In [1]:

```python
#1Write a Python program to accept n numbers in list and remove du


# Accept numbers in a list and remove duplicates
def remove_duplicates():
    n = int(input("Enter the number of elements in the list: "))
    numbers = []

    for i in range(n):
        num = int(input(f"Enter number {i + 1}: "))
        numbers.append(num)

    # Remove duplicates by converting the list to a set and back t
    unique_numbers = list(set(numbers))

    print("List after removing duplicates:", unique_numbers)

# Call the function
remove_duplicates()
```

```
Enter the number of elements in the list: 5
Enter number 1: 21
Enter number 2: 12
Enter number 3: 21
Enter number 4: 3
Enter number 5: 4
List after removing duplicates: [4, 3, 12, 21]
```

In [2]:

```
#2Write a Python program to merge two lists and remove any duplica

def merge_and_remove_duplicates():
    # Accept the first list from the user
    n1 = int(input("Enter the number of elements in the first list
    list1 = [int(input(f"Enter element {i + 1} for the first list:

    # Accept the second list from the user
    n2 = int(input("Enter the number of elements in the second lis
    list2 = [int(input(f"Enter element {i + 1} for the second list

    # Merge both lists and remove duplicates
    merged_list = list(set(list1 + list2))

    print("Merged list without duplicates:", merged_list)

# Call the function
merge_and_remove_duplicates()
```

```
Enter the number of elements in the first list: 4
Enter element 1 for the first list: 12
Enter element 2 for the first list: 3
Enter element 3 for the first list: 4
Enter element 4 for the first list: 5
Enter the number of elements in the second list: 4
Enter element 1 for the second list: 54
Enter element 2 for the second list: 3
Enter element 3 for the second list: 4
Enter element 4 for the second list: 5
Merged list without duplicates: [3, 4, 5, 12, 54]
```

In [3]:

```
#3Write a Python program to reverse a list without using built-in

def reverse_list():
    # Accept the list from the user
    n = int(input("Enter the number of elements in the list: "))
    numbers = [int(input(f"Enter element {i + 1}: ")) for i in ran

    # Reverse the list manually
    reversed_list = []
    for i in range(len(numbers) - 1, -1, -1):
        reversed_list.append(numbers[i])

    print("Reversed list:", reversed_list)

# Call the function
reverse_list()
```

```
Enter the number of elements in the list: 2
Enter element 1: 23
Enter element 2: 43
Reversed list: [43, 23]
```

In [4]:

```python
#4Write a Python program to rotate a list to the right by a given 

def rotate_list():
    # Accept the list from the user
    n = int(input("Enter the number of elements in the list: "))
    numbers = [int(input(f"Enter element {i + 1}: ")) for i in ran

    # Accept the number of positions to rotate
    k = int(input("Enter the number of positions to rotate: "))

    # Calculate the effective number of rotations (handles cases w
    k = k % n

    # Rotate the list to the right by k positions manually
    rotated_list = numbers[-k:] + numbers[:-k]

    print("Rotated list:", rotated_list)

# Call the function
rotate_list()
```

```
Enter the number of elements in the list: 5
Enter element 1: 12
Enter element 2: 32
Enter element 3: 3
Enter element 4: 12
Enter element 5: 32
Enter the number of positions to rotate: 3
Rotated list: [3, 12, 32, 12, 32]
```

In [5]:

```python
#5Write a Python program that combines two sorted lists into a sing

def merge_sorted_lists():
    # Accept the first sorted list from the user
    n1 = int(input("Enter the number of elements in the first sorte
    list1 = [int(input(f"Enter element {i + 1} for the first sorte

    # Accept the second sorted list from the user
    n2 = int(input("Enter the number of elements in the second sor
    list2 = [int(input(f"Enter element {i + 1} for the second sorte

    # Merge the two sorted lists
    merged_list = []
    i, j = 0, 0

    while i < n1 and j < n2:
        if list1[i] < list2[j]:
            merged_list.append(list1[i])
            i += 1
        else:
            merged_list.append(list2[j])
            j += 1

    # Add any remaining elements from both lists
    while i < n1:
        merged_list.append(list1[i])
        i += 1

    while j < n2:
        merged_list.append(list2[j])
        j += 1

    print("Merged sorted list:", merged_list)

# Call the function
merge_sorted_lists()
```

```
Enter the number of elements in the first sorted list: 3
Enter element 1 for the first sorted list: 21
Enter element 2 for the first sorted list: 12
Enter element 3 for the first sorted list: 32
Enter the number of elements in the second sorted list: 4
Enter element 1 for the second sorted list: 3
Enter element 2 for the second sorted list: 2
Enter element 3 for the second sorted list: 12
Enter element 4 for the second sorted list: 32
Merged sorted list: [3, 2, 12, 21, 12, 32, 32]
```

