1.Write a program to check if the input year is leap year or not. Validate the input

```
def is_leap_year(year):
    # Leap year conditions
    if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
        return True
    else:
        return False

def main():
    # Input validation
    while True:
        try:
            year = int(input("Enter a year: "))
            if year > 0:
                break
            else:
                print("Please enter a positive year.")
        except ValueError:
            print("Invalid input. Please enter a valid integer.")

    if is_leap_year(year):
        print(f"{year} is a leap year.")
    else:
        print(f"{year} is not a leap year.")

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

2.Write a program to find the factorial of a number using recursion

```
def factorial(n):
    # Base case: factorial of 0 or 1 is 1
    if n == 0 or n == 1:
        return 1
    else:
        # Recursive case: n! = n * (n-1)!
        return n * factorial(n - 1)
```

```
def main():
    # Input validation
    while True:
        try:
            num = int(input("Enter a non-negative integer: "))
            if num >= 0:
                break
            else:
                print("Please enter a non-negative integer.")
        except ValueError:
            print("Invalid input. Please enter a valid integer.")

    result = factorial(num)
    print(f"The factorial of {num} is: {result}")
if __name__ == "__main__":
    main()
```

3.Write a program to print the sum of natural numbers using recursion.

```
def sum_of_natural_numbers(n):
    # Base case: sum of first natural number is 1
    if n == 1:
        return 1
    else:
        # Recursive case: sum(n) = n + sum(n-1)
        return n + sum_of_natural_numbers(n - 1)
def main():
    # Input validation
    while True:
        try:
            num = int(input("Enter a positive integer: "))
            if num > 0:
                break
            else:
                print("Please enter a positive integer.")
        except ValueError:
            print("Invalid input. Please enter a valid integer.")
    result = sum_of_natural_numbers(num)
    print(f"The sum of natural numbers up to {num} is: {result}")
if __name__ == "__main__":
    main()
```

4.Write a program to reverse each word of “data.txt” file

```
def reverse_words_in_file(file_path):
    try:
        with open(file_path, 'r') as file:
            # Read the content of the file
            content = file.read()

            # Reverse each word in the content
            reversed_content = ' '.join(word[::-1] for word in content.split())

        with open(file_path, 'w') as file:
            # Write the reversed content back to the file
            file.write(reversed_content)

        print(f"Successfully reversed each word in {file_path}")
    except FileNotFoundError:
        print(f"Error: File {file_path} not found.")
    except Exception as e:
        print(f"An error occurred: {e}")

def main():
    file_path = "data.txt"  # Replace with the path to your file

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

5.Write a python program using MongoDB database to create a “Books” collection having fields: title, Author (a list), Publisher, PubAddress, (a dict with keys like area, city, country), Price, ISBN. Accept input from user to insert documents.

```
import pymongo


def get_book_details():
    title = input("Enter the book title: ")
    authors = [author.strip() for author in input("Enter authors (comma-separated): ").split(',')]
    publisher = input("Enter the publisher: ")
    pub_address = {
        'area': input("Enter the publication area: "),
        'city': input("Enter the publication city: "),
        'country': input("Enter the publication country: ")
    }
    price = float(input("Enter the book price: "))
    isbn = input("Enter the ISBN: ")

    book = {
        'title': title,
        'authors': authors,
        'publisher': publisher,
        'pubAddress': pub_address,
        'price': price,
        'ISBN': isbn
    }

    return book


def insert_book(collection, book):
    result = collection.insert_one(book)
    print(f"Book inserted with ObjectId: {result.inserted_id}")


def main():
    try:
        # Connect to MongoDB
        client = pymongo.MongoClient("mongodb://localhost:27017/")
        database_name = "library"
        db = client[database_name]

        # Create or get the "Books" collection
        collection_name = "Books"
        books_collection = db[collection_name]

        # Accept input from user to insert documents
        while True:
            book = get_book_details()
            insert_book(books_collection, book)

            another_entry = input("Do you want to enter another book? (yes/no): ").lower()
            if another_entry != 'yes':
                break

    except Exception as e:
        print(f"An error occurred: {e}")
    finally:
        # Close the MongoDB connection
        client.close()

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

6.Write a mongoDB program to update the “Books” collection in que 5

```
import pymongo

def update_book(collection, title_to_update, new_price):
    # Update the document with the specified title
    result = collection.update_one(
        {'title': title_to_update},
        {'$set': {'price': new_price}}
    )

    if result.modified_count > 0:
        print(f"Successfully updated the price for book '{title_to_update}'")
    else:
        print(f"No matching document found for book '{title_to_update}'")

def main():
    try:
        # Connect to MongoDB
        client = pymongo.MongoClient("mongodb://localhost:27017/")
        database_name = "library"
        db = client[database_name]

        # Get the "Books" collection
        collection_name = "Books"
        books_collection = db[collection_name]

        # Display the current contents of the collection
        print("Current contents of the 'Books' collection:")
        for book in books_collection.find():
            print(book)

        # Accept input from the user for updating a book
        title_to_update = input("Enter the title of the book to update: ")
        new_price = float(input("Enter the new price for the book: "))

        # Update the specified book in the collection
        update_book(books_collection, title_to_update, new_price)
        # Display the updated contents of the collection
        print("Updated contents of the 'Books' collection:")
        for book in books_collection.find():
            print(book)

    except Exception as e:
        print(f"An error occurred: {e}")
    finally:
        # Close the MongoDB connection
        client.close()

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

7. Write a program to accept decimal number and print its octal and hexadecimal equivalent.

```
def convert_decimal_to_octal_and_hex(decimal_number):
    octal_equivalent = oct(decimal_number)
    hexadecimal_equivalent = hex(decimal_number)

    return octal_equivalent, hexadecimal_equivalent

def main():
    try:
        decimal_number = int(input("Enter a decimal number: "))

        octal_equivalent, hexadecimal_equivalent =
convert_decimal_to_octal_and_hex(decimal_number)

        print(f"Octal equivalent: {octal_equivalent}")
        print(f"Hexadecimal equivalent: {hexadecimal_equivalent}")

    except ValueError:
        print("Invalid input. Please enter a valid decimal number.")


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

8. Write a program to read the contents of file and display occurrance of given character.

