Name:- Milind Kailas Tajane
Roll No:- CS061

Date:-_____

Practical No:1

AIM:- Write a python program to display all types of pyramids of stars.

CODE:-

```
def print_right_angle_triangle(n):
    print("Right-Angle Triangle:")
    for i in range(1, n + 1):
        print('*' * i)
    print()

def print_isosceles_triangle(n):
    print("Isosceles Triangle:")
    for i in range(1, n + 1):
        print(' ' * (n - i) + '*' * (2 * i - 1))
    print()

def print_inverted_triangle(n):
    print("Inverted Triangle:")
    for i in range(n, 0, -1):
        print('*' * i)
    print()

def print_full_pyramid(n):
    print("Full Pyramid:")
    for i in range(1, n + 1):
        print(' ' * (n - i) + '*' * (2 * i - 1))
    print()

def print_diamond(n):
    print("Diamond Shape:")
    # Upper part
    for i in range(1, n + 1):
        print(' ' * (n - i) + '*' * (2 * i - 1))
    # Lower part
    for i in range(n - 1, 0, -1):
        print(' ' * (n - i) + '*' * (2 * i - 1))
    print()

# Set the height of the pyramids
n = 5
```

print_right_angle_triangle(n)
print_isosceles_triangle(n)
print_inverted_triangle(n)
print_full_pyramid(n)
print_diamond(n)

Output:-

```
Right-Angle Triangle:
*
**
***
****
*****

Isosceles Triangle:
    *
   ***
  *****
 *******
*********

Inverted Triangle:
*****
****
***
**
*

Full Pyramid:
    *
   ***
  *****
 *******
*********

Diamond Shape:
    *
   ***
  *****
 *******
*********
 *******
  *****
   ***
    *
```

Name:- Milind Kailas Tajane
Roll No:- CS061

Date:-_____

Practical No:2

AIM:- Write a program to display multiplication table of all numbers from 1 to 10.

CODE:-

```
# Loop through numbers 1 to 10 for the multiplication table
for i in range(1, 11):
    # Print a header for the current multiplication table
    print("\n\nMULTIPLICATION TABLE FOR %d\n" % (i))
    # Loop through numbers 1 to 10 for the multipliers
    for j in range(1, 11):
        # Print the multiplication expression and result with formatted output
        print("%-5d X %5d = %5d" % (i, j, i * j))
```

Output:-

```
MULTIPLICATION TABLE FOR 1
1     X     1 =     1
1     X     2 =     2
1     X     3 =     3
1     X     4 =     4
1     X     5 =     5
1     X     6 =     6
1     X     7 =     7
1     X     8 =     8
1     X     9 =     9
1     X    10 =    10


MULTIPLICATION TABLE FOR 2
2     X     1 =     2
2     X     2 =     4
2     X     3 =     6
2     X     4 =     8
2     X     5 =    10
2     X     6 =    12
2     X     7 =    14
2     X     8 =    16
2     X     9 =    18
2     X    10 =    20


MULTIPLICATION TABLE FOR 3
3     X     1 =     3
3     X     2 =     6
3     X     3 =     9
3     X     4 =    12
3     X     5 =    15
3     X     6 =    18
3     X     7 =    21
3     X     8 =    24
3     X     9 =    27
3     X    10 =    30


MULTIPLICATION TABLE FOR 4
4     X     1 =     4
4     X     2 =     8
4     X     3 =    12
4     X     4 =    16
4     X     5 =    20
4     X     6 =    24
4     X     7 =    28
4     X     8 =    32
4     X     9 =    36
4     X    10 =    40


MULTIPLICATION TABLE FOR 5
5     X     1 =     5
5     X     2 =    10
5     X     3 =    15
5     X     4 =    20
5     X     5 =    25
5     X     6 =    30
5     X     7 =    35
5     X     8 =    40
5     X     9 =    45
5     X    10 =    50


MULTIPLICATION TABLE FOR 6
6     X     1 =     6
6     X     2 =    12
6     X     3 =    18
6     X     4 =    24
6     X     5 =    30
6     X     6 =    36
6     X     7 =    42
6     X     8 =    48
6     X     9 =    54
6     X    10 =    60


MULTIPLICATION TABLE FOR 7
7     X     1 =     7
7     X     2 =    14
7     X     3 =    21
7     X     4 =    28
7     X     5 =    35
7     X     6 =    42
7     X     7 =    49
7     X     8 =    56
7     X     9 =    63
7     X    10 =    70


MULTIPLICATION TABLE FOR 8
8     X     1 =     8
8     X     2 =    16
8     X     3 =    24
8     X     4 =    32
8     X     5 =    40
8     X     6 =    48
8     X     7 =    56
8     X     8 =    64
8     X     9 =    72
8     X    10 =    80


MULTIPLICATION TABLE FOR 9
9     X     1 =     9
9     X     2 =    18
9     X     3 =    27
9     X     4 =    36
9     X     5 =    45
9     X     6 =    54
9     X     7 =    63
9     X     8 =    72
9     X     9 =    81
9     X    10 =    90


MULTIPLICATION TABLE FOR 10
10    X     1 =    10
10    X     2 =    20
10    X     3 =    30
10    X     4 =    40
10    X     5 =    50
10    X     6 =    60
10    X     7 =    70
10    X     8 =    80
10    X     9 =    90
10    X    10 =   100
```

