private attribute"

    def public_method(self):
        print("Public method called")
        # Accessing all attributes inside the class
        print("Accessing public attribute:", self.public_attr)
        print("Accessing protected attribute:", self._protected_attr)
        print("Accessing private attribute:", self.__private_attr)

    def _protected_method(self):
        print("Protected method called")

    def __private_method(self):
        print("Private method called")


# Create an object of the class
obj = MyClass()

# Accessing public attributes and methods
print("Accessing public attribute outside the class:", obj.public_attr)
obj.public_method()

# Accessing protected attributes and methods (conventionally)
print("Accessing protected attribute outside the class:",
obj._protected_attr)
obj._protected_method()

# Accessing private attributes and methods (conventionally)
# Note: Accessing private attributes and methods directly outside the
class may raise AttributeError
# print("Accessing private attribute outside the class:",
obj.__private_attr)  # Raises AttributeError
# obj.__private_method()  # Raises AttributeError

# Accessing private attributes and methods using name mangling
```

```
print("Accessing private attribute outside the class:",
obj._MyClass__private_attr)
obj._MyClass__private_method()
```

OUTPUT:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS\Desktop\Python Practicals> & C:/Users/ASUS/AppData/Local/Programs/Python
er.py"
Accessing public attribute outside the class: I am a public attribute
Public method called
Accessing public attribute: I am a public attribute
Accessing protected attribute: I am a protected attribute
Accessing private attribute: I am a private attribute
Accessing protected attribute outside the class: I am a protected attribute
Protected method called
Accessing private attribute outside the class: I am a private attribute
Private method called
PS C:\Users\ASUS\Desktop\Python Practicals>
```

EXPERIMENT 9.5 – 9.9

Aim:

9.5 Write a python program to apply polymorphism.
9.6 Write a python program to demonstrate inheritance.
9.7 Write a python program for method overriding.
9.8 Write a python program to define abstraction.
9.9 Write a python program to demonstrate Interface.

9.5 Code:

```
class Animal:
    def speak(self):
        pass

class Dog(Animal):
    def speak(self):
```

```python
        return "Woof!"

class Cat(Animal):
    def speak(self):
        return "Meow!"

# Polymorphic function
def make_sound(animal):
    return animal.speak()

# Test the polymorphic function
dog = Dog()
cat = Cat()
print(make_sound(dog))  # Output: Woof!
print(make_sound(cat))  # Output: Meow!
```

9.6 Code:

```python
class Animal:
    def __init__(self, species):
        self.species = species

    def sound(self):
        return "Some generic sound"

class Dog(Animal):
    def __init__(self, name):
        super().__init__("Dog")
        self.name = name

    def sound(self):
        return "Woof!"

# Create an object of the subclass
dog = Dog("Buddy")
print(dog.species)  # Output: Dog
print(dog.sound())  # Output: Woof!
```

9.7 Code:

```
class Animal:
    def sound(self):
        return "Some generic sound"

class Dog(Animal):
    def sound(self):
        return "Woof!"

# Create an object of the subclass
dog = Dog()
print(dog.sound())  # Output: Woof!
```

9.8 Code:

```
from abc import ABC, abstractmethod

class Animal(ABC):
    @abstractmethod
    def sound(self):
        pass

class Dog(Animal):
    def sound(self):
        return "Woof!"

# Create an object of the subclass
dog = Dog()
print(dog.sound())  # Output: Woof!
```

9.9 Code:

```
from abc import ABC, abstractmethod

class Interface(ABC):
    @abstractmethod
    def method1(self):
        pass
```

```
    @abstractmethod
    def method2(self):
        pass

class MyClass(Interface):
    def method1(self):
        return "Implementation of method1"

    def method2(self):
        return "Implementation of method2"

# Create an object of the subclass
obj = MyClass()
print(obj.method1())  # Output: Implementation of method1
print(obj.method2())  # Output: Implementation of method2
```

EXPERIMENT 10.1

Aim: Demonstrate a python code to print try, except and finally block statements.