```
def count_occurrences(file_path, char_to_count):
    try:
        with open(file_path, 'r') as file:
            content = file.read()

            # Count occurrences of the specified character
            occurrences = content.count(char_to_count)

        print(f"The character '{char_to_count}' occurs {occurrences} times in the file.")

    except FileNotFoundError:
        print(f"Error: File {file_path} not found.")
    except Exception as e:
        print(f"An error occurred: {e}")

def main():
    file_path = "example.txt"  # Replace with the path to your file
    char_to_count = input("Enter a character to count its occurrences: ")

    try:
        # Validate input to make sure it's a single character
        if len(char_to_count) == 1:
            count_occurrences(file_path, char_to_count)
        else:
            print("Please enter a single character.")
    except ValueError:
        print("Invalid input. Please enter a valid character.")


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

9. Write a python program using mongoDB database to create a “student” collection having fields: Student-ID, Name, Course, Mobile, Address. (a dict with keys like area, city, country, pin) Accept input from user to insert documents

```
import pymongo


def get_student_details():
    student_id = int(input("Enter Student ID: "))
    name = input("Enter student name: ")
    course = input("Enter course: ")
    mobile = input("Enter mobile number: ")
    address = {
        'area': input("Enter area: "),
        'city': input("Enter city: "),
        'country': input("Enter country: "),
        'pin': input("Enter PIN code: ")
    }

    student = {
        'Student-ID': student_id,
        'Name': name,
        'Course': course,
        'Mobile': mobile,
        'Address': address
    }
    return student


def insert_student(collection, student):
    result = collection.insert_one(student)
    print(f"Student information inserted with ObjectId: {result.inserted_id}")


def main():
    try:
        # Connect to MongoDB
        client = pymongo.MongoClient("mongodb://localhost:27017/")
        database_name = "school"
        db = client[database_name]

        # Create or get the "student" collection
        collection_name = "students"
        student_collection = db[collection_name]

        # Accept input from user to insert documents
        while True:
            student = get_student_details()
            insert_student(student_collection, student)

            another_entry = input("Do you want to enter another student? (yes/no): ").lower()
            if another_entry != 'yes':
                break

    except Exception as e:
        print(f"An error occurred: {e}")
    finally:
        # Close the MongoDB connection
        client.close()
if __name__ == "__main__":
    main()
```

10. Write a MongoDB program to delete selected documents given in que 9.

```
import pymongo


def display_students(collection):
    print("Current contents of the 'students' collection:")
    for student in collection.find():
        print(student)


def delete_student_by_id(collection, student_id):
    result = collection.delete_one({'Student-ID': student_id})
    if result.deleted_count > 0:
        print(f"Successfully deleted the student with Student ID {student_id}")
    else:
        print(f"No matching document found for Student ID {student_id}")


def main():
    try:
        # Connect to MongoDB
        client = pymongo.MongoClient("mongodb://localhost:27017/")
        database_name = "school"
        db = client[database_name]

        # Get the "students" collection
        collection_name = "students"
        student_collection = db[collection_name]

        # Display the current contents of the collection
        display_students(student_collection)

        # Accept input from the user for deleting a student
        student_id_to_delete = int(input("Enter the Student ID to delete: "))

        # Delete the specified student from the collection
        delete_student_by_id(student_collection, student_id_to_delete)


        # Display the updated contents of the collection
        display_students(student_collection)


    except Exception as e:
        print(f"An error occurred: {e}")
    finally:
        # Close the MongoDB connection
        client.close()


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

11. Write a program to validate email address using regular expression. Also explain the meaning of each and energy special character of the regular expression used by you in this program.

```
import re


def validate_email(email):
    # Regular expression for validating an Email
    email_pattern = r'^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$'


    # Using re.match() to search the pattern at the beginning of the string
    match = re.match(email_pattern, email)


    # Check if the email matches the pattern
    if match:
        print(f"{email} is a valid email address.")
    else:
        print(f"{email} is not a valid email address.")


def main():
    email_address = input("Enter an email address to validate: ")
    validate_email(email_address)


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

12. Write user defined exception program in python which will except age as an input from the user and check whether the user is eligible for voting or not. If age

```
class NotEligibleForVotingException(Exception):
    def __init__(self, age):
        self.age = age
        super().__init__(f"Age {age} is not eligible for voting. Minimum age required: 18")


def check_voting_eligibility(age):
    if age < 18:
        raise NotEligibleForVotingException(age)
    else:
        print(f"Age {age} is eligible for voting.")


def main():
    try:
        age = int(input("Enter your age: "))
        check_voting_eligibility(age)
    except ValueError:
        print("Invalid input. Please enter a valid age.")
    except NotEligibleForVotingException as e:
        print(e)


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

13. Write a program to validate URL using regular expression. Also explain the meaning of each and every special character of the regular expression used by you in this program.

```
import re


def validate_url(url):

    # Regular expression for validating a URL

    url_pattern = r'^(https?|ftp)://[^\s/$.?#].[^\s]*$'


    # Using re.match() to search the pattern at the beginning of the string

    match = re.match(url_pattern, url)


    # Check if the URL matches the pattern

    if match:

        print(f"{url} is a valid URL.")

    else:

        print(f"{url} is not a valid URL.")


def main():

    url = input("Enter a URL to validate: ")

    validate_url(url)


if __name__ == "__main__":

    main()
```

14. Write multithread program, where one thread prints square of a number and another thread prints cube of numbers. Also display the total time taken for execution.