Name:- Milind Kailas Tajane
Roll No:- CS061

Date:-_____

Practical No: 3

AIM:- Write a program to implement tower of Hanoi.

CODE:-

```
def tower_of_hanoi(n, source, destination, auxiliary):

    if n == 1:
        print(f"Move disk 1 from {source} to {destination}")
        return

    # Move n-1 disks from source to auxiliary, using destination as auxiliary.
    tower_of_hanoi(n - 1, source, auxiliary, destination)

    # Move the nth disk from source to destination.
    print(f"Move disk {n} from {source} to {destination}")

    # Move the n-1 disks from auxiliary to destination, using source as auxiliary.
    tower_of_hanoi(n - 1, auxiliary, destination, source)

# Example usage
num_disks = int(input("Enter the number of disks: "))
tower_of_hanoi(num_disks, "A", "C", "B")
```

Output:-

```
Enter the number of disks: 3
Move disk 1 from A to C
Move disk 2 from A to B
Move disk 1 from C to B
Move disk 3 from A to C
Move disk 1 from B to A
Move disk 2 from B to C
Move disk 1 from A to C
```

Name:- Milind Kailas Tajane

Roll No:- CS061

Date:-_____

Practical No: 4

AIM:- Write a program to calculate simple interest using a user defined function.Accept amount, duration from user. Set interest rate as default parameter.

CODE:-

```
# Function to calculate simple interest
def calculate_simple_interest(principal, duration, rate=5.0):
    """
    Calculate simple interest using the formula: (P * R * T) / 100
    :param principal: Principal amount (float)
    :param duration: Duration in years (float)
    :param rate: Rate of interest in % (float, default is 5.0)
    :return: Simple interest (float)
    """
    return (principal * rate * duration) / 100

# Prompt the user to enter the principal amount and convert it to a float
principal = float(input("Enter the principal amount: "))

# Prompt the user to enter the duration in years and convert it to a float
duration = float(input("Enter the duration (in years): "))

# Calculate simple interest using the user inputs and default rate
simple_interest = calculate_simple_interest(principal, duration)

# Print the calculated simple interest, formatted to two decimal places
print(f"The simple interest is: {simple_interest:.2f}")
```

Output:-

```
Enter the principal amount: 1000
Enter the duration (in years): 2
The simple interest is: 100.00
```

Name:- Milind Kailas Tajane

Roll No:- CS061

Date:-_____

Practical No: 5

AIM:- Write a program to count even and odd number in the list.