Code:

```
def divide():
    try:
        x = float(input("Enter the numerator: "))
        y = float(input("Enter the denominator: "))
        result = x / y
    except ZeroDivisionError:
        print("Error: Division by zero!")
    else:
        print("Result of division:", result)
    finally:
        print("Executing finally block")
```

```
# Call the function
divide()
```

OUTPUT :

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS\Desktop\Python Practicals> & C:/Users/ASUS/
elling.py"
Enter the numerator: 50
Enter the denominator: 25
Result of division: 2.0
Executing finally block
PS C:\Users\ASUS\Desktop\Python Practicals>
```

EXPERIMENT 10.2

Aim: Demonstrate a python code to implement abnormal termination.

Code:

```
import sys

def divide(x, y):
    try:
        result = x / y
    except ZeroDivisionError:
        print("Error: Division by zero!")
        sys.exit(1)  # Abnormal termination with exit code 1
    else:
        print("Result of division:", result)

# Test cases
divide(10, 2)  # Output: Result of division: 5.0
divide(10, 0)  # Output: Error: Division by zero!
```

OUTPUT:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS\Desktop\Python Practicals> & C:/Users/ASUS/AppData/Local/Programs/
nation.py"
Result of division: 5.0
Error: Division by zero!
PS C:\Users\ASUS\Desktop\Python Practicals>
```

EXPERIMENT 10.3

Aim: Write a Python program that prompts the user to input an integer and raises a ValueErrorException if the input is not a valid integer.

Code :

```python
try:
    # Prompt the user to input an integer
    num = int(input("Please enter an integer: "))
    print("You entered:", num)
except ValueError:
    # Handle the case where the input is not a valid integer
    print("Error: Please enter a valid integer.")
```

OUTPUT :

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS\Desktop\Python Practicals> & C:/Users/ASUS/AppData/
eption.py"
Please enter an integer: 2.3
Error: Please enter a valid integer.
PS C:\Users\ASUS\Desktop\Python Practicals>
```

EXPERIMENT 10.4

Aim: Write a Python program that executes division and handles an ArithmeticError exception if there is an arithmetic error.

Code:

```
try:
    # Perform division
    result = 10 / 0  # This will raise an ArithmeticError
except ArithmeticError:
    # Handle the arithmetic error
    print("Error: ArithmeticError occurred!")
```

OUTPUT :

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS\Desktop\Python Practicals> & C:/Users/ASUS/Ap
ror.py"
Error: ArithmeticError occurred!
PS C:\Users\ASUS\Desktop\Python Practicals>
```

EXPERIMENT 11.1

Aim: Write a python code to perform following operations with a text file:

i) create ii) open iii) read iv) write v) append vi) close vii) delete.

Code :

```python
import os

# Specify the full path to the file in the desired directory
directory = "C:\\Users\\ASUS\\Desktop\\Python Practicals"
filename = os.path.join(directory, "sample.txt")

# i) Create a text file
def create_file(filename):
    try:
        with open(filename, 'x'):
            print(f"File '{filename}' created successfully.")
    except FileExistsError:
        print(f"Error: File '{filename}' already exists.")

# ii) Open a text file
def open_file(filename, mode='r'):
    try:
        file = open(filename, mode)
        print(f"File '{filename}' opened successfully.")
        return file
    except FileNotFoundError:
        print(f"Error: File '{filename}' not found.")
        return None

# iii) Read from a text file
def read_file(file):
    if file:
        content = file.read()
        print("File content:")
        print(content)
        file.close()  # Close the file after reading its content

# iv) Write to a text file
def write_to_file(filename, content):
    with open(filename, 'w') as file:
        file.write(content)
        print(f"Content written to file '{filename}' successfully.")

# v) Append to a text file
def append_to_file(filename, content):
    with open(filename, 'a') as file:
        file.write(content)
        print(f"Content appended to file '{filename}' successfully.")
```

```python
# vi) Close a text file
# Since files are opened using 'with' statement, they are automatically
closed

# vii) Delete a text file
def delete_file(filename):
    try:
        os.remove(filename)
        print(f"File '{filename}' deleted successfully.")
    except FileNotFoundError:
        print(f"Error: File '{filename}' not found.")

# Test the functions
create_file(filename)
file = open_file(filename)
read_file(file)
write_to_file(filename, "Hello, world!\n")
file = open_file(filename)
read_file(file)
append_to_file(filename, "This is a new line.\n")
file = open_file(filename)
read_file(file)
delete_file(filename)
```