```
import threading

import time


def print_squares(numbers):

    for num in numbers:

        time.sleep(0.1)  # Simulating some computation time
        print(f"Square: {num * num}")

def print_cubes(numbers):
    for num in numbers:
        time.sleep(0.1)  # Simulating some computation time
        print(f"Cube: {num * num * num}")

def main():
    numbers = [1, 2, 3, 4, 5]

    start_time = time.time()

    # Creating two threads
    square_thread = threading.Thread(target=print_squares, args=(numbers,))
    cube_thread = threading.Thread(target=print_cubes, args=(numbers,))

    # Starting both threads
    square_thread.start()
    cube_thread.start()

    # Waiting for both threads to finish
    square_thread.join()
    cube_thread.join()

    end_time = time.time()

    total_time = end_time - start_time
    print(f"Total time taken for execution: {total_time} seconds")

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

15. Create a 5×5 2D numPy array and retrieve bottom right corner 2×2 array from it.

```
import numpy as np


# Create a 5x5 2D NumPy array
original_array = np.array([[1, 2, 3, 4, 5],
                           [6, 7, 8, 9, 10],
                           [11, 12, 13, 14, 15],
                           [16, 17, 18, 19, 20],
                           [21, 22, 23, 24, 25]])


# Retrieve the bottom right corner 2x2 array
bottom_right_corner = original_array[-2:, -2:]


print("Original Array:")
print(original_array)
print("\nBottom Right Corner 2x2 Array:")
print(bottom_right_corner)
```

16. Given a dataFrame df in Pandas as below: [4] City MaxTemp MinTemp RainFall Delhi 40 32 24.1 Bengaluru 31 25 36.2 Chennai 35 27 40.8 Mumbai 29 21 35.2 Kolkata 39 23 41.8 Write commands: i) to compute sum of every column of the dateFrame. ii) to compute mean of column rainFall. iii) to compute Median of the MaxTemp column. iv) to display all column names.

import pandas as pd

```
# Creating the DataFrame
data = {'City': ['Delhi', 'Bengaluru', 'Chennai', 'Mumbai', 'Kolkata'],
    'MaxTemp': [40, 31, 35, 29, 39],
    'MinTemp': [32, 25, 27, 21, 23],
    'RainFall': [24.1, 36.2, 40.8, 35.2, 41.8]}


df = pd.DataFrame(data)
```

```
# i) Compute sum of every column
column_sums = df.sum()
print("Sum of every column:")
print(column_sums)

# ii) Compute mean of column RainFall
rainfall_mean = df['RainFall'].mean()
print("\nMean of column RainFall:", rainfall_mean)

# iii) Compute Median of the MaxTemp column
maxtemp_median = df['MaxTemp'].median()
print("\nMedian of column MaxTemp:", maxtemp_median)

# iv) Display all column names
column_names = df.columns
print("\nColumn names:")
print(column_names)
```

17. Create a 4×3 numPy array and find it's column-wise mean

```
import numpy as np

# Create a 4x3 NumPy array
array_4x3 = np.array([[1, 2, 3],
                      [4, 5, 6],
                      [7, 8, 9],
                      [10, 11, 12]])

# Find column-wise mean
column_means = np.mean(array_4x3, axis=0)

print("4x3 Array:")
print(array_4x3)
print("\nColumn-wise Mean:")
print(column_means)
```

18.Create a series from a numPy array and find frequency count of unique items of a series.

```
import numpy as np


ini_array = np.array([10, 20, 5,
               10, 8, 20,
               8, 9])


# Get a tuple of unique values
# and their frequency in
# numpy array
unique, frequency = np.unique(ini_array,
                    return_counts = True)
# print unique values array
print("Unique Values:",
   unique)


# print frequency array
print("Frequency Values:",
   frequency)
```

19.Create a pandas dataFrame using CSV file and perform a following: i) Display first 10 rows. ii) Display list of all columns.

```
import pandas as pd
df = pd.read_csv('movies_metadata.csv')
#Display the first 10 rows
result = df.head(10)
print("First 10 rows of the DataFrame:")
print(result)
pd.set_option('display.max_columns', None)
movies.head()
```

20. Write a program to check the number entered by user is even or odd. Program should accept integer digits only

```
num = int(input("Enter a Number:"))

if num % 2 == 0:

  print("Given number is Even")

else:

  print("Given number is Odd")
```

21. Write a python program for the following. [3] i) Create list of fruits ii) Add new fruit in list. iii) sort the list. iv) delete last fruit name from list

```
# i) Create list of fruits

fruits = ['Apple', 'Banana', 'Orange', 'Grapes', 'Mango']


# ii) Add new fruit in list

new_fruit = input("Enter a new fruit to add to the list: ")

fruits.append(new_fruit)


# iii) Sort the list

fruits.sort()


# iv) Delete last fruit name from list

if len(fruits) > 0:

    deleted_fruit = fruits.pop()

    print(f"Deleted last fruit: {deleted_fruit}")

else:

    print("The list is empty after deleting the last fruit.")


# Display the updated list

print("Updated list of fruits:", fruits)
```

22. Write a python function to check the given number is even or odd. Handle suitable exceptions.

```
def check_even_odd(number):

    try:
        # Convert the input to an integer
        num = int(number)

        # Check if the number is even or odd
        if num % 2 == 0:
            return f"{num} is an even number."
        else:
            return f"{num} is an odd number."

    except ValueError:
        return "Invalid input. Please enter an integer."

# Example usage:
user_input = input("Enter a number: ")

result = check_even_odd(user_input)
print(result)
```