CODE:-

```
# Function to count even and odd numbers in a list
def count_even_odd(numbers):
    # Initialize counters for even and odd numbers
    even_count = 0
    odd_count = 0

    # Iterate through each number in the provided list
    for number in numbers:
        # Check if the number is even
        if number % 2 == 0:
            even_count += 1  # Increment even count
        else:
            odd_count += 1   # Increment odd count

    # Return the counts of even and odd numbers
    return even_count, odd_count

# List of numbers from 1 to 20
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]

# Call the function and store the results in even_count and odd_count
even_count, odd_count = count_even_odd(numbers)

# Print the count of even numbers
print(f"Even numbers: {even_count}")

# Print the count of odd numbers
print(f"Odd numbers: {odd_count}")
```

Output:-

```
Even numbers: 10
Odd numbers: 10
```

Name:- Milind Kailas Tajane

Roll No:- CS061

Date:-_____

Practical No: 6

AIM:- Write a program to find sum of all numbers, min, max, mean, median, mode of numbers in a list.

CODE:-

```
from collections import Counter
def calculate_statistics(numbers):
    if not numbers:
        return None, None, None, None, None
    # Calculate sum
    total_sum = sum(numbers)
    # Calculate minimum
    minimum = min(numbers)

    # Calculate maximum
    maximum = max(numbers)

    # Calculate mean
    mean = total_sum / len(numbers)
    # Calculate mode
    frequency = Counter(numbers)
    mode_data = frequency.most_common()
    mode = [num for num, freq in mode_data if freq == mode_data[0][1]]
    return total_sum, minimum, maximum, mean, mode
# Example usage
numbers = [1, 2, 2, 3, 4, 4, 4, 5]
total_sum, minimum, maximum, mean, mode = calculate_statistics(numbers)
print(f"Sum: {total_sum}")
print(f"Min: {minimum}")
print(f"Max: {maximum}")
print(f"Mean: {mean}")
print(f"Mode: {mode}")
```

Output:-

```
Sum: 25
Min: 1
Max: 5
Mean: 3.125
Mode: [4]
```

Name:- Milind Kailas Tajane

Roll No:- CS061

Date:-_____

Practical No:7

AIM:- Write a program to store student roll number and marks using dictionary.
Implements following functions:-
Add a record, delete, update marks, search a roll number and display marks, sort the records in ascending and descending order, display student information with highest marks. Implement a menu driven program.

CODE:-

```
# Dictionary to store student records
student_records = {}

# Function to add a new record
def add_record(roll_number, marks):
    if roll_number in student_records:
        print("Roll number already exists!")
    else:
        student_records[roll_number] = marks
        print("Record added successfully!")

# Function to delete a record
def delete_record(roll_number):
    if roll_number in student_records:
        del student_records[roll_number]
        print("Record deleted successfully!")
    else:
        print("Roll number not found!")

# Function to update marks
def update_marks(roll_number, marks):
    if roll_number in student_records:
        student_records[roll_number] = marks
        print("Marks updated successfully!")
    else:
        print("Roll number not found!")

# Function to search for a roll number and display marks
def search_roll_number(roll_number):
```

```
    if roll_number in student_records:
        print(f"Roll Number: {roll_number}, Marks: {student_records[roll_number]}")
    else:
        print("Roll number not found!")

# Function to display all records sorted in ascending order
def display_sorted_ascending():
    sorted_records = sorted(student_records.items())
    print("Records in ascending order:")
    for roll, marks in sorted_records:
        print(f"Roll Number: {roll}, Marks: {marks}")

# Function to display all records sorted in descending order
def display_sorted_descending():
    sorted_records = sorted(student_records.items(), reverse=True)
    print("Records in descending order:")
    for roll, marks in sorted_records:
        print(f"Roll Number: {roll}, Marks: {marks}")

# Function to display the student with the highest marks
def display_highest_marks():
    if student_records:
        highest = max(student_records.items(), key=lambda x: x[1])
        print(f"Student with highest marks: Roll Number: {highest[0]}, Marks: {highest[1]}")
    else:
        print("No records available!")

# Menu-driven program
def menu():
    while True:
        print("\n--- Student Record Management ---")
        print("1. Add a record")
        print("2. Delete a record")
        print("3. Update marks")
        print("4. Search a roll number")
        print("5. Display records in ascending order")
        print("6. Display records in descending order")
        print("7. Display student with highest marks")
        print("8. Exit")
        choice = input("Enter your choice (1-8): ")

        if choice == "1":
            roll = input("Enter roll number: ")
            marks = float(input("Enter marks: "))
            add_record(roll, marks)
        elif choice == "2":
            roll = input("Enter roll number to delete: ")
            delete_record(roll)
        elif choice == "3":
            roll = input("Enter roll number to update: ")
```

```
            marks = float(input("Enter new marks: "))
            update_marks(roll, marks)
        elif choice == "4":
            roll = input("Enter roll number to search: ")
            search_roll_number(roll)
        elif choice == "5":
            display_sorted_ascending()
        elif choice == "6":
            display_sorted_descending()
        elif choice == "7":
            display_highest_marks()
        elif choice == "8":
            print("Exiting program. Goodbye!")
            break
        else:
            print("Invalid choice! Please try again.")

# Run the program
menu()
```