OUTPUT :

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS\Desktop\Python Practicals> & C:/Users/ASUS/AppData/Local/Programs/Python/Python312/python.e
.py"
File 'C:\Users\ASUS\Desktop\Python Practicals\sample.txt' created successfully.
File 'C:\Users\ASUS\Desktop\Python Practicals\sample.txt' opened successfully.
File content:

Content written to file 'C:\Users\ASUS\Desktop\Python Practicals\sample.txt' successfully.
File 'C:\Users\ASUS\Desktop\Python Practicals\sample.txt' opened successfully.
File content:
Hello, world!

Content appended to file 'C:\Users\ASUS\Desktop\Python Practicals\sample.txt' successfully.
File 'C:\Users\ASUS\Desktop\Python Practicals\sample.txt' opened successfully.
File content:
Hello, world!
This is a new line.

File 'C:\Users\ASUS\Desktop\Python Practicals\sample.txt' deleted successfully.
PS C:\Users\ASUS\Desktop\Python Practicals>
```

EXPERIMENT 11.2

Aim: Write python program to find the most frequent words in a text read from a file.

Code:

```
from collections import Counter

def create_file(filename, content):
    with open(filename, 'w') as file:
        file.write(content)
    print(f"File '{filename}' created successfully.")

def count_most_frequent_words(filename, num_words=5):
    # Read the text from the file
    with open(filename, 'r') as file:
        text = file.read()

    # Tokenize the text into words
    words = text.split()

    # Count the frequency of each word
    word_freq = Counter(words)

    # Find the most frequent words
    most_common_words = word_freq.most_common(num_words)

    return most_common_words

# Create a text file and add some content to it
filename = "sample.txt"
content = "apple banana banana cherry cherry cherry cherry cherry"
create_file(filename, content)

# Count the most frequent words from the file
num_words = 3  # Number of most frequent words to find
most_frequent_words = count_most_frequent_words(filename, num_words)

print("Most frequent words:")
for word, frequency in most_frequent_words:
    print(f"{word}: {frequency}")
```

OUTPUT :

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS\Desktop\Python Practicals> & C:/Users/ASUS/AppData/Lo
File 'sample.txt' created successfully.
Most frequent words:
cherry: 5
banana: 2
apple: 1
PS C:\Users\ASUS\Desktop\Python Practicals>
```

EXPERIMENT 11.3

Aim: Write a python program to read first n lines of a file.

Code :

```python
def read_first_n_lines(filename, n):
    lines = []
    # Open the file
    with open(filename, 'r') as file:
        # Read the first n lines
        for _ in range(n):
            line = file.readline()
            if not line:  # If end of file is reached
                break
            lines.append(line.strip())  # Remove newline character and
add to list
    return lines

# Test the function
filename = "sample.txt"  # Replace with the path to your text file
n = 3  # Number of lines to read
first_n_lines = read_first_n_lines(filename, n)

print(f"First {n} lines of '{filename}':")
for line in first_n_lines:
    print(line)
```

OUTPUT :

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS\Desktop\Python Practicals> & C:/Users/ASUS/AppData/L
.py"
First 3 lines of 'sample.txt':
apple banana banana cherry cherry cherry cherry cherry
PS C:\Users\ASUS\Desktop\Python Practicals>
```

EXPERIMENT 11.4 – 11.5

Aim:

11.4 Write a python program to count the number of lines in a text file.

11.5 Write a python program to count the frequency of words in a file.

Code :

```python
from collections import Counter

def count_lines(filename):
    # Open the file and read lines
    with open(filename, 'r') as file:
        lines = file.readlines()
    # Count the number of lines
    num_lines = len(lines)
    return num_lines

def count_word_frequency(filename):
    # Open the file and read content
    with open(filename, 'r') as file:
        content = file.read()
    # Tokenize the content into words
    words = content.split()
```

```python
    # Count the frequency of each word
    word_freq = Counter(words)
    return word_freq

# Test the functions
filename = "sample.txt"  # Replace with the path to your text file
num_lines = count_lines(filename)
print(f"Number of lines in '{filename}': {num_lines}")

word_freq = count_word_frequency(filename)
print("Word frequencies:")
for word, frequency in word_freq.items():
    print(f"{word}: {frequency}")
```

OUTPUT :

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS\Desktop\Python Practicals> & C:/Users/ASUS/AppData/Local
.py"
Number of lines in 'sample.txt': 1
Word frequencies:
apple: 1
banana: 2
cherry: 5
PS C:\Users\ASUS\Desktop\Python Practicals>
```

EXPERIMENT 11.6

Aim: Write a python program to copy the contents of a file to another file.