23. Write a python program to create an employee. txt file and store employee name and address.

```
def create_employee_file():
    try:
        # Open the file in write mode
        with open('employee.txt', 'w') as file:
            # Accept input from the user for employee details
            while True:
                employee_name = input("Enter employee name (or type 'exit' to stop): ")
                if employee_name.lower() == 'exit':
                    break

                employee_address = input("Enter employee address: ")

                # Write employee details to the file
            file.write(f"{employee_name}, {employee_address}\n")


        print("Employee file created successfully.")


    except Exception as e:
        print(f"An error occurred: {e}")


if __name__ == "__main__":
    create_employee_file
```

24. Write a python program to create “employee” collection with fields” (ID, name, address, phone email and dept) in mongoDB. Prform the following operations.

i) Display all employees in “Accounts” department

ii) Delete employee with ID - 210345

iii) Update phone with new phone for employee ID -123

```
=import pymongo
def connect_to_mongodb():
    # Connect to MongoDB
    client = pymongo.MongoClient("mongodb://localhost:27017/")


    # Create or get the "company" database
    database_name = "company"
    db = client[database_name]


    # Create or get the "employee" collection
    collection_name = "employees"
    employees_collection = db[collection_name]


    return employees_collection


def display_employees_in_department(employees_collection, department):
    # i) Display all employees in the specified department
    employees_in_department = employees_collection.find({'dept': department})
    print(f"\nEmployees in '{department}' department:")
    for employee in employees_in_department:
        print(employee)

def delete_employee_by_id(employees_collection, employee_id):
    # ii) Delete employee with the specified ID
    result = employees_collection.delete_one({'ID': employee_id})
    if result.deleted_count > 0:
        print(f"\nEmployee with ID {employee_id} deleted successfully.")
    else:
        print(f"\nNo matching employee found with ID {employee_id}.")

def update_phone_by_id(employees_collection, employee_id, new_phone):
    # iii) Update phone with new phone for the specified employee ID
    result = employees_collection.update_one(
        {'ID': employee_id},
        {'$set': {'phone': new_phone}}
    )
    if result.modified_count > 0:
        print(f"\nPhone updated successfully for employee with ID {employee_id}.")
    else:
        print(f"\nNo matching employee found with ID {employee_id}.")

def main():
    try:
        employees_collection = connect_to_mongodb()

        # Assuming you have some employees in the database
        display_employees_in_department(employees_collection, 'Accounts')

        delete_employee_by_id(employees_collection, 210345)

        update_phone_by_id(employees_collection, 123, 'NEW_PHONE_NUMBER')
    except Exception as e:
        print(f"An error occurred: {e}")
    finally:
        # Close the MongoDB connection
        if 'client' in locals():
            client.close()

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

25.Write a program to retrieve and display employee details from “Employee” collection stored in mangoDB database.

```
=import pymongo
def connect_to_mongodb():
    # Connect to MongoDB
    client = pymongo.MongoClient("mongodb://localhost:27017/")

    # Create or get the "company" database
    database_name = "company"
    db = client[database_name]

    # Create or get the "Employee" collection
    collection_name = "employees"
    employees_collection = db[collection_name]

    return employees_collection

def display_employee_details(employees_collection):
    # Retrieve and display all employee details
    all_employees = employees_collection.find()
    print("\nAll Employee Details:")
    for employee in all_employees:
        print(employee)


def main():
    try:
        employees_collection = connect_to_mongodb()


        # Display all employee details
        display_employee_details(employees_collection)


    except Exception as e:
        print(f"An error occurred: {e}")
    finally:
        # Close the MongoDB connection
        if 'client' in locals():
            client.close()


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

26.Write a program to update the employee details stored in “Employee” collection stored in Mangodb database.

```
=import pymongo


def connect_to_mongodb():
    # Connect to MongoDB
    client = pymongo.MongoClient("mongodb://localhost:27017/")


    # Create or get the "company" database
    database_name = "company"
    db = client[database_name]


    # Create or get the "Employee" collection
    collection_name = "employees"
    employees_collection = db[collection_name]

    return employees_collection

def update_employee_details(employees_collection, employee_id, new_phone):
    # Update phone with new phone for the specified employee ID
    result = employees_collection.update_one(
        {'ID': employee_id},
        {'$set': {'phone': new_phone}}
    )
    if result.modified_count > 0:
        print(f"\nPhone updated successfully for employee with ID {employee_id}.")
    else:
        print(f"\nNo matching employee found with ID {employee_id}.")

def main():
    try:
        employees_collection = connect_to_mongodb()
        # Assuming you have some employees in the database
        employee_id = 123  # Replace with the actual employee ID you want to update
        new_phone_number = 'NEW_PHONE_NUMBER'  # Replace with the new phone number

        update_employee_details(employees_collection, employee_id, new_phone_number)
    except Exception as e:
        print(f"An error occurred: {e}")
    finally:
        # Close the MongoDB connection
        if 'client' in locals():
            client.close()

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

27.Write python program to read “employee” . txt” file and display alternate employee record.

```
=def display_alternate_records(file_path):
    try:
        with open(file_path, 'r') as file:
            lines = file.readlines()

            # Display alternate employee records
            for i in range(0, len(lines), 2):
                print(lines[i].strip())  # Use strip to remove newline characters

    except FileNotFoundError:
        print(f"Error: File {file_path} not found.")
    except Exception as e:
        print(f"An error occurred: {e}")

def main():
    file_path = "employee.txt"  # Replace with the actual path to your file

    # Display alternate employee records from the file
    display_alternate_records(file_path)

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

28.Write a program for extracting email address from a given webpage

```
import requests
import re

def extract_emails_from_webpage(url):
    try:
        # Fetch the HTML content of the webpage
        response = requests.get(url)
        response.raise_for_status()  # Raise an error for unsuccessful HTTP status codes
        html_content = response.text

        # Use a regular expression to find email addresses in the HTML content
        email_pattern = r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b'
        emails = re.findall(email_pattern, html_content)

        # Display the extracted email addresses
        print("Extracted Email Addresses:")
        for email in emails:
            print(email)

    except requests.exceptions.RequestException as e:
        print(f"Error fetching the webpage: {e}")
    except Exception as e:
        print(f"An error occurred: {e}")

def main():
    webpage_url = input("Enter the URL of the webpage to extract email addresses: ")
    extract_emails_from_webpage(webpage_url)

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

29.Write a program to validate URL using regular expression. Explain every special character of the regular expression used in this program