Output:-

```
--- Student Record Management ---
1. Add a record
2. Delete a record
3. Update marks
4. Search a roll number
5. Display records in ascending order
6. Display records in descending order
7. Display student with highest marks
8. Exit
Enter your choice (1-8): 1
Enter roll number: 101
Enter marks: 85
Record added successfully!

--- Student Record Management ---
1. Add a record
2. Delete a record
3. Update marks
4. Search a roll number
5. Display records in ascending order
6. Display records in descending order
7. Display student with highest marks
8. Exit
Enter your choice (1-8): 1
Enter roll number: 102
Enter marks: 90
Record added successfully!

--- Student Record Management ---
1. Add a record
2. Delete a record
3. Update marks
4. Search a roll number
5. Display records in ascending order
6. Display records in descending order
7. Display student with highest marks
8. Exit
Enter your choice (1-8): 7
Student with highest marks: Roll Number: 102, Marks: 90.0

--- Student Record Management ---
1. Add a record
2. Delete a record
3. Update marks
4. Search a roll number
5. Display records in ascending order
6. Display records in descending order
7. Display student with highest marks
8. Exit
Enter your choice (1-8): 8
Exiting program. Goodbye!
```

Name:- Milind Kailas Tajane
Roll No:- CS061

Date:-_____

Practical No:8

AIM:- Write a program to implement function decorator to display cube of a number.

CODE:-

```
# Define the decorator
def cube_decorator(func):
    def wrapper(num):
        result = func(num)
        print(f"The cube of {num} is {result}")
        return result
    return wrapper

# Use the decorator on a function
@cube_decorator
def cube(num):
    return num ** 3

# Test the function
if __name__ == "__main__":
    number = int(input("Enter a number: "))
cube(number)
```

Output:-

```
Enter a number: 10
The cube of 10 is 1000
```

Name:- Milind Kailas Tajane
Roll No:- CS061

Date:-_____

Practical No: 9

AIM:- Write a program to implement a package and module:-
Package- Employeemgmt
Module empsalary - function to calculate gross and net salary
Module emphrinfo- function to display employee information i.e. name, designation, dept, qualification ,experience.