In [6]:

```python
#6Write a Python program to swap the values of two variables using
# Swap two variables using a tuple
def swap_variables():
    # Accept two variables from the user
    a = input("Enter the value of a: ")
    b = input("Enter the value of b: ")

    # Swap values using a tuple
    a, b = b, a

    print(f"After swapping: a = {a}, b = {b}")

# Call the function
swap_variables()

```

```
Enter the value of a: 13
Enter the value of b: 21
After swapping: a = 21, b = 13
```

In [7]:

```python
#7Write a program to concatenate two tuples and find the length of

def concatenate_tuples():

    tuple1 = tuple(input("Enter the elements of the first tuple (s
    tuple2 = tuple(input("Enter the elements of the second tuple (


    concatenated_tuple = tuple1 + tuple2


    length = len(concatenated_tuple)

    print("Concatenated tuple:", concatenated_tuple)
    print("Length of the concatenated tuple:", length)

# Call the function
concatenate_tuples()

```

```
Enter the elements of the first tuple (separated by spaces): 12 32 45
65 67
Enter the elements of the second tuple (separated by spaces): 45 65 7
8 90 43
Concatenated tuple: ('12', '32', '45', '65', '67', '45', '65', '78',
'90', '43')
Length of the concatenated tuple: 10
```

In [9]:

```python
#8Write a Python program to find the intersection of two sets with

def find_intersection():

    set1 = set(input("Enter the elements of the first set (separat
    set2 = set(input("Enter the elements of the second set (separa
    intersection = {element for element in set1 if element in set2

    print("Intersection of the two sets:", intersection)

# Call the function
find_intersection()

```

```
Enter the elements of the first set (separated by spaces): 1 2 3 5 4
6 7
Enter the elements of the second set (separated by spaces): 8 9 7 64
3
Intersection of the two sets: {'7', '3'}
```

In [10]:

```python
#9Using a set, write a Python program that removes duplicate eleme

def remove_duplicates_preserve_order():
    # Accept the list from the user
    n = int(input("Enter the number of elements in the list: "))
    elements = [input(f"Enter element {i + 1}: ") for i in range(n

    # Use a set to track seen elements
    seen = set()
    unique_elements = []

    for element in elements:
        if element not in seen:
            unique_elements.append(element)
            seen.add(element)

    print("List with unique elements in order:", unique_elements)

# Call the function
remove_duplicates_preserve_order()

```

```
Enter the number of elements in the list: 3
Enter element 1: 34
Enter element 2: 56
Enter element 3: 76
List with unique elements in order: ['34', '56', '76']
```

In [ ]:

```python

```

In [11]:

```python
#10Create a Python program that takes a set of numbers and returns

def filter_even_numbers():
    # Accept a set of numbers from the user
    numbers = set(map(int, input("Enter the numbers in the set (se


    even_numbers = {num for num in numbers if num % 2 == 0}

    print("Set of even numbers:", even_numbers)

# Call the function
filter_even_numbers()

```

```
Enter the numbers in the set (separated by spaces): 34 22 31 13 45 67
Set of even numbers: {34, 22}
```

