Ex.No: 1 CREATE A SIMPLE CALCULATOR TO DO ALL THE ARITHMETIC OPERATIONS.

AIM:

To create a simple calculator that can perform basic arithmetic operations (addition, subtraction, multiplication, and division) based on user input.

ALGORITHM:

Step1: Display a menu with options for different arithmetic operations.

Step 2: Take user input for the desired operation (1 for addition, 2 for subtraction, 3 for multiplication, 4 for division).

Step 3: Take user input for two numbers.

Step 4: Perform the selected operation on the input numbers.

Step 5: Display the result.

CODING:

```
# Simple Calculator Program
def add(x, y):
    return x + y
def subtract(x, y):
    return x - y
def multiply(x, y):
    return x * y
def divide(x, y):
    if y != 0:
        return x / y
    else:
        return "Cannot divide by zero."
def calculator():
    print("Simple Calculator")
    print("Select operation:")
    print("1. Addition")
    print("2. Subtraction")
    print("3. Multiplication")
    print("4. Division")
    choice = input("Enter choice (1/2/3/4): ")
    if choice in ('1', '2', '3', '4'):
        num1 = float(input("Enter first number: "))
        num2 = float(input("Enter second number: "))
        if choice == '1':
            result = add(num1, num2)
            operator = "+"
        elif choice == '2':
            result = subtract(num1, num2)
            operator = "-"
        elif choice == '3':
            result = multiply(num1, num2)
            operator = "*"
        elif choice == '4':
            result = divide(num1, num2)
            operator = "/"
        print(f"{num1} {operator} {num2} = {result}")
    else:
        print("Invalid input. Please enter a valid operation (1/2/3/4).")

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

SAMPLE INPUT / OUTPUT:

```
Simple Calculator
Select operation:
1. Addition
2. Subtraction
3. Multiplication
4. Division
Enter choice (1/2/3/4): 3
Enter first number: 4
Enter second number: 5
4.0 * 5.0 = 20.0
```

RESULT:

Thus the above program is verified and executed successfully.

Ex.No: 2 WRITE A PROGRAM TO USE CONTROL FLOW TOOLS LIKE IF.

AIM:

Determine whether a given number is positive, negative, or zero.

ALGORITHM:

Step 1: Start the program

Step 2: Take user input for a number.

Step 3: Use an if statement to check the sign of the number.

Step 4: Print whether the number is positive, negative, or zero.

Step 5: Stop the program.

CODING:

```
number = float(input("Enter a number: "))

if number > 0:
    print(f"{number} is a positive number.")
elif number < 0:
    print(f"{number} is a negative number.")
else:
    print("The number is zero.")
```

SAMPLE INPUT / OUTPUT:

```
Enter a number: -5
-5.0 is a negative number.
```

RESULT:

Thus the above program is verified and executed successfully.

Ex.No: 3 WRITE A PROGRAM TO USE FOR LOOP.

AIM:

To write a program to Calculate the sum of the first n natural numbers.

ALGORITHM:

Step 1: Start the program.

Step 2: Take user input for n (the number of natural numbers).

Step 3: Initialize a variable sum to store the sum.

Step 4: Use a for loop to iterate through the range from 1 to n (inclusive).

Step 5: Add each number to the sum.

Step 6: Print the sum.

Step 7: Stop the program

CODING:

```
n = int(input("Enter the value of n: "))

# Sum calculation using for loop
sum_of_numbers = 0
for i in range(1, n + 1):
    sum_of_numbers += i

print(f"The sum of the first {n} natural numbers is: {sum_of_numbers}")
```

SAMPLE INPUT / OUTPUT:

```
Enter the value of n: 5
The sum of the first 5 natural numbers is: 15
```

RESULT:

Thus the above program is verified and executed successfully.

Ex.No: 4 **DATA STRUCTURES**

A. USE LIST AS STACK. B. USE LIST AS QUEUE. C. TUPLE, SEQUENCE.

AIM:

To demonstrate how to use a list as a stack.

ALGORITHM:

A. USE LIST AS STACK

STEP 1: Start the program

STEP 2: Initialize an empty list to represent the stack.

STEP 3: Use append() to push elements onto the stack.

STEP 4: Use pop() to remove elements from the top of the stack.

STEP 5: Stop the program

CODING:

```
# Using list as a stack

# Initialize an empty list as a stack
stack = []

# Push elements onto the stack
stack.append(1)
stack.append(2)
stack.append(3)

# Pop elements from the top of the stack
popped_element = stack.pop()

# Output
print("Stack:", stack)
print("Popped Element:", popped_element)
```

SAMPLE INPUT / OUTPUT:

```
Stack: [1, 2]
Popped Element: 3
```

B. USE LIST AS QUEUE

AIM:

To demonstrate how to use a list as a queue.

ALGORITHM:

Step 1: Start the program.

Step 2: Initialize an empty list to represent the queue.

Step 3: Use append() to enqueue elements.

Step 4: Use pop(0) to dequeue elements.

Step 5: Stop the program.

CODING:

```
# Using list as a queue

# Initialize an empty list as a queue
queue = []

# Enqueue elements
queue.append(1)
queue.append(2)
queue.append(3)

# Dequeue elements
dequeued_element = queue.pop(0)

# Output
print("Queue:", queue)
print("Dequeued Element:", dequeued_element)
```

SAMPLE INPUT / OUTPUT:

```
Queue: [2, 3]
Dequeued Element: 1
```

C. TUPLE, SEQUENCE

AIM:

To demonstrate the use of a tuple as a sequence.

ALGORITHM:

```
Step 1: Start the program
Step 2: Initialize a tuple with a sequence of elements.
Step 3: Access elements using indexing.
Step 4: Iterate through the tuple using a for loop.
Step 5: Stop the program.
```

CODING:

```
# Using tuple as a sequence

# Initialize a tuple
my_tuple = (1, 2, 3, 4, 5)

# Access elements using indexing
element_at_index_2 = my_tuple[2]

# Iterate through the tuple
for element in my_tuple:
    print(element)

# Output
print("Element at index 2:", element_at_index_2)
```

SAMPLE INPUT / OUTPUT:

```
1
2
3
4
5
Element at index 2: 3
```

RESULT:

Thus above program is verified and executed successfully.

Ex.No: 5 CREATE NEW MODULE FOR MATHEMATICAL OPERATIONS AND USE IN YOUR PROGRAM.

AIM:

To create a module for basic mathematical operations.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Define functions for addition, subtraction, multiplication, and division in the module.

Coding: Create a file named math_operations.py:

STEP 3: Import the math_operations module.

STEP 4: Take user input for two numbers.

STEP 5: Perform mathematical operations using functions from the module.

STEP 6: Print the results.

STEP 7: Stop the program.

CODING:

```
        # math_operations.py
def add(x, y):
   return x + y
def subtract(x, y):
   return x - y
def multiply(x, y):
   return x * y
def divide(x, y):
   if y != 0:
      return x / y
   else:
      return "Cannot divide by zero."

# Program using the mathematical operations module
# Import the mathematical operations module
import math_operations as math_ops
```

```
# Input
num1 = float(input("Enter the first number: "))
num2 = float(input("Enter the second number: "))

# Perform mathematical operations
sum_result = math_ops.add(num1, num2)
difference_result = math_ops.subtract(num1, num2)
product_result = math_ops.multiply(num1, num2)
division_result = math_ops.divide(num1, num2)

# Output
print(f"Sum: {sum_result}")
print(f"Difference: {difference_result}")
print(f"Product: {product_result}")
print(f"Division: {division_result}")
```

SAMPLE INPUT / OUTPUT:

```
Enter the first number: 10
Enter the second number: 5
Sum: 15.0
Difference: 5.0
Product: 50.0
Division: 2.0
```

RESULT:

Thus the above program is verified and executed successfully.

Ex.No: 6 WRITE A PROGRAM TO READ AND WRITE FILES, CREATE AND DELETE DIRECTORIES.

AIM:

The aim of this program is to provide a simple command-line interface for basic file and directory operations. The program allows the user to read the contents of a file, write to a file, create a new directory, and delete an existing directory.

ALGORITHM:

STEP 1: Read File Function (read_file):

- Take the filename as input from the user.
- Attempt to open the file in read mode using a try-except block.
- If the file is found, read its content and display it. If not, handle the FileNotFoundError by notifying the user.

STEP 2: Write to File Function (write_to_file):

- Take the filename and content as input from the user.
- Open the file in write mode and write the provided content to it.
- Notify the user upon successful writing.

STEP 3: Create Directory Function (create_directory):

- Take the directory name as input from the user.
- Use os.mkdir to create the specified directory.
- Handle the FileExistsError by notifying the user if the directory already exists.

STEP 4: Delete Directory Function (delete_directory):

- Take the directory name as input from the user.
- Use os.rmdir to delete the specified directory.
- Handle FileNotFoundError by notifying the user if the directory is not found.
- Handle other OSError exceptions by displaying an error message.

CODING:

```
import os
def read_file():
    filename = input("Enter the name of the file to read: ")
    try:
        with open(filename, 'r') as file:
            content = file.read()
            print("\nFile content:\n", content)
    except FileNotFoundError:
        print(f"File '{filename}' not found.")

def write_to_file():
    filename = input("Enter the name of the file to write: ")
    content = input("Enter the content to write to the file: ")
    with open(filename, 'w') as file:
        file.write(content)
    print(f"Content successfully written to '{filename}'.")