CODE:-

```
from Employeemgmt.empsalary import calculate_salary
from Employeemgmt.emphrinfo import display_employee_info

def main():
    print("Welcome to Employee Management System")

    # Input for employee information
    name = input("Enter employee's name: ")
    designation = input("Enter employee's designation: ")
    dept = input("Enter employee's department: ")
    qualification = input("Enter employee's qualification: ")
    experience = int(input("Enter employee's experience (in years): "))

    # Display employee information
    display_employee_info(name, designation, dept, qualification, experience)

    # Input for salary calculation
    basic_salary = float(input("\nEnter employee's basic salary: "))
    hra_percentage = float(input("Enter HRA percentage (default is 20): ") or 20)
    tax_percentage = float(input("Enter tax percentage (default is 10): ") or 10)

    # Calculate and display salary details
    gross_salary, net_salary = calculate_salary(basic_salary, hra_percentage, tax_percentage)
    print("\n--- Salary Details ---")
    print(f"Gross Salary: {gross_salary:.2f}")
    print(f"Net Salary: {net_salary:.2f}")

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

Output:-

```
Welcome to Employee Management System
Enter employee's name: Milind Tajane
Enter employee's designation: Fullstack Web Developer
Enter employee's department: CS
Enter employee's qualification: MCA
Enter employee's experience (in years): 5

--- Employee Information ---
Name: Milind Tajane
Designation: Fullstack Web Developer
Department: CS
Qualification: MCA
Experience: 5 years

Enter employee's basic salary: 50000
Enter HRA percentage (default is 20):
Enter tax percentage (default is 10):

--- Salary Details ---
Gross Salary: 60000.00
Net Salary: 54000.00
```

Name:- Milind Kailas Tajane
Roll No:- CS061

Date:-_____

Practical No: 10

AIM:- Write a program to implement a class to store student information as id, name,marks.
Implement all class, instance, public, private attributes.
Implement instance, class, constructor, destructor, getter and setter methods.

CODE:-

```
class Student:
    # Class Attribute
    total_students = 0

    # Constructor
    def __init__(self, student_id, name, marks):
        # Public Instance Attributes
        self.student_id = student_id
        self.name = name

        # Private Instance Attribute
        self.__marks = marks

        # Increment class attribute
        Student.total_students += 1
        print(f"Student {self.name} added successfully!")

    # Destructor
    def __del__(self):
        # Decrement class attribute
        Student.total_students -= 1
        print(f"Student {self.name} removed from records.")

    # Instance Method to Display Student Info
    def display_info(self):
        print("\n--- Student Information ---")
        print(f"ID: {self.student_id}")
        print(f"Name: {self.name}")
        print(f"Marks: {self.__marks}")

    # Getter for Marks (Private Attribute)
    def get_marks(self):
        return self.__marks
```

```
    # Setter for Marks (Private Attribute)
    def set_marks(self, marks):
        if 0 <= marks <= 100:  # Validating marks range
            self.__marks = marks
            print(f"Marks updated for {self.name}.")
        else:
            print("Invalid marks! Must be between 0 and 100.")

    # Class Method to Display Total Students
    @classmethod
    def display_total_students(cls):
        print(f"\nTotal Students: {cls.total_students}")

    # Static Method Example: Validating Marks
    @staticmethod
    def is_valid_marks(marks):
        return 0 <= marks <= 100

# Main Program
def main():
    # Creating Student Instances
    student1 = Student(1, "Milind Tajane", 85)
    student2 = Student(2, "Gayatri Jadhav", 92)

    # Display Student Information
    student1.display_info()
    student2.display_info()

    # Using Getter and Setter for Marks
    print(f"\nMarks for {student1.name}: {student1.get_marks()}")
    student1.set_marks(95)
    print(f"Updated Marks for {student1.name}: {student1.get_marks()}")

    # Demonstrate Class Method
    Student.display_total_students()

    # Demonstrate Static Method
    print(f"Are 105 marks valid? {Student.is_valid_marks(105)}")
    print(f"Are 95 marks valid? {Student.is_valid_marks(95)}")

    # Delete a Student Instance
    del student1

    # Display Total Students After Deletion
    Student.display_total_students()

# Run the Program
if __name__ == "__main__":
    main()
```

Output:-

```
Student Milind Tajane added successfully!
Student Gayatri Jadhav added successfully!