In [12]:

```python
#11Write a Python program to merge two dictionaries and resolve any

# Merge two dictionaries and sum values for common keys
def merge_dictionaries():
    # Accept the first dictionary from the user
    dict1 = eval(input("Enter the first dictionary (e.g., {'a': 1,

    # Accept the second dictionary from the user
    dict2 = eval(input("Enter the second dictionary (e.g., {'b': 3

    # Merge dictionaries with conflict resolution
    merged_dict = dict1.copy()  # Start with the first dictionary
    for key, value in dict2.items():
        if key in merged_dict:
            merged_dict[key] += value  # Sum values for common key
        else:
            merged_dict[key] = value  # Add new key-value pair

    print("Merged dictionary:", merged_dict)

# Call the function
merge_dictionaries()

```

```
Enter the first dictionary (e.g., {'a': 1, 'b': 2}):  {'a': 1, 'b':
2}
Enter the second dictionary (e.g., {'b': 3, 'c': 4}): {'b': 3, 'c':
4}
Merged dictionary: {'a': 1, 'b': 5, 'c': 4}
```

In [13]:

```python

#12Write a Python program that takes a dictionary of student names
# Find the student with the highest score
def find_top_student():
    # Accept the dictionary of student scores from the user
    scores = eval(input("Enter the dictionary of student scores (e

    # Find the student with the highest score
    top_student = max(scores, key=scores.get)

    print("Student with the highest score:", top_student)

# Call the function
find_top_student()

```

```
Enter the dictionary of student scores (e.g., {'Alice': 85, 'Bob': 9
0}): {'Om':99,'Sagar':95}
Student with the highest score: Om
```

In [14]:

```python

#13Using a dictionary, write a Python program that groups the list
# Group list of tuples by the first element
def group_tuples_by_first_element():
    # Accept the list of tuples from the user
    tuples_list = eval(input("Enter the list of tuples (e.g., [('a

    # Create an empty dictionary to group the tuples
    grouped_dict = {}

    for key, value in tuples_list:
        if key not in grouped_dict:
            grouped_dict[key] = []  # Initialize an empty list for
        grouped_dict[key].append(value)  # Append the value to the

    print("Grouped dictionary:", grouped_dict)

# Call the function
group_tuples_by_first_element()

```

```
Enter the list of tuples (e.g., [('a', 1), ('b', 2), ('a', 3)]):
[('a', 1), ('b', 2), ('a', 3)]
Grouped dictionary: {'a': [1, 3], 'b': [2]}
```

In [15]:

```
#14Write a Python program to check if a given key already exists i
# Check if a key exists and replace it with another key/value pair
def replace_key_in_dictionary():
    # Accept the dictionary from the user
    dictionary = eval(input("Enter the dictionary (e.g., {'a': 1,

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

    # Accept the new key and value
    new_key = input("Enter the new key: ")
    new_value = input("Enter the new value: ")

    # Check if the key exists and replace it
    if key_to_check in dictionary:
        dictionary.pop(key_to_check)  # Remove the old key
        dictionary[new_key] = new_value  # Add the new key-value p
        print(f"Key '{key_to_check}' found and replaced with ({new
    else:
        print(f"Key '{key_to_check}' not found in the dictionary."

    # Print the updated dictionary
    print("Updated dictionary:", dictionary)

# Call the function
replace_key_in_dictionary()

```

```
Enter the dictionary (e.g., {'a': 1, 'b': 2}):  {'a': 1, 'b': 2}
Enter the key to check: a
Enter the new key: b
Enter the new value: 09
Key 'a' found and replaced with (b: 09)
Updated dictionary: {'b': '09'}
```

In [16]:

```
#15Write a Python script to generate and print a dictionary which
# Generate a dictionary with numbers and their squares
def generate_square_dictionary():
    # Accept the value of n from the user
    n = int(input("Enter the value of n: "))

    # Create the dictionary using dictionary comprehension
    square_dict = {x: x * x for x in range(1, n + 1)}

    print("Generated dictionary:", square_dict)

# Call the function
generate_square_dictionary()

```

```
Enter the value of n: 10
Generated dictionary: {1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49,
8: 64, 9: 81, 10: 100}
```

In [17]:

```
#16Write a Python program to convert a tuple of string values to a

def convert_tuple():

    str_tuple = eval(input("Enter a tuple of string values (e.g.,


    int_tuple = tuple(tuple(int(num) for num in inner_tuple) for i

    print("Converted tuple:", int_tuple)

# Call the function
convert_tuple()
```

```
Enter a tuple of string values (e.g., (('333', '33'), ('1416', '5
5'))): (('333', '33'), ('1416', '55'))
Converted tuple: ((333, 33), (1416, 55))
```

