**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)

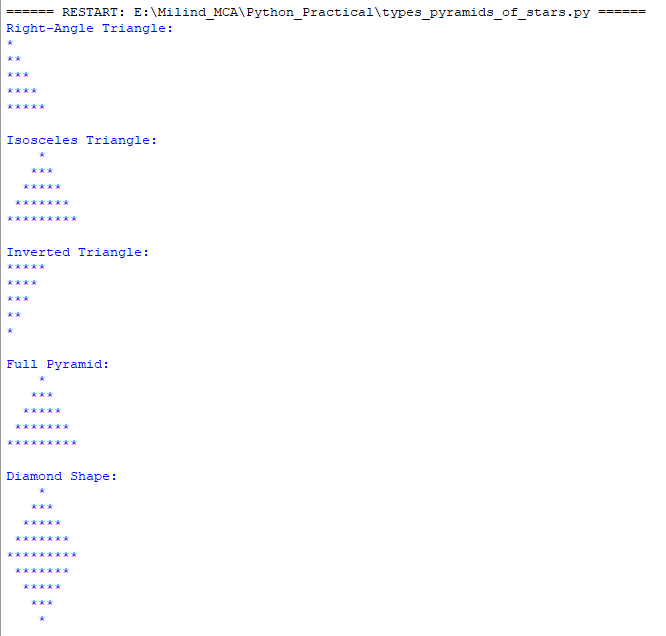
print\_isosceles\_triangle(n)

print\_inverted\_triangle(n)

print\_full\_pyramid(n)

print\_diamond(n)

**Output:-**



**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))

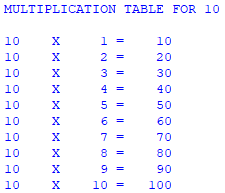
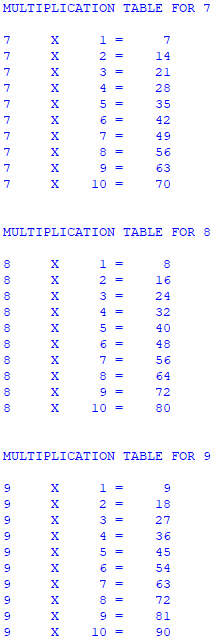
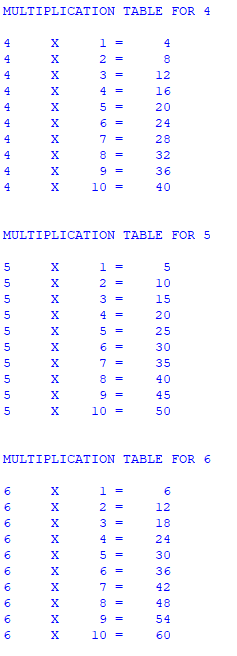
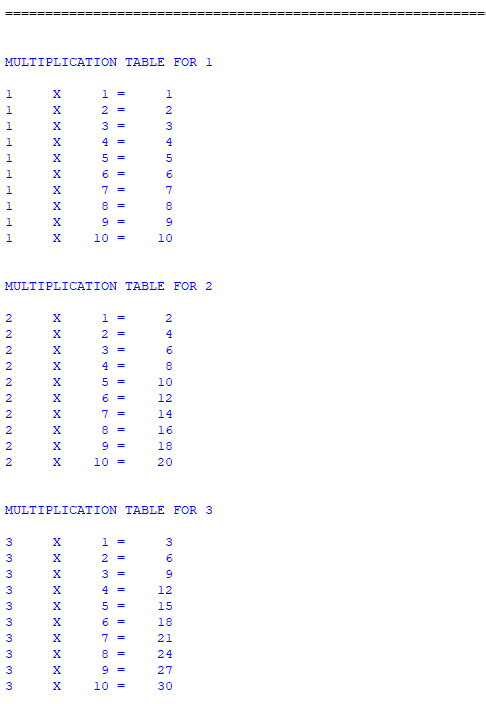
# Loop through numbers 1 to 10 for the multipliers

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:-**



**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.

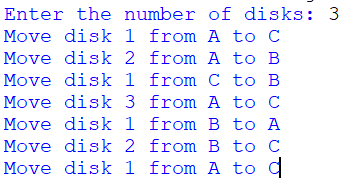
tower\_of\_hanoi(n - 1, auxiliary, destination, source)

# Example usage

num\_disks = int(input("Enter the number of disks: "))

tower\_of\_hanoi(num\_disks, "A", "C", "B")

**Output:-**



**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): "))

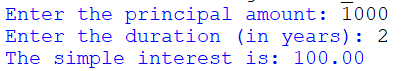
# Calculate simple interest using the user inputs and default rate

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:-**



**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)

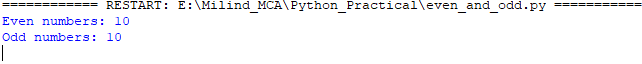
# Print the count of even numbers

print(f"Even numbers: {even\_count}")

# Print the count of odd numbers

print(f"Odd numbers: {odd\_count}")

**Output:-**



**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}")

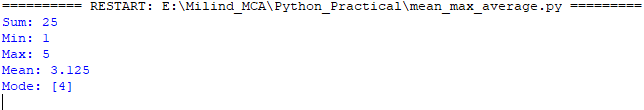
print(f"Min: {minimum}")

print(f"Max: {maximum}")

print(f"Mean: {mean}")

print(f"Mode: {mode}")

**Output:-**



**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*

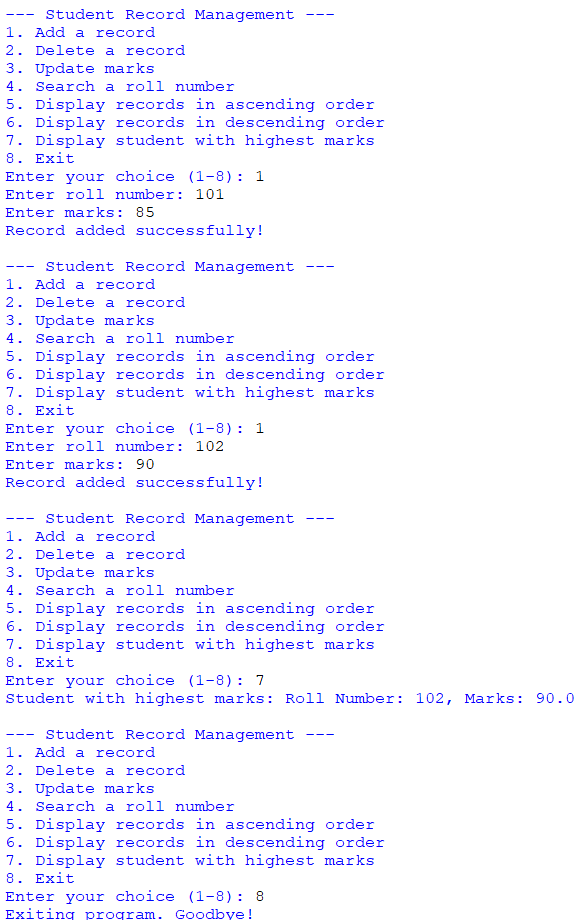
*else*:

            print("Invalid choice! Please try again.")

*# Run the program*

menu()

**Output:-**



**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:-**



**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 ---")

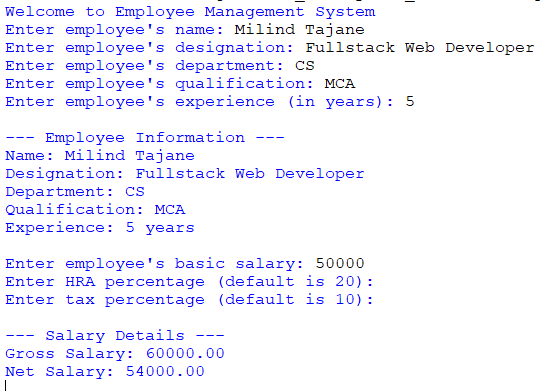
print(f"Gross Salary: {gross\_salary:.2f}")

print(f"Net Salary: {net\_salary:.2f}")

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

main()

**Output:-**

****

**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*

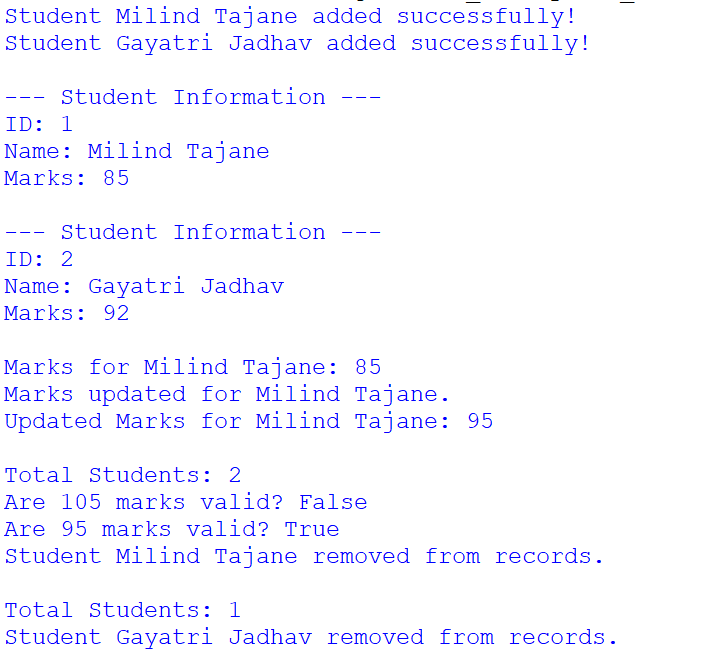
    Student.display\_total\_students()

*# Run the Program*

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

    main()

**Output:-**



**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")

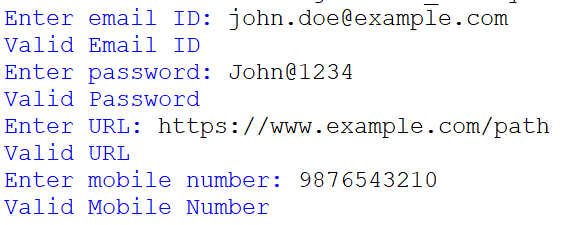
*else*:

        print("Invalid Mobile Number. Must be 10 digits and start with 7, 8, or 9.")

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

    main()

**Output:-**



**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}")

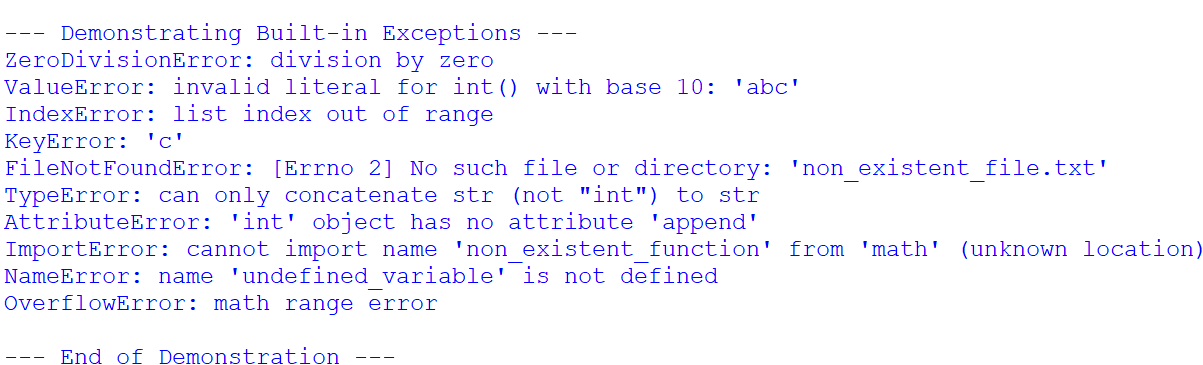
print("\n--- End of Demonstration ---")

# Main program

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

demonstrate\_exceptions()

**Output:-**



**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)

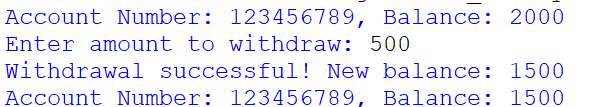
    # Display balance after withdrawal

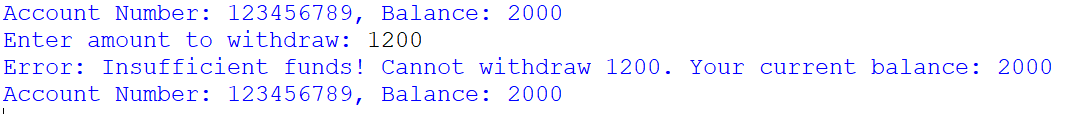
    account.display\_balance()

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

    main()

**Output:-**





**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()

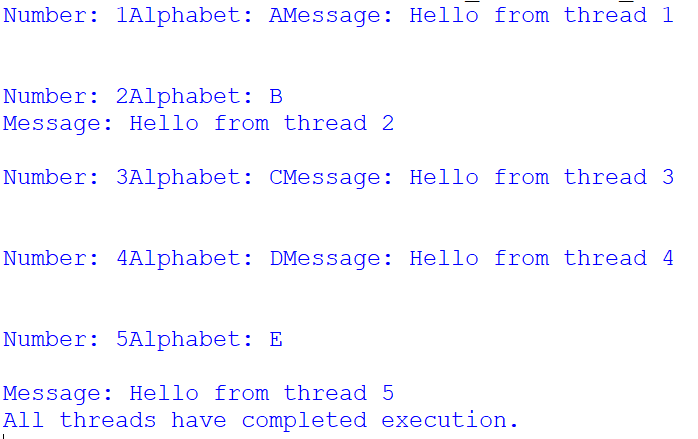
thread3.join()

print("All threads have completed execution.")

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

main()

**Output:-**



**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:-**

*# 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:-**