def create_directory():
    dirname = input("Enter the name of the directory to create: ")
    try:
        os.mkdir(dirname)
        print(f"Directory '{dirname}' created successfully.")
    except FileExistsError:
        print(f"Directory '{dirname}' already exists.")


def delete_directory():
    dirname = input("Enter the name of the directory to delete: ")
    try:
        os.rmdir(dirname)
        print(f"Directory '{dirname}' deleted successfully.")
    except FileNotFoundError:
        print(f"Directory '{dirname}' not found.")
    except OSError as e:
        print(f"Error deleting directory '{dirname}': {e}")
```

```
# Sample Input & Output
read_file()
write_to_file()
create_directory()
delete_directory()
```

SAMPLE INPUT / OUTPUT:

```
Enter the name of the file to read: sample.txt
File 'sample.txt' not found.

Enter the name of the file to write: new_file.txt
Enter the content to write to the file: This is a new file content.

Content successfully written to 'new_file.txt'.

Enter the name of the directory to create: new_directory
Directory 'new_directory' created successfully.

Enter the name of the directory to delete: new_directory
Directory 'new_directory' deleted successfully.
```

RESULT:

Thus the above program is verified and executed successfully.

Ex.No: 7 WRITE A PROGRAM WITH EXCEPTION HANDLING.

AIM:

The aim of this program is to demonstrate the use of exception handling in Python. The program will perform a division operation and handle possible exceptions that may occur during the execution.

ALGORITHM:

STEP1 : Input:

- Take two numbers as input from the user.

STEP 2: Exception Handling:

- Use a try-except block to handle potential exceptions during the division operation.
- Handle ZeroDivisionError if the user attempts to divide by zero.
- Handle ValueError if the input is not a valid number.

STEP 3: Division Operation:

- Perform the division operation inside the try block if no exceptions occur.

STEP 4: Output:

- Display the result of the division if successful.
- Display an error message if any exceptions are caught.

CODING:

```
      def perform_division():
  try:
    # Input
    numerator = float(input("Enter the numerator: "))
    denominator = float(input("Enter the denominator: "))
    # Division Operation
    result = numerator / denominator
    # Output
    print(f"Result of {numerator} / {denominator} = {result}")

  except ZeroDivisionError:
    print("Error: Cannot divide by zero.")
  except ValueError:
    print("Error: Please enter valid numbers.")
```

```
# Sample Input & Output
perform_division()
```

SAMPLE INPUT / OUTPUT:

```
Enter the numerator: 10
Enter the denominator: 2
Result of 10.0 / 2.0 = 5.0

Enter the numerator: 8
Enter the denominator: 0
Error: Cannot divide by zero.

Enter the numerator: abc
Error: Please enter valid numbers.
```

RESULT:

Thus the above program is verified and executed successfully.

Ex.No: 8 WRITE A PROGRAM USING CLASSES.

AIM:

The aim of this program is to demonstrate the use of classes in Python. The program will define a simple class representing a book, and instances of this class will be used to manage information about different books.

ALGORITHM:

STEP 1: Define the Book Class:

- Create a class named Book with attributes like title, author, and publication year.
- Implement a method within the class to display information about the book.

STEP 2: Create Book Instances:

- Instantiate multiple objects of the Book class, representing different books.

STEP 3: Access and Display Information:

- Access the attributes of each book object and display their information.

CODING:

```
class Book:
    def __init__(self, title, author, publication_year):
        self.title = title
        self.author = author
        self.publication_year = publication_year

def display_info(self):
        print(f"Title: {self.title}")
        print(f"Author: {self.author}")
        print(f"Publication Year: {self.publication_year}")
        print()

# Sample Input & Output
book1 = Book("The Great Gatsby", "F. Scott Fitzgerald", 1925)
book2 = Book("To Kill a Mockingbird", "Harper Lee", 1960)
book3 = Book("1984", "George Orwell", 1949)
```

```
# Display information about each book
print("Information about Book 1:")
book1.display_info()
print("Information about Book 2:")
book2.display_info()
print("Information about Book 3:")
book3.display_info()
```

SAMPLE INPUT / OUTPUT:

```
Information about Book 1:
Title: The Great Gatsby
Author: F. Scott Fitzgerald
Publication Year: 1925
Information about Book 2:
Title: To Kill a Mockingbird
Author: Harper Lee
Publication Year: 1960
Information about Book 3:
Title: 1984
Author: George Orwell
Publication Year: 1949
```

RESULT:

Thus the above program is verified and executed successfully.

Ex.No: 9 CONNECT WITH MYSQL AND CREATE ADDRESS BOOK.

AIM:

The aim of this program is to connect to a MySQL database and create an address book. The address book will store information about contacts, such as name, email, and phone number.

ALGORITHM:

STEP 1: Install Required Module:

- Install the mysql-connector-python module if it's not already installed. You can install it using pip install mysql-connector-python.

STEP 2: Connect to MySQL Database:

- Use the mysql.connector module to connect to a MySQL database.

STEP 3: Create a Table for Address Book:

- Execute a SQL query to create a table named contacts with columns for name, email, and phone number.

STEP 4: Insert Contacts into Address Book:

- Allow the user to input contact information (name, email, phone number) and insert it into the contacts table.

STEP 5: Retrieve and Display Contacts:

- Fetch and display the contacts from the contacts table.

CODING:

```
import mysql.connector
class AddressBook:
    def __init__(self, host, user, password, database):
        self.connection = mysql.connector.connect(
            host=host,
            user=user,
            password=password,
            database=database
        )
        self.cursor = self.connection.cursor()
        self.create_table()
```

```
def create_table(self):
    # SQL query to create a 'contacts' table
    create_table_query = """
    CREATE TABLE IF NOT EXISTS contacts (
        id INT AUTO_INCREMENT PRIMARY KEY,
        name VARCHAR(255),
        email VARCHAR(255),
        phone_number VARCHAR(20)
    )
    """
    self.cursor.execute(create_table_query)
    self.connection.commit()

def insert_contact(self, name, email, phone_number):
    # SQL query to insert a contact into the 'contacts' table
    insert_query = "INSERT INTO contacts (name, email, phone_number) VALUES (%s, %s, %s)"
    contact_data = (name, email, phone_number)
    self.cursor.execute(insert_query, contact_data)
    self.connection.commit()
    print("Contact added successfully.")

def display_contacts(self):
    # SQL query to retrieve all contacts from the 'contacts' table
    select_query = "SELECT * FROM contacts"
    self.cursor.execute(select_query)
    contacts = self.cursor.fetchall()

    # Display contacts
    print("\nAddress Book:")
    for contact in contacts:
        print(f"ID: {contact[0]}, Name: {contact[1]}, Email: {contact[2]}, Phone: {contact[3]}")
    print()
    def close_connection(self):
        # Close the cursor and connection
        self.cursor.close()
        self.connection.close()

# Sample Input & Output
address_book = AddressBook(
    host="your_mysql_host",
    user="your_mysql_user",
    password="your_mysql_password",
    database="your_mysql_database"
)

# Insert contacts
address_book.insert_contact("John Doe", "john.doe@example.com", "123-456-7890")
address_book.insert_contact("Jane Smith", "jane.smith@example.com", "987-654-3210")



# Display contacts
address_book.display_contacts()

# Close the connection
address_book.close_connection()
```

SAMPLE INPUT/OUTPUT:

RESULT:

Thus the above program is verified and executed successfully.

Ex.No: 10 WRITE A PROGRAM USING STRING HANDLING AND REGULAR EXPRESSIONS.

AIM:

The aim of this program is to demonstrate the use of string handling and regular expressions in Python. The program will validate and extract information from strings using regular expressions.

ALGORITHM:

STEP 1: Import Required Modules:

- Import the re module for regular expressions.
- Define a Regular Expression Pattern:
- Define a regular expression pattern to match a specific format in the input string. In this example, we will validate and extract email addresses.

STEP2: Input:

- Take an email address as input from the user.

STEP 3: String Handling with Regular Expressions:

- Use the re.match function to check if the input string matches the defined pattern.
- Use the re.findall function to extract information from the input string based on the regular expression pattern.

STEP 4: Output:

- Display whether the input string is valid or not.
- If valid, display the extracted information.

CODING:

```
import re

def validate_and_extract_email(input_email):
    # Regular expression pattern for a simple email validation
    email_pattern = r'^\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b'

    # Check if the input email matches the pattern
    match_result = re.match(email_pattern, input_email)
    if match_result:
        print(f"The email '{input_email}' is valid.")

        # Extract information from the email using regular expressions
        extracted_info = re.findall(r'([A-Za-z0-9._%+-]+)@([A-Za-z0-9.-]+)\.([A-Z|a-z]{2,})', input_email)

        # Display extracted information
        print("Username:", extracted_info[0][0])
        print("Domain:", extracted_info[0][1])
        print("Top-level Domain:", extracted_info[0][2])
    else:
        print(f"The email '{input_email}' is not valid.")

# Sample Input & Output
input_email = input("Enter an email address: ")
validate_and_extract_email(input_email)
```

SAMPLE INPUT/OUTPUT:

```
Enter an email address: john.doe@example.com
The email 'john.doe@example.com' is valid.
Username: john.doe
Domain: example
Top-level Domain: com
Enter an email address: invalid_email
The email 'invalid_email' is not valid.
```

RESULT:

Thus the above program is verified and executed successfully.