In [18]:

```
#17Write a Python program to compute element-wise sum of given tup
def element_wise_sum():
    tuple1 = tuple(map(int, input("Enter the first tuple (e.g., (1
    tuple2 = tuple(map(int, input("Enter the second tuple (e.g., (
    tuple3 = tuple(map(int, input("Enter the third tuple (e.g., (2

    result = tuple(a + b + c for a, b, c in zip(tuple1, tuple2, tu
    print("Element-wise sum of the tuples:", result)

element_wise_sum()
```

```
Enter the first tuple (e.g., (1, 2, 3, 4)): (1, 2, 3, 4)
Enter the second tuple (e.g., (3, 5, 2, 1)): (3, 5, 2, 1)
Enter the third tuple (e.g., (2, 2, 3, 1)): (2, 2, 3, 1)
Element-wise sum of the tuples: (6, 9, 8, 6)
```

In [19]:

```python
#18 Write a python program to count repeated characters in a strin
from collections import Counter

# Sample string
sample_string = 'the quick brown fox jumps over the lazy dog'

# Count characters
char_count = Counter(sample_string)

# Filter characters with more than 1 occurrence
repeated_chars = {char: count for char, count in char_count.items(

# Display output
for char, count in repeated_chars.items():
    print(f"{repr(char)}-{count}", end=", ")

```

```
't'-2, 'h'-2, 'e'-3, ' '-8, 'u'-2, 'r'-2, 'o'-4,
```

In [20]:

```python
#19 Write a python script to find the repeated items of a tuple
def find_repeated_items():
    user_input = input("Enter a tuple of numbers (e.g., (1, 2, 3, 
    input_tuple = tuple(map(int, user_input.strip("()").split(',')

    repeated_items = {item for item in input_tuple if input_tuple.
    print("Repeated items in the tuple:", repeated_items)

find_repeated_items()

```

```
Enter a tuple of numbers (e.g., (1, 2, 3, 4, 5, 1, 2, 6, 3, 7, 4)):
(1, 2, 3, 4, 5, 1, 2, 6, 3, 7, 4)
Repeated items in the tuple: {1, 2, 3, 4}
```

In [21]:

```python
#20Write a python script to generate Fibonacci terms using generat
def fibonacci_generator():

    a, b = 0, 1
    while True:
        yield a
        a, b = b, a + b

# Example usage
if __name__ == "__main__":
    fib_gen = fibonacci_generator()
    num_terms = int(input("Enter the number of Fibonacci terms to 
    for _ in range(num_terms):
        print(next(fib_gen))

```

```
Enter the number of Fibonacci terms to generate: 8
0
1
1
2
3
5
8
13
```

In [22]:

```python
#21Write a python program to accept string and remove the characte
def remove_odd_index_characters(input_string):
    """
    Function to remove characters with odd index values from a str
    """
    return ''.join(char for index, char in enumerate(input_string)

# Main program
if __name__ == "__main__":
    user_input = input("Enter a string: ")
    result = remove_odd_index_characters(user_input)
    print(f"String after removing characters with odd index values

```

```
Enter a string: hello brother
String after removing characters with odd index values: hlobohr
```

In [23]:

```python
#22Given a list of integers, create a Python function that returns
def second_largest(numbers):
    """
    Function to return the second largest number in a list of inte
    """
    if len(numbers) < 2:
        raise ValueError("List must contain at least two distinct

    unique_numbers = list(set(numbers))  # Remove duplicates
    if len(unique_numbers) < 2:
        raise ValueError("List must contain at least two distinct

    unique_numbers.sort(reverse=True)  # Sort in descending order
    return unique_numbers[1]  # Return the second largest number

# Example usage
if __name__ == "__main__":
    try:

        nums = list(map(int, input("Enter a list of integers separ
        print(f"The second largest number is: {second_largest(nums
    except ValueError as e:
        print(e)
```

```
Enter a list of integers separated by spaces: 21 32 43 54 65
The second largest number is: 54
```