Code:

```python
import os

def copy_file(source_file, destination_file):
    # Open the source file for reading
    with open(source_file, 'r') as source:
```

```
        # Read the content of the source file
        content = source.read()
    # Open the destination file for writing
    with open(destination_file, 'w') as destination:
        # Write the content to the destination file
        destination.write(content)
    print(f"Contents of '{source_file}' copied to '{destination_file}'
successfully.")

# Source file
source_file = "sample.txt"  # Replace with the path to your source file

# Destination file in the specified directory
destination_file = "destination.txt"  # Replace with the name of your
destination file
destination_path = os.path.join(destination_file)

# Copy the contents of the source file to the destination file
copy_file(source_file, destination_path)
```

OUTPUT :

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS\Desktop\Python Practicals> & C:/Users/ASUS/AppData/Local/Pr
Contents of 'sample.txt' copied to 'destination.txt' successfully.
PS C:\Users\ASUS\Desktop\Python Practicals>
```

EXPERIMENT 11.7

Aim: Write a python program to search for a string in text file.

Code:

```python
def search_string_in_file(filename, search_string):
    # Open the file for reading
    with open(filename, 'r') as file:
        # Read each line in the file
        for line_number, line in enumerate(file, start=1):
            # Check if the search string is in the current line
            if search_string in line:
                print(f"Found '{search_string}' in '{filename}' at line
{line_number}:")
                print(line.strip())

# Test the function
filename = "sample.txt"  # Replace with the path to your text file
search_string = "apple"  # The string to search for
search_string_in_file(filename, search_string)
```

OUTPUT :

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS\Desktop\Python Practicals> & C:/Users/ASUS/App
py"
Found 'apple' in 'sample.txt' at line 1:
apple banana banana cherry cherry cherry cherry cherry
PS C:\Users\ASUS\Desktop\Python Practicals>
```

EXPERIMENT 12.1

Aim: Write a python code to read a csv file using pandas' module and print the first and last five lines of a file.

Code :

```
import pandas as pd

def create_csv_and_read():
    # Create a DataFrame
    data = {
        'Name': ['John', 'Alice', 'Bob', 'Emily', 'David'],
        'Age': [25, 30, 35, 40, 45],
        'City': ['New York', 'Los Angeles', 'Chicago', 'Houston',
'Phoenix']
    }
    df = pd.DataFrame(data)

    # Write the DataFrame to a CSV file
    filename = "data.csv"
    df.to_csv(filename, index=False)
    print(f"CSV file '{filename}' created successfully.")

    # Read the CSV file into a DataFrame
    df_read = pd.read_csv(filename)

    # Print the DataFrame
    print("Data read from CSV file:")
    print(df_read)

# Test the function
create_csv_and_read()
```

OUTPUT :

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS\Desktop\Python Practicals> & C:/Users/ASUS/AppData/
CSV file 'data.csv' created successfully.
Data read from CSV file:
    Name  Age         City
0   John   25     New York
1  Alice   30  Los Angeles
2    Bob   35      Chicago
3  Emily   40      Houston
4  David   45      Phoenix
PS C:\Users\ASUS\Desktop\Python Practicals>
```

EXPERIMENT 12.2

Aim: Write a python program to create, write and read CSV files Into a Dictionary.

Code :

```python
import csv

def write_csv_from_dict(filename, data):
    # Write data to a CSV file from a dictionary
    with open(filename, 'w', newline='') as csvfile:
        fieldnames = data[0].keys()
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

        writer.writeheader()
        for row in data:
            writer.writerow(row)

def read_csv_to_dict(filename):
    # Read data from a CSV file into a dictionary
    data = []
    with open(filename, 'r') as csvfile:
        reader = csv.DictReader(csvfile)
```

```python
        for row in reader:
            data.append(row)
    return data

# Test the functions
filename = "data.csv"

# Create data as a list of dictionaries
data_to_write = [
    {"Name": "John", "Age": 25, "City": "New York"},
    {"Name": "Alice", "Age": 30, "City": "Los Angeles"},
    {"Name": "Bob", "Age": 35, "City": "Chicago"},
    {"Name": "Emily", "Age": 40, "City": "Houston"},
    {"Name": "David", "Age": 45, "City": "Phoenix"}
]

# Write data to CSV file
write_csv_from_dict(filename, data_to_write)
print(f"Data written to '{filename}' successfully.")