```
=import re

def validate_url(url):
    # Regular expression for validating a URL
    url_pattern = r'^(https?|ftp)://[^\s/$.?#].[^\s]*$'

    # Using re.match() to search the pattern at the beginning of the string
    match = re.match(url_pattern, url)

    # Check if the URL matches the pattern
    if match:
        print(f"{url} is a valid URL.")
    else:
        print(f"{url} is not a valid URL.")

def main():
    url = input("Enter a URL to validate: ")
    validate_url(url)

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

30.Write a multithreading program, where one thread prints square of a number and another thread prints factorial of a number. Also display the

total time taken for the execution.

```
=import threading
import time
import math

def calculate_square(number):
    time.sleep(1)  # Simulate some computation time
    result_square = number * number
    print(f"Square of {number}: {result_square}")

def calculate_factorial(number):
    time.sleep(1)  # Simulate some computation time
    result_factorial = math.factorial(number)
    print(f"Factorial of {number}: {result_factorial}")

def main():
    number = int(input("Enter a number: "))

    start_time = time.time()

    # Create two threads
    square_thread = threading.Thread(target=calculate_square, args=(number,))
    factorial_thread = threading.Thread(target=calculate_factorial, args=(number,))

    # Start both threads
    square_thread.start()
    factorial_thread.start()

    # Wait for both threads to finish
    square_thread.join()
    factorial_thread.join()

    end_time = time.time()

    total_time = end_time - start_time
    print(f"Total time taken for execution: {total_time} seconds")

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

31.Create 5×5 2D numpy assay and retrieve top left corner 2×2 array from it.

```
=import numpy as np

# Create a 5x5 2D NumPy array
original_array = np.array([[1, 2, 3, 4, 5],
                    [6, 7, 8, 9, 10],
                    [11, 12, 13, 14, 15],
                    [16, 17, 18, 19, 20],
                    [21, 22, 23, 24, 25]])

# Retrieve the top-left corner 2x2 array
top_left_corner = original_array[:2, :2]

print("Original Array:")
print(original_array)

print("\nTop Left Corner 2x2 Array:")
print(top_left_corner)
```

32.Write a program to illustrate slicing in numpy array.

```
=import numpy as np

# Create a 3x3 NumPy array
original_array = np.array([[1, 2, 3],
                    [4, 5, 6],
                    [7, 8, 9]])

print("Original Array:")
print(original_array)

# Slicing examples
row_slice = original_array[1, :]
column_slice = original_array[:, 2]
subarray_slice = original_array[:2, 1:]
print("\nSliced Rows:")
print(row_slice)

print("\nSliced Column:")
print(column_slice)


print("\nSliced Subarray:")
print(subarray_slice)
```

33.Create paundas dataframe using two dimensional list. Perform following operations.
i) Count number of rows.
ii) Count missing values in first column.
iii) Display number of columns in data frame.

```
=import pandas as pd


# Create a two-dimensional list
data = [
    ['Alice', 25, 'Engineer'],
    ['Bob', 30, 'Manager'],
    ['Charlie', 22, 'Data Scientist'],
    ['David', 28, 'Analyst'],
    ['Eve', None, 'Designer'],
]


# Create a DataFrame
df = pd.DataFrame(data, columns=['Name', 'Age', 'Occupation'])


# i) Count number of rows
num_rows = len(df)
print(f"Number of Rows: {num_rows}")


# ii) Count missing values in the first column
missing_values_first_column = df['Name'].isnull().sum()
print(f"Missing Values in the First Column: {missing_values_first_column}")
```

```
# iii) Display number of columns in the DataFrame
num_columns = len(df.columns)
print(f"Number of Columns: {num_columns}")

# Display the DataFrame
print("\nDataFrame:")
print(df)
```

34.Create 3×3 numpy array and display column wise mean and median

```
=import numpy as np

# Create a 3x3 NumPy array
array_3x3 = np.array([[1, 2, 3],
                      [4, 5, 6],
                      [7, 8, 9]])

# Display the original array
print("Original 3x3 Array:")
print(array_3x3)

# Calculate and display column-wise mean
column_means = np.mean(array_3x3, axis=0)
print("\nColumn-wise Mean:")
print(column_means)

# Calculate and display column-wise median
column_medians = np.median(array_3x3, axis=0)
print("\nColumn-wise Median:")
print(column_medians)
```

35.Create a series from numpy array and find max and mean of unique items of series.

```
=import numpy as np
import pandas as pd

# Create a NumPy array
numpy_array = np.array([1, 2, 3, 4, 4, 5, 5, 6, 6])

# Create a Pandas Series from the NumPy array
series_from_array = pd.Series(numpy_array)

# Find and display the unique items in the series
unique_items = series_from_array.unique()
print("Unique Items in the Series:")
print(unique_items)

# Find and display the maximum and mean of unique items in the series
max_value = unique_items.max()
mean_value = unique_items.mean()

print("\nMaximum of Unique Items:", max_value)
print("Mean of Unique Items:", mean_value)
```

36.Given data frame as below:

ID Name HRA TA DA

1001 Mohan 12000 6000 10000

1002 Sachin 13000 5000 9000

1003 Virat 11000 4000 8000

i) Compute sum of each column.

ii) Compute mean of each integer column.

iii) Compute median of each integer column