In [24]:

```python
#23Write a Python function that accepts a string and calculate the

def count_case_characters(input_string):

    upper_case_count = sum(1 for char in input_string if char.isup
    lower_case_count = sum(1 for char in input_string if char.islo

    return upper_case_count, lower_case_count

# Example usage
if __name__ == "__main__":
    sample_string = 'The quick Brown Fox'
    upper_count, lower_count = count_case_characters(sample_string
    print(f"No. of Upper case characters: {upper_count}")
    print(f"No. of Lower case characters: {lower_count}")

```

```
No. of Upper case characters: 3
No. of Lower case characters: 13
```

In [25]:

```
#24Create a function that counts the frequency of each word in a g
def word_frequency(sentence):
    words = sentence.split()
    frequency = {}
    for word in words:
        frequency[word] = frequency.get(word, 0) + 1
    return frequency
sentence = "the quick brown fox jumps over the lazy dog"
print(word_frequency(sentence))

```

```
{'the': 2, 'quick': 1, 'brown': 1, 'fox': 1, 'jumps': 1, 'over': 1,
'lazy': 1, 'dog': 1}
```

In [26]:

```
#25Create a Python function that takes a dictionary and returns a
def swap_dict_keys_values():
    user_input = input("Enter a dictionary (e.g., {'a': 1, 'b': 2,
    try:
        d = eval(user_input)
        if not isinstance(d, dict):
            raise ValueError("Input is not a dictionary.")
        swapped_dict = {value: key for key, value in d.items()}
        print("Swapped dictionary:", swapped_dict)
    except Exception as e:
        print("Invalid input. Please enter a valid dictionary.", e

swap_dict_keys_values()

```

```
Enter a dictionary (e.g., {'a': 1, 'b': 2, 'c': 3}): {'a': 1, 'b': 2,
'c': 3}
Swapped dictionary: {1: 'a', 2: 'b', 3: 'c'}
```

In [27]:

```
#26Given a tuple of integers, create a function that converts the
def tuple_to_sorted_list():
    user_input = input("Enter a tuple of integers (e.g., (5, 3, 8,
    t = eval(user_input)
    sorted_list = sorted(list(t))
    print("Sorted list:", sorted_list)

tuple_to_sorted_list()

```

```
Enter a tuple of integers (e.g., (5, 3, 8, 1, 4)): (5, 3, 8, 1, 4)
Sorted list: [1, 3, 4, 5, 8]
```

In [28]:

```
#27Create a Python function that takes a tuple of strings and retu
def string_lengths():
    user_input = input("Enter a tuple of strings (e.g., ('apple',
    t = eval(user_input)
    lengths = tuple(len(s) for s in t)
    print("Lengths of each string:", lengths)

string_lengths()

```

```
Enter a tuple of strings (e.g., ('apple', 'banana', 'cherry')): ('app
le', 'banana', 'cherry')
Lengths of each string: (5, 6, 6)
```

In [29]:

```
#28Create a Python function that takes a list of strings and retur
def convert_to_uppercase():
    user_input = input("Enter a list of strings (e.g., ['hello', '
    strings = eval(user_input)
    uppercase_list = [s.upper() for s in strings]
    print("Uppercase list:", uppercase_list)

convert_to_uppercase()

```

```
Enter a list of strings (e.g., ['hello', 'world', 'python']): ['hell
o', 'world', 'python']
Uppercase list: ['HELLO', 'WORLD', 'PYTHON']
```

In [34]:

```
#29Using a tuple of numbers, create a function that calculates and
import ast

def sum_tuple_elements():
    user_input = input("Enter a tuple of numbers (e.g., (10, 20, 3
    # Use ast.literal_eval to parse the input safely
    t = ast.literal_eval(user_input)

    # Check if the input is a tuple and contains only numbers
    if isinstance(t, tuple) and all(isinstance(i, (int, float)) fo
        total = sum(t)
        print("Sum of elements in the tuple:", total)
    else:
        print("Invalid input: Please enter a valid tuple of number

sum_tuple_elements()