# Read data from CSV file
data_read = read_csv_to_dict(filename)
print("Data read from CSV file:")
for row in data_read:
    print(row)
```

OUTPUT :

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS\Desktop\Python Practicals> & C:/Users/ASUS/AppData/
py"
Data written to 'data.csv' successfully.
Data read from CSV file:
{'Name': 'John', 'Age': '25', 'City': 'New York'}
{'Name': 'Alice', 'Age': '30', 'City': 'Los Angeles'}
{'Name': 'Bob', 'Age': '35', 'City': 'Chicago'}
{'Name': 'Emily', 'Age': '40', 'City': 'Houston'}
{'Name': 'David', 'Age': '45', 'City': 'Phoenix'}
PS C:\Users\ASUS\Desktop\Python Practicals>
```

EXPERIMENT 13.1

Aim: Write a python program to program to connect with MySQL database.

Code:

```
import mysql.connector

def connect_to_mysql(host, user, password, database):
    try:
        # Establish a connection to the MySQL database
        connection = mysql.connector.connect(
            host=host,
            user=user,
            password=password,
            database=database,
            auth_plugin='mysql_native_password'  # Specify the
authentication plugin
        )
        print("Connected to MySQL database successfully.")
        return connection
    except mysql.connector.Error as err:
        print(f"Error: {err}")
        return None

# Test the function
host = "localhost"  # Hostname of the MySQL server
user = "root"    # MySQL username
password = "Shreyansh7"  # MySQL password
database = "exampledb"  # Name of the database you want to connect to

connection = connect_to_mysql(host, user, password, database)
```

OUTPUT :

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS\Desktop\Python Practicals> & C:/Users/ASUS/AppData/
py"
Data written to 'data.csv' successfully.
Data read from CSV file:
{'Name': 'John', 'Age': '25', 'City': 'New York'}
{'Name': 'Alice', 'Age': '30', 'City': 'Los Angeles'}
{'Name': 'Bob', 'Age': '35', 'City': 'Chicago'}
{'Name': 'Emily', 'Age': '40', 'City': 'Houston'}
{'Name': 'David', 'Age': '45', 'City': 'Phoenix'}
PS C:\Users\ASUS\Desktop\Python Practicals>
```

EXPERIMENT 13.2

Aim: Write a python program to perform following database operations:
i) create ii) alter iii) insert iv) update v) drop vi) delete

Code :

```python
import mysql.connector

# Function to connect to MySQL database
def connect_to_mysql():
    try:
        connection = mysql.connector.connect(
            host="localhost",
            user="root",
            password="Shreyansh7",
            database="exampledb"
        )
        print("Connected to MySQL database successfully.")
        return connection
    except mysql.connector.Error as err:
        print(f"Error: {err}")
```

```
        return None

# Function to create table
def create_table(connection):
    try:
        cursor = connection.cursor()
        cursor.execute("CREATE TABLE IF NOT EXISTS example_table (id INT
AUTO_INCREMENT PRIMARY KEY, name VARCHAR(255), age INT)")
        print("Table created successfully.")
        print_table(connection)
        print("\n")
    except mysql.connector.Error as err:
        print(f"Error: {err}")

# Function to alter table
def alter_table(connection):
    try:
        cursor = connection.cursor()
        cursor.execute("ALTER TABLE example_table ADD COLUMN email
VARCHAR(255)")
        print("Table altered successfully.")
        print_table(connection)
        print("\n")
    except mysql.connector.Error as err:
        print(f"Error: {err}")