```
=import pandas as pd

# Given DataFrame
```

```
data = {
    'ID': [1001, 1002, 1003],
    'Name': ['Mohan', 'Sachin', 'Virat'],
    'HRA': [12000, 13000, 11000],
    'TA': [6000, 5000, 4000],
    'DA': [10000, 9000, 8000]
}

df = pd.DataFrame(data)

# i) Compute sum of each column
column_sums = df.sum()
print("\nSum of Each Column:")
print(column_sums)

# ii) Compute mean of each integer column
integer_columns = df.select_dtypes(include='int64')
column_means = integer_columns.mean()
print("\nMean of Each Integer Column:")
print(column_means)

# iii) Compute median of each integer column
column_medians = integer_columns.median()
print("\nMedian of Each Integer Column:")
print(column_medians)
```

37.Write a program that accept the string from user and display the same string after removing vowels from it.

=

```
def remove_vowels(input_string):
    vowels = "aeiouAEIOU"
    result_string = ''.join(char for char in input_string if char not in vowels)
    return result_string


def main():
    user_input = input("Enter a string: ")
    result = remove_vowels(user_input)
    print(f"String after removing vowels: {result}")


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

38.Create class called, library with data attributes like Acc-number publisher, title and author, the methods of the class should include

i) Read ( ) - Acc- number, title, author, publisher.

ii) Compute ( ) - to accept the number of day late, calculate and display the fine charged at the rate of Rupees 5/- per day.

iii) Display the data

```
=class Library:
    def __init__(self, acc_number, title, author, publisher):
        self.acc_number = acc_number
        self.title = title
        self.author = author
        self.publisher = publisher


    def read(self):
        # Read data attributes from the user
        self.acc_number = input("Enter Access Number: ")
        self.title = input("Enter Title: ")
        self.author = input("Enter Author: ")
        self.publisher = input("Enter Publisher: ")


    def compute_fine(self, days_late):
        # Compute and display fine at the rate of Rs. 5/- per day
        fine_rate = 5
        fine = days_late * fine_rate
        print(f"Fine Charged: Rs. {fine}")

    def display_data(self):
        # Display data attributes
        print("\nLibrary Book Details:")
        print(f"Access Number: {self.acc_number}")
        print(f"Title: {self.title}")
        print(f"Author: {self.author}")
        print(f"Publisher: {self.publisher}")

def main():
    # Create an instance of the Library class
    book = Library(acc_number="", title="", author="", publisher="")

    # Read data attributes from the user
    book.read()

    # Display data attributes
    book.display_data()

    # Calculate and display fine for 3 days late
    book.compute_fine(3)

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

39.Develop a program to print the number of lines, words and characters present in the given file? Accept the file name from user. Handle necessary exceptions.

```
def count_lines_words_characters(file_name):
    try:
        with open(file_name, 'r', encoding='utf-8') as file:
```

```
        content = file.read()

        # Count lines, words, and characters
        num_lines = len(content.split('\n'))
        num_words = len(content.split())
        num_characters = len(content)

        # Display the counts
        print(f"\nNumber of Lines: {num_lines}")
        print(f"Number of Words: {num_words}")
        print(f"Number of Characters: {num_characters}")

    except FileNotFoundError:
        print(f"Error: File '{file_name}' not found.")
    except Exception as e:
        print(f"An error occurred: {e}")

def main():
    file_name = input("Enter the file name: ")

    # Count lines, words, and characters in the given file
    count_lines_words_characters(file_name)

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

40. Write aprogram to check whether entered string & number is palindrome or not.

```
def is_palindrome(input_string):
    # Remove spaces and convert to lowercase (for case-insensitive comparison)
    cleaned_string = ''.join(input_string.split()).lower()
    reversed_string = cleaned_string[::-1]
    return cleaned_string == reversed_string
```

```
def main():
    user_input = input("Enter a string or number to check for palindrome: ")

    if is_palindrome(user_input):
        print(f"{user_input} is a palindrome.")
    else:
        print(f"{user_input} is not a palindrome.")

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

41.Develop a pythan program to remove the comment character from all the lines in the given file. Accept the file name from user

```
def remove_comments(file_name):
    try:
        with open(file_name, 'r', encoding='utf-8') as file:
            lines = file.readlines()

        # Remove comment characters from each line
        lines_without_comments = [line.split('#')[0] for line in lines]

        # Join the lines without comments
        content_without_comments = ''.join(lines_without_comments)

        # Print the content without comments
        print("\nFile Content without Comments:")
        print(content_without_comments)

    except FileNotFoundError:
        print(f"Error: File '{file_name}' not found.")
    except Exception as e:
        print(f"An error occurred: {e}")

def main():
    file_name = input("Enter the file name: ")

    # Remove comment characters from the given file
    remove_comments(file_name)

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

42. Write a pythan program to perform following operations. on MongoDB Database.

i) Create collection "EMP" with fields: Emp-name, Emp- mobile, Emp, sal, Age ii) Insert 5 documents. iii) Find the employees getting salary between 5000 to 10000.

iv) Update mobile number for the employee named as "Riddhi"