```

```
Enter a tuple of numbers (e.g., (10, 20, 30, 40)): (10, 20, 30, 40)
Sum of elements in the tuple: 100
```

In [35]:

```
#30Create a function that takes two sets and returns the symmetric
def symmetric_difference():
    set1 = set(map(int, input("Enter the first set (e.g., {1, 2, 3
    set2 = set(map(int, input("Enter the second set (e.g., {3, 4,

    result = set1.symmetric_difference(set2)
    print("Symmetric difference between the sets:", result)

symmetric_difference()

```

```
Enter the first set (e.g., {1, 2, 3, 4}): {1, 2, 3, 4}
Enter the second set (e.g., {3, 4, 5, 6}): {3, 4, 5, 6}
Symmetric difference between the sets: {1, 2, 5, 6}
```

In [36]:

```

#31Write a Python function that checks if a given set is a subset
def is_subset(set1, set2):

    return set1.issubset(set2)
set_a = {1, 2}
set_b = {1, 2, 3, 4}
print(is_subset(set_a, set_b))

```

```
True
```

In [38]:

```python
#32**Write python script using package to calculate area and volum
def cube_area(side_length):
    """Calculate the surface area of a cube."""
    return 6 * (side_length ** 2)

def cube_volume(side_length):
    """Calculate the volume of a cube."""
    return side_length ** 3

def sphere_area(radius):
    """Calculate the surface area of a sphere."""
    pi = 3.141592653589793
    return 4 * pi * (radius ** 2)

def sphere_volume(radius):
    """Calculate the volume of a sphere."""
    pi = 3.141592653589793
    return (4 / 3) * pi * (radius ** 3)

def main():
    print("Geometry Calculations")

    # Cube calculations
    side_length = float(input("Enter the side length of the cube:
    print(f"Cube Surface Area: {cube_area(side_length):.2f}")
    print(f"Cube Volume: {cube_volume(side_length):.2f}")

    # Sphere calculations
    radius = float(input("Enter the radius of the sphere: "))
    print(f"Sphere Surface Area: {sphere_area(radius):.2f}")
    print(f"Sphere Volume: {sphere_volume(radius):.2f}")

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

```
Geometry Calculations
Enter the side length of the cube: 5
Cube Surface Area: 150.00
Cube Volume: 125.00
Enter the radius of the sphere: 7
Sphere Surface Area: 615.75
Sphere Volume: 1436.76
```

In [39]:

```python

#33.** Write a Python program to input a positive integer. Display

def input_positive_integer():
    try:
        num = int(input("Enter a positive integer: "))
        if num <= 0:
            raise ValueError("The number must be positive.")
        print(f"Valid input! You entered: {num}")
    except ValueError as e:
        print(f"Invalid input: {e}")

input_positive_integer()

```

```
Enter a positive integer: 89
Valid input! You entered: 89
```

In [40]:

```python
#34 Write a python program to check the given number is prime or n
def is_prime(num):
    if num <= 1:
        return False
    for i in range(2, int(num ** 0.5) + 1):
        if num % i == 0:
            return False
    return True

def check_prime():
    try:
        num = int(input("Enter a number to check if it is prime: "
        if num < 0:
            raise ValueError("Negative numbers cannot be prime.")
        if is_prime(num):
            print(f"{num} is a prime number.")
        else:
            print(f"{num} is not a prime number.")
    except ValueError as e:
        print(f"Invalid input: {e}")

check_prime()

```

```
Enter a number to check if it is prime: 89
89 is a prime number.
```

In [41]:

```python
#35 Write a program for registering for Driving Licence, if age is
def register_driving_licence():
    try:
        age = int(input("Enter your age: "))
        if age < 18:
            raise Exception("Not a valid age.")
        print("Successful registration!")
    except Exception as e:
        print(f"Registration failed: {e}")
register_driving_licence()

```

```
Enter your age: 87
Successful registration!
```

In [42]:

```python
#36 Write a function to compute 5/0 and use try/except to catch th
def compute_division():
    try:
        result = 5 / 0
        print(f"Result: {result}")
    except ZeroDivisionError as e:
        print(f"Exception caught: {e}")
compute_division()

```