# Function to insert into table
def insert_into_table(connection):
    try:
        cursor = connection.cursor()
        sql = "INSERT INTO example_table (name, age, email) VALUES (%s,
%s, %s)"
        val = [("John", 30, "john@example.com"), ("Alice", 25,
"alice@example.com"), ("Bob", 35, "bob@example.com")]
        cursor.executemany(sql, val)
        connection.commit()
        print(cursor.rowcount, "record(s) inserted.")
        print_table(connection)
        print("\n")
    except mysql.connector.Error as err:
        print(f"Error: {err}")

# Function to update table
def update_table(connection):
    try:
        cursor = connection.cursor()
        sql = "UPDATE example_table SET age = %s WHERE name = %s"
```

```python
        val = (40, "John")
        cursor.execute(sql, val)
        connection.commit()
        print(cursor.rowcount, "record(s) updated.")
        print_table(connection)
        print("\n")
    except mysql.connector.Error as err:
        print(f"Error: {err}")

# Function to drop table
def drop_table(connection):
    try:
        cursor = connection.cursor()
        cursor.execute("DROP TABLE IF EXISTS example_table")
        print("Table dropped successfully.")
        print("\n")
    except mysql.connector.Error as err:
        print(f"Error: {err}")

# Function to print table contents
def print_table(connection):
    try:
        cursor = connection.cursor()
        cursor.execute("SELECT * FROM example_table")
        result = cursor.fetchall()
        print("Table contents:")
        for row in result:
            print(row)
    except mysql.connector.Error as err:
        print(f"Error: {err}")

# Main function
def main():
    connection = connect_to_mysql()
    if connection:
        create_table(connection)
        alter_table(connection)
        insert_into_table(connection)
        update_table(connection)
        drop_table(connection)
        connection.close()
        print("Connection closed.")

if __name__ == "__main__":
    main()
```

OUTPUT :

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS\Desktop\Python Practicals> & C:/Users/ASUS/AppData/Local/Progr
py"
Connected to MySQL database successfully.
Table created successfully.
Table contents:


Table altered successfully.
Table contents:


3 record(s) inserted.
Table contents:
(1, 'John', 30, 'john@example.com')
(2, 'Alice', 25, 'alice@example.com')
(3, 'Bob', 35, 'bob@example.com')


1 record(s) updated.
Table contents:
(1, 'John', 40, 'john@example.com')
(2, 'Alice', 25, 'alice@example.com')
(3, 'Bob', 35, 'bob@example.com')


Table dropped successfully.


Connection closed.
PS C:\Users\ASUS\Desktop\Python Practicals>
```

EXPERIMENT 13.3

Aim: Write a python program to perform following database operation:
delete

Code :

```
import mysql.connector
```

```python
# Function to connect to MySQL database
def connect_to_mysql():
    try:
        connection = mysql.connector.connect(
            host="localhost",
            user="root",
            password="Shreyansh7",
            database="exampledb"
        )
        print("Connected to MySQL database successfully.")
        return connection
    except mysql.connector.Error as err:
        print(f"Error: {err}")
        return None

# Function to delete from table
def delete_from_table(connection):
    try:
        cursor = connection.cursor()
        sql = "DELETE FROM example_table WHERE name = %s"
        val = ("John",)
        cursor.execute(sql, val)
        connection.commit()
        print(cursor.rowcount, "record(s) deleted.")
        print_table(connection)
        print("\n")
    except mysql.connector.Error as err:
        print(f"Error: {err}")

# Function to print table contents
def print_table(connection):
    try:
        cursor = connection.cursor()
        cursor.execute("SELECT * FROM example_table")
        result = cursor.fetchall()
        print("Table contents:")
        for row in result:
            print(row)
    except mysql.connector.Error as err:
        print(f"Error: {err}")

# Main function
def main():
    connection = connect_to_mysql()
    if connection:
        delete_from_table(connection)
```

```
        connection.close()
        print("Connection closed.")

if __name__ == "__main__":
    main()
```

OUTPUT :

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS\Desktop\Python Practicals> & C:/Users/ASUS/AppData
on.py"
Connected to MySQL database successfully.
1 record(s) deleted.
Table contents:
(2, 'Alice', 25, 'alice@example.com')


Connection closed.
PS C:\Users\ASUS\Desktop\Python Practicals>
```

END.