v) Display all employees in the order of "Age"

```
import pymongo

def connect_to_mongodb():
    # Connect to MongoDB
    client = pymongo.MongoClient("mongodb://localhost:27017/")

    # Create or get the "company" database
    database_name = "company"
    db = client[database_name]

    return db

def create_emp_collection(db):
    # Create "EMP" collection with fields: Emp-name, Emp-mobile, Emp-sal, Age
    emp_collection = db["EMP"]
    return emp_collection

def insert_documents(emp_collection):
    # Insert 5 documents into "EMP" collection
    employees_data = [
        {"Emp-name": "John", "Emp-mobile": "9876543210", "Emp-sal": 8000, "Age": 28},
        {"Emp-name": "Alice", "Emp-mobile": "8765432109", "Emp-sal": 6000, "Age": 25},
        {"Emp-name": "Bob", "Emp-mobile": "7654321098", "Emp-sal": 9000, "Age": 32},
        {"Emp-name": "Riddhi", "Emp-mobile": "6543210987", "Emp-sal": 7000, "Age": 29},
        {"Emp-name": "David", "Emp-mobile": "5432109876", "Emp-sal": 8000, "Age": 27},
    ]

    emp_collection.insert_many(employees_data)

def find_salary_range(emp_collection, min_salary, max_salary):
    # Find employees with salary between min_salary and max_salary
    query = {"Emp-sal": {"$gte": min_salary, "$lte": max_salary}}
    result = emp_collection.find(query)

    print("\nEmployees with Salary between {} and {}:".format(min_salary, max_salary))
    for employee in result:
        print(employee)

def update_mobile_number(emp_collection, emp_name, new_mobile_number):
    # Update mobile number for the employee named as "Riddhi"
    query = {"Emp-name": emp_name}
    update = {"$set": {"Emp-mobile": new_mobile_number}}
    emp_collection.update_one(query, update)
    print("\nMobile number updated for employee {}.".format(emp_name))

def display_employees_by_age(emp_collection):
    # Display all employees in the order of "Age"
    result = emp_collection.find().sort("Age", pymongo.ASCENDING)

    print("\nAll Employees in the Order of Age:")
    for employee in result:
        print(employee)
```

```
def main():
    try:
        db = connect_to_mongodb()
        emp_collection = create_emp_collection(db)

        # i) Create collection "EMP"
        # ii) Insert 5 documents
        insert_documents(emp_collection)

        # iii) Find employees getting salary between 5000 to 10000
        find_salary_range(emp_collection, 5000, 10000)

        # iv) Update mobile number for the employee named as "Riddhi"
        update_mobile_number(emp_collection, "Riddhi", "9999999999")

        # v) Display all employees in the order of "Age"
        display_employees_by_age(emp_collection)

    except Exception as e:
        print(f"An error occurred: {e}")

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

43. Write a pythan program to find the factorial of a given number using recursion

```
def factorial_recursive(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial_recursive(n - 1)

def main():
    try:
        # Input: Get a number from the user
        num = int(input("Enter a non-negative integer: "))

        # Check if the number is non-negative
        if num < 0:
            print("Please enter a non-negative integer.")
        else:
            # Calculate and display the factorial using recursion
            result = factorial_recursive(num)
            print(f"The factorial of {num} is: {result}")

    except ValueError:
        print("Invalid input. Please enter a valid integer.")

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

**44. Write a program to demonstrate: i) Creating a Thread without using any class
ii) Creating a Thread by extending Thread class**

```
import threading
import time

# i) Creating a Thread without using any class
def print_numbers():
    for i in range(1, 6):
        time.sleep(1)
        print(f"Thread without class: {i}")

# ii) Creating a Thread by extending Thread class
class PrintLettersThread(threading.Thread):
    def run(self):
        for letter in ['A', 'B', 'C', 'D', 'E']:
            time.sleep(1)
            print(f"Thread with Thread class: {letter}")


def main():
    # i) Creating a Thread without using any class
    thread_without_class = threading.Thread(target=print_numbers)


    # ii) Creating a Thread by extending Thread class
    thread_with_class = PrintLettersThread()


    # Start both threads
    thread_without_class.start()
    thread_with_class.start()


    # Wait for both threads to finish
    thread_without_class.join()
    thread_with_class.join()


    print("Both threads have finished.")


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

45. Write a Pythan program to check the validity of a password given by user. The password should satisfy following criteria: i) Contain at least 1 letter between a and z

ii) Contain at least 1 number between 0 and 9 iii) Contain at least 1 letter between A and Z

iv) Contain at least 1 character from $, #, @,* v) Minimum length of password : 8

vi) Maximum length of password : 20

```
import re


def is_valid_password(password):
    # i) Contain at least 1 letter between a and z
    lowercase_letter = re.search(r'[a-z]', password)
    # ii) Contain at least 1 number between 0 and 9
    digit = re.search(r'[0-9]', password)

    # iii) Contain at least 1 letter between A and Z
    uppercase_letter = re.search(r'[A-Z]', password)

    # iv) Contain at least 1 character from $, #, @, *
    special_char = re.search(r'[$#@*]', password)

    # v) Minimum length of password: 8
    # vi) Maximum length of password: 20
    length_criteria = 8 <= len(password) <= 20

    # Check all criteria
    if lowercase_letter and digit and uppercase_letter and special_char and length_criteria:
        return True
    else:
        return False

def main():
    user_password = input("Enter a password: ")

    if is_valid_password(user_password):
        print("Valid password.")
    else:
        print("Invalid password. Please follow the specified criteria.")

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

46. Write a program for synchronization of threads using RLOCK. Accept the two numbers from user and calculate factorial of both numbers simultaneonly

```
import threading

import time

import math


# RLock for synchronization

factorial_lock = threading.RLock()


# Function to calculate factorial

def calculate_factorial(number):

    with factorial_lock:

        result = 1

        for i in range(1, number + 1):

            result *= i

        return result


def calculate_and_print_factorial(number, result_list):

    result = calculate_factorial(number)

    result_list.append(result)

    print(f"Factorial of {number}: {result}")


def main():

    try:

        num1 = int(input("Enter the first number: "))

        num2 = int(input("Enter the second number: "))


        # Using a list to store results for each thread

        results = []


        # Creating threads for each number

        thread1 = threading.Thread(target=calculate_and_print_factorial, args=(num1, results))

        thread2 = threading.Thread(target=calculate_and_print_factorial, args=(num2, results))


        # Start both threads
        thread1.start()

        thread2.start()


        # Wait for both threads to finish

        thread1.join()

        thread2.join()


        # Calculate and print the product of both factorials

        product = results[0] * results[1]

        print(f"\nProduct of Factorials: {product}")


    except ValueError:

        print("Invalid input. Please enter valid integers.")


if __name__ == "__main__":

    main()
```