--- Student Information ---
ID: 1
Name: Milind Tajane
Marks: 85

--- Student Information ---
ID: 2
Name: Gayatri Jadhav
Marks: 92

Marks for Milind Tajane: 85
Marks updated for Milind Tajane.
Updated Marks for Milind Tajane: 95

Total Students: 2
Are 105 marks valid? False
Are 95 marks valid? True
Student Milind Tajane removed from records.

Total Students: 1
Student Gayatri Jadhav removed from records.
```

Name:- Milind Kailas Tajane

Roll No:- CS061

Date:-_____

Practical No: 11

AIM:- Write a program to validate email id, password, url and mobile using regular expression.

CODE:-

```
import re

# Function to validate an email ID
def validate_email(email):
    pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
    return re.match(pattern, email) is not None

# Function to validate a password
def validate_password(password):
    """
    Password criteria:
    - At least 8 characters long
    - At least one uppercase letter
    - At least one lowercase letter
    - At least one digit
    - At least one special character
    """
    pattern = r'^(?=.*[A-Z])(?=.*[a-z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{8,}$'
    return re.match(pattern, password) is not None

# Function to validate a URL
def validate_url(url):
    pattern = r'^(https?:\/\/)?(www\.)?[a-zA-Z0-9-]+\.[a-zA-Z]{2,}(\/\S*)?$'
    return re.match(pattern, url) is not None

# Function to validate a mobile number
def validate_mobile(mobile):
    """
    Mobile number criteria:
    - 10 digits
    - Starts with 7, 8, or 9
    """
    pattern = r'^[789]\d{9}$'
    return re.match(pattern, mobile) is not None

# Main Program
```

```
def main():
    # Validate email
    email = input("Enter email ID: ")
    if validate_email(email):
        print("Valid Email ID")
    else:
        print("Invalid Email ID")

    # Validate password
    password = input("Enter password: ")
    if validate_password(password):
        print("Valid Password")
    else:
        print("Invalid Password. Must contain at least 8 characters, one uppercase, one
lowercase, one digit, and one special character.")

    # Validate URL
    url = input("Enter URL: ")
    if validate_url(url):
        print("Valid URL")
    else:
        print("Invalid URL")

    # Validate mobile number
    mobile = input("Enter mobile number: ")
    if validate_mobile(mobile):
        print("Valid Mobile Number")
    else:
        print("Invalid Mobile Number. Must be 10 digits and start with 7, 8, or 9.")

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

Output:-

```
Enter email ID: john.doe@example.com
Valid Email ID
Enter password: John@1234
Valid Password
Enter URL: https://www.example.com/path
Valid URL
Enter mobile number: 9876543210
Valid Mobile Number
```

Name:- Milind Kailas Tajane

Roll No:- CS061

Date:-_____

Practical No: 12

AIM:- Write a program to implement built in exceptions.

CODE:-

```
def demonstrate_exceptions():
    print("\n--- Demonstrating Built-in Exceptions ---")

    # 1. ZeroDivisionError
    try:
        result = 10 / 0
    except ZeroDivisionError as e:
        print(f"ZeroDivisionError: {e}")

    # 2. ValueError
    try:
        num = int("abc")  # Invalid conversion from string to int
    except ValueError as e:
        print(f"ValueError: {e}")

    # 3. IndexError
    try:
        lst = [1, 2, 3]
        print(lst[5])  # Index out of range
    except IndexError as e:
        print(f"IndexError: {e}")

    # 4. KeyError
    try:
        dictionary = {"a": 1, "b": 2}
        print(dictionary["c"])  # Accessing a non-existent key
    except KeyError as e:
        print(f"KeyError: {e}")

    # 5. FileNotFoundError
    try:
        with open("non_existent_file.txt", "r") as f:
            content = f.read()
    except FileNotFoundError as e:
        print(f"FileNotFoundError: {e}")

    # 6. TypeError
```

```
    try:
        result = "10" + 5  # Adding string and integer
    except TypeError as e:
        print(f"TypeError: {e}")

    # 7. AttributeError
    try:
        num = 10
        num.append(5)  # `int` has no `append` method
    except AttributeError as e:
        print(f"AttributeError: {e}")

    # 8. ImportError
    try:
        from math import non_existent_function  # Non-existent import
    except ImportError as e:
        print(f"ImportError: {e}")

    # 9. NameError
    try:
        print(undefined_variable)  # Variable not defined
    except NameError as e:
        print(f"NameError: {e}")

    # 10. OverflowError
    try:
        import math
        print(math.exp(1000))  # Exceeds the limit of floating-point numbers
    except OverflowError as e:
        print(f"OverflowError: {e}")

    print("\n--- End of Demonstration ---")

# Main program
if __name__ == "__main__":
    demonstrate_exceptions()
```