```
Exception caught: division by zero
```

In [43]:

```python

#37Write a python script to define a class student having members r
class Student:
    def __init__(self, roll_no, name, age, gender):
        self.roll_no = roll_no
        self.name = name
        self.age = age
        self.gender = gender

class Test(Student):
    def __init__(self, roll_no, name, age, gender, marks1, marks2,
        super().__init__(roll_no, name, age, gender)
        self.marks1 = marks1
        self.marks2 = marks2
        self.marks3 = marks3

    def total_marks(self):
        return self.marks1 + self.marks2 + self.marks3

    def display(self):
        print(f"Roll No: {self.roll_no}")
        print(f"Name: {self.name}")
        print(f"Age: {self.age}")
        print(f"Gender: {self.gender}")
        print(f"Total Marks: {self.total_marks()}")

# Creating three objects
student1 = Test(101, 'Alice', 20, 'Female', 85, 90, 92)
student2 = Test(102, 'Bob', 21, 'Male', 78, 82, 88)
student3 = Test(103, 'Charlie', 22, 'Male', 92, 89, 95)

student1.display()
student2.display()
student3.display()

```

```
Roll No: 101
Name: Alice
Age: 20
Gender: Female
Total Marks: 267
Roll No: 102
Name: Bob
Age: 21
Gender: Male
Total Marks: 248
Roll No: 103
Name: Charlie
Age: 22
Gender: Male
Total Marks: 276
```

In [45]:

```python
#38 Define a class Employee having members id, name, department, s
class Employee:
    def __init__(self, emp_id, name, department, salary):
        self.emp_id = emp_id
        self.name = name
        self.department = department
        self.salary = salary

    def accept(self):
        pass

    def display(self):
        print(f"ID: {self.emp_id}, Name: {self.name}, Department:

class Manager(Employee):
    def __init__(self, emp_id, name, department, salary, bonus):
        super().__init__(emp_id, name, department, salary)
        self.bonus = bonus

    def total_salary(self):
        return self.salary + self.bonus

    def accept(self):
        pass

    def display(self):
        super().display()
        print(f"Bonus: {self.bonus}")
        print(f"Total Salary: {self.total_salary()}")

# Creating Manager objects
manager1 = Manager(101, 'John', 'Sales', 50000, 8000)
manager2 = Manager(102, 'Emma', 'HR', 60000, 10000)
manager3 = Manager(103, 'David', 'Finance', 55000, 12000)

# List of managers
managers = [manager1, manager2, manager3]

# Find the manager with the maximum total salary
max_salary_manager = max(managers, key=lambda m: m.total_salary())
max_salary_manager.display()
```

```
ID: 102, Name: Emma, Department: HR, Salary: 60000
Bonus: 10000
Total Salary: 70000
```

In [46]:

```python
# Write Python class to perform addition of two complex numbers us
class Complex:
    def __init__(self, real, imag):
        self.real = real
        self.imag = imag

    def __add__(self, other):
        return Complex(self.real + other.real, self.imag + other.i

    def display(self):
        print(f"{self.real} + {self.imag}i")

# Creating complex number objects
c1 = Complex(3, 4)
c2 = Complex(1, 2)

# Adding complex numbers
result = c1 + c2
result.display()

```

```
4 + 6i
```

In [47]:

```python
#39 Write Python class to perform addition of two complex numbers (
class ComplexNumber:
    def __init__(self, real, imag):
        self.real = real
        self.imag = imag

    def __add__(self, other):
        # Overloading the + operator for complex number addition
        return ComplexNumber(self.real + other.real, self.imag + o

    def __str__(self):
        return f"{self.real} + {self.imag}i"

# Example usage
complex1 = ComplexNumber(3, 4)
complex2 = ComplexNumber(1, 2)

result = complex1 + complex2  # Using overloaded + operator
print("Sum of complex numbers:", result)

```

```
Sum of complex numbers: 4 + 6i
```

In [49]:

```python
#40Write a Python class which has two methods get_String and print_
class StringManipulator:
    def __init__(self):
        self.input_string = ""

    def get_string(self):
        self.input_string = input("Enter a string: ")

    def print_string(self):
        print(self.input_string.upper())

    def reverse_word_by_word(self):
        words = self.input_string.split()
        reversed_words = ' '.join(reversed(words))
        print(reversed_words.lower())

# Using the class
str_manipulator = StringManipulator()
str_manipulator.get_string()
str_manipulator.print_string()
str_manipulator.reverse_word_by_word()

```

```
Enter a string: hello my name is omkar
HELLO MY NAME IS OMKAR
omkar is name my hello
```

In [51]:

```python
#41Write a Python script using class to reverse a String (sentence)
class StringReversal:
    def __init__(self, sentence):
        self.sentence = sentence

    def reverse_words(self):
        # Split the sentence into words and reverse the list of wo
        words = self.sentence.split()
        reversed_words = ' '.join(reversed(words))
        return reversed_words

# Example usage
sentence = input("Enter a sentence: ")
string_reversal = StringReversal(sentence)
reversed_sentence = string_reversal.reverse_words()
print("Reversed sentence:", reversed_sentence)

```

```
Enter a sentence: hello my name is omkar
Reversed sentence: omkar is name my hello
```

In [54]:

```python
#42Write a program to check if the input year is leap year or not.
import re

def is_leap_year(year):
    if re.match(r'^\d{4}$', year):
        year = int(year)
        if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0
            return True
        else:
            return False
    else:
        return False

year = input("Enter a year: ")
if is_leap_year(year):
    print("Leap Year")
else:
    print("Not a Leap Year")

```

```
Enter a year: 2024
Leap Year
```

In [55]:

```python
#43 Write a Python code to validate email address using regular ex
import re

def validate_email(email):
    pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
    if re.match(pattern, email):
        return True
    else:
        return False

email = input("Enter an email address: ")
if validate_email(email):
    print("Valid email address")
else:
    print("Invalid email address")

```

```
Enter an email address: om@gmail.com
Valid email address
```

In [56]:

```python
#44 Write 2 Python functions to validate IP address and phone numb
import re

def validate_ip(ip):
    pattern = r'^(?:[0-9]{1,3}\.){3}[0-9]{1,3}$'
    if re.match(pattern, ip):
        return True
    else:
        return False

def validate_phone_number(phone):
    pattern = r'^\+?[0-9]{1,3}?[-.\s]?[0-9]{1,4}[-.\s]?[0-9]{1,4}[
    if re.match(pattern, phone):
        return True
    else:
        return False

ip = input("Enter an IP address: ")
phone = input("Enter a phone number: ")

if validate_ip(ip):
    print("Valid IP address")
else:
    print("Invalid IP address")

if validate_phone_number(phone):
    print("Valid phone number")
else:
    print("Invalid phone number")

```

```
Enter an IP address: 127.0.0
Enter a phone number: 9870567487
Invalid IP address
Valid phone number
```

In [57]:

```python
#45 Write a Python class to find validity of a string of parenthes
import re

def validate_parentheses(string):
    return bool(re.match(r'^[\(\)\{\}\[\]]*$', string))

parentheses = input("Enter string of parentheses: ")
if validate_parentheses(parentheses):
    print("Valid Parentheses")
else:
    print("Invalid Parentheses")

```

```
Enter string of parentheses: ("ELLO"]
Invalid Parentheses
```

In [58]:

```python
#46 Write python code to validate URL (e.g. https://www.google.com
import re

def validate_url(url):
    return bool(re.match(r'^(https?|ftp)://[^\s/$.?#].[^\s]*$', ur

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

```

```
Enter URL: http://localhost:8888/notebooks/labbook.ipynb (http://loca
lhost:8888/notebooks/labbook.ipynb)
Valid URL
```

In [59]:

```python
#47. Write python code to validate password. Password should conta
import re

def validate_password(password):
    return bool(re.match(r'^(?=.*[A-Z])(?=.*\d)(?=.*[\W_]).{8,15}$

password = input("Enter Password: ")
if validate_password(password):
    print("Valid Password")
else:
    print("Invalid Password")


```

```
Enter Password: Omkar@3223
Valid