47. Write a pythan program i) To remove all leading 'zeros' from an IP address ii) To find all 5 character long words in a string Accept string from user

```
import re


def remove_leading_zeros(ip_address):

    # i) To remove all leading 'zeros' from an IP address

    cleaned_ip = '.'.join([str(int(part)) for part in ip_address.split('.')])

    return cleaned_ip


def find_5_character_words(input_string):

    # ii) To find all 5 character long words in a string

    words = re.findall(r'\b\w{5}\b', input_string)

    return words


def main():

    # i) To remove all leading 'zeros' from an IP address

    ip_address = input("Enter an IP address: ")
    cleaned_ip = remove_leading_zeros(ip_address)
    print(f"Cleaned IP address: {cleaned_ip}")

    # ii) To find all 5 character long words in a string
    input_string = input("\nEnter a string: ")
    five_character_words = find_5_character_words(input_string)
    print(f"5-character long words in the string: {five_character_words}")

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

48. Draw bar graph using matplotlib and decorate it by adding various elements

```
import matplotlib.pyplot as plt

def draw_bar_graph():
    # Sample data
    categories = ['Category A', 'Category B', 'Category C', 'Category D']
    values = [25, 40, 30, 50]

    # Create a bar graph
    plt.bar(categories, values, color='skyblue')

    # Decorate the bar graph
    plt.title('Bar Graph Example', fontsize=16)
    plt.xlabel('Categories', fontsize=12)
    plt.ylabel('Values', fontsize=12)
    plt.ylim(0, max(values) + 10)

    # Add data labels on each bar
    for i, value in enumerate(values):
        plt.text(i, value + 1, str(value), ha='center', va='bottom', fontsize=10, color='black')

    # Display the legend
    plt.legend(['Values'], loc='upper right')

    # Add a grid for better readability
    plt.grid(axis='y', linestyle='--', alpha=0.7)

    # Show the plot
    plt.show()

def main():
    # Draw and decorate the bar graph
    draw_bar_graph()

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

49. Prepare the pandas dataframe from csv file. perform following operations. i) Fill all 'NaN' values with the mean of respective column.

ii) Display last 5 rows.

```
import pandas as pd

def fill_nan_with_mean(df):
    # i) Fill all 'NaN' values with the mean of respective column
    df_filled = df.fillna(df.mean())

    return df_filled

def main():
    try:
        # Load the CSV file into a DataFrame
        file_path = 'your_file_path.csv'  # Replace with the path to your CSV file
        df = pd.read_csv(file_path)

        # Display original DataFrame
        print("\nOriginal DataFrame:")
        print(df)

        # Fill NaN values with the mean of respective columns
        df_filled = fill_nan_with_mean(df)

        # ii) Display last 5 rows of the DataFrame
        print("\nDataFrame after filling NaN values with mean:")
        print(df_filled.tail(5))

    except FileNotFoundError:
        print("Error: CSV file not found.")
    except Exception as e:
        print(f"An error occurred: {e}")

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

50. Write a program to illustrate numpy array attributes/functions. [4] i) ndarray. Shape

ii) np. zeros ( ) iii) np. eye ( ) iv) np. random. random ( )

```
import numpy as np

def main():
    # i) Create a numpy array and demonstrate ndarray.shape
    array_shape = np.array([[1, 2, 3], [4, 5, 6]])
    print(f"i) Numpy array:\n{array_shape}")
    print(f"   Shape of the array: {array_shape.shape}\n")

    # ii) Create a numpy array filled with zeros using np.zeros()
    zeros_array = np.zeros((3, 4))
    print(f"ii) Numpy array filled with zeros:\n{zeros_array}\n")

    # iii) Create an identity matrix using np.eye()
    identity_matrix = np.eye(3)
    print(f"iii) Identity matrix:\n{identity_matrix}\n")


    # iv) Create a numpy array with random values using np.random.random()
    random_array = np.random.random((2, 3))
    print(f"iv) Numpy array with random values:\n{random_array}\n")


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

51. Read data from csv five and create dataframe. Perform following operations. i) Display list of all columns. ii) Display data with last three rows and first three columns

```
import pandas as pd


def main():
    try:
        # i) Read data from CSV file and create DataFrame
        file_path = 'your_file_path.csv'  # Replace with the path to your CSV file
        df = pd.read_csv(file_path)


        # ii) Display list of all columns
        print("\ni) List of all columns:")
        print(df.columns.tolist())


        # iii) Display data with last three rows and first three columns
        print("\nii) Data with last three rows and first three columns:")
        print(df.iloc[-3:, :3])


    except FileNotFoundError:
        print("Error: CSV file not found.")
    except Exception as e:
        print(f"An error occurred: {e}")


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

52. Draw line graph using matplot lib and decorate it by adding various elements. Use suitable data.

```
import matplotlib.pyplot as plt


def draw_line_graph():
    # Sample data
    x_values = [1, 2, 3, 4, 5]
    y_values = [10, 14, 8, 17, 12]

    # Create a line graph
    plt.plot(x_values, y_values, marker='o', linestyle='-', color='blue', label='Line Graph')

    # Decorate the line graph
    plt.title('Line Graph Example', fontsize=16)
    plt.xlabel('X-axis', fontsize=12)
    plt.ylabel('Y-axis', fontsize=12)
    plt.legend(loc='upper left')
    plt.grid(True, linestyle='--', alpha=0.7)

    # Add annotations to specific points
    for i, (x, y) in enumerate(zip(x_values, y_values)):
        plt.annotate(f'({x}, {y})', (x, y), textcoords="offset points", xytext=(0, 5), ha='center', fontsize=8)

    # Show the plot
    plt.show()

def main():
    # Draw and decorate the line graph
    draw_line_graph()

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