Output:-

```
--- Demonstrating Built-in Exceptions ---
ZeroDivisionError: division by zero
ValueError: invalid literal for int() with base 10: 'abc'
IndexError: list index out of range
KeyError: 'c'
FileNotFoundError: [Errno 2] No such file or directory: 'non_existent_file.txt'
TypeError: can only concatenate str (not "int") to str
AttributeError: 'int' object has no attribute 'append'
ImportError: cannot import name 'non_existent_function' from 'math' (unknown location)
NameError: name 'undefined_variable' is not defined
OverflowError: math range error

--- End of Demonstration ---
```

Name:- Milind Kailas Tajane

Roll No:- CS061

Date:-_____

Practical No: 13

AIM:- Write a program to implement user defined exception to display message if account balance is below 1000 while withdrawing amount.

CODE:-

```
# User-defined exception class
class InsufficientBalanceError(Exception):
    def __init__(self, message="Balance is below the minimum required amount of 1000."):
        self.message = message
        super().__init__(self.message)

# BankAccount class
class BankAccount:
    def __init__(self, account_number, balance):
        self.account_number = account_number
        self.balance = balance

    def withdraw(self, amount):
        try:
            if self.balance - amount < 1000:
                raise InsufficientBalanceError(
                    f"Insufficient funds! Cannot withdraw {amount}. Your current balance:
{self.balance}"
                )
            self.balance -= amount
            print(f"Withdrawal successful! New balance: {self.balance}")
        except InsufficientBalanceError as e:
            print(f"Error: {e}")

    def display_balance(self):
        print(f"Account Number: {self.account_number}, Balance: {self.balance}")

# Main program
def main():
    # Create a bank account with an initial balance
    account = BankAccount("123456789", 2000)

    # Display current balance
    account.display_balance()

    # Attempt a withdrawal
```

```
    withdrawal_amount = int(input("Enter amount to withdraw: "))
    account.withdraw(withdrawal_amount)

    # Display balance after withdrawal
    account.display_balance()

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

Output:-

```
Account Number: 123456789, Balance: 2000
Enter amount to withdraw: 500
Withdrawal successful! New balance: 1500
Account Number: 123456789, Balance: 1500
```

```
Account Number: 123456789, Balance: 2000
Enter amount to withdraw: 1200
Error: Insufficient funds! Cannot withdraw 1200. Your current balance: 2000
Account Number: 123456789, Balance: 2000
```

Name:- Milind Kailas Tajane

Roll No:- CS061

Date:-_____

Practical No: 14

AIM:- Write a program to implement multithreading.

CODE:-

```
import threading
import time

# Function to print numbers
def print_numbers():
    for i in range(1, 6):
        print(f"Number: {i}")
        time.sleep(1)  # Simulate a delay

# Function to print alphabets
def print_alphabets():
    for char in 'ABCDE':
        print(f"Alphabet: {char}")
        time.sleep(1)  # Simulate a delay

# Function to print a message repeatedly
def print_message():
    for i in range(1, 6):
        print(f"Message: Hello from thread {i}")
        time.sleep(1)  # Simulate a delay

# Main program
def main():
    # Create threads for different tasks
    thread1 = threading.Thread(target=print_numbers)
    thread2 = threading.Thread(target=print_alphabets)
    thread3 = threading.Thread(target=print_message)

    # Start the threads
    thread1.start()
    thread2.start()
    thread3.start()

    # Wait for threads to finish
    thread1.join()
    thread2.join()
    thread3.join()
```

```
    print("All threads have completed execution.")

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

Output:-

```
Number: 1Alphabet: AMessage: Hello from thread 1


Number: 2Alphabet: B
Message: Hello from thread 2

Number: 3Alphabet: CMessage: Hello from thread 3


Number: 4Alphabet: DMessage: Hello from thread 4


Number: 5Alphabet: E

Message: Hello from thread 5
All threads have completed execution.
```

Name:- Milind Kailas Tajane

Roll No:- CS061

Date:-_____

Practical No: 15

AIM:- Write a menu driven program to perform following functions on product table product table(pid,pname,price) . Use mongodb database

1.Insert a record
2.Update price
3.Delete by pid
4.Dispaly all
5. Exit.

CODE:-

```
from pymongo import MongoClient

# Establish connection to MongoDB
client = MongoClient("mongodb://localhost:27017/")  # Replace with your MongoDB
connection string
db = client["productdb"]  # Database name
product_collection = db["product"]  # Collection name

# Function to insert a product record
def insert_product():
    pid = input("Enter product ID: ")
    pname = input("Enter product name: ")
    price = float(input("Enter product price: "))
    product = {"pid": pid, "pname": pname, "price": price}
    product_collection.insert_one(product)
    print("Product inserted successfully.")

# Function to update product price
def update_price():
    pid = input("Enter product ID to update: ")
    new_price = float(input("Enter new price: "))
    result = product_collection.update_one({"pid": pid}, {"$set": {"price": new_price}})
    if result.matched_count > 0:
        print("Price updated successfully.")
    else:
        print("Product ID not found.")

# Function to delete a product by ID
```

```
def delete_product():
    pid = input("Enter product ID to delete: ")
    result = product_collection.delete_one({"pid": pid})
    if result.deleted_count > 0:
        print("Product deleted successfully.")
    else:
        print("Product ID not found.")

# Function to display all products
def display_all():
    products = product_collection.find()
    print("\n--- Product List ---")
    for product in products:
        print(f"ID: {product['pid']}, Name: {product['pname']}, Price: {product['price']}")
    print("----------------------")

# Menu-driven program
def main():
    while True:
        print("\n--- Product Management ---")
        print("1. Insert a record")
        print("2. Update price")
        print("3. Delete by PID")
        print("4. Display all products")
        print("5. Exit")

        choice = input("Enter your choice: ")

        if choice == "1":
            insert_product()
        elif choice == "2":
            update_price()
        elif choice == "3":
            delete_product()
        elif choice == "4":
            display_all()
        elif choice == "5":
            print("Exiting the program.")
            break
        else:
            print("Invalid choice. Please try again.")

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

Output:-

```
--- Product Management ---
1. Insert a record
2. Update price
3. Delete by PID
4. Display all products
5. Exit
Enter your choice: 2
Enter product ID to update: 101
Enter new price: 72000
Price updated successfully.

--- Product Management ---
1. Insert a record
2. Update price
3. Delete by PID
4. Display all products
5. Exit
Enter your choice: 3
Enter product ID to delete: 101
Product deleted successfully.

--- Product Management ---
1. Insert a record
2. Update price
3. Delete by PID
4. Display all products
5. Exit
Enter your choice: 4

--- Product List ---
ID: 101, Name: Laptop, Price: 75000.0
ID: 101, Name: Laptop, Price: 75000.0
ID: 102, Name: Phone, Price: 25000.0
ID: 101, Name: Laptop, Price: 75000.0
ID: 102, Name: Phone, Price: 25000.0
ID: 103, Name: Tablet, Price: 15000.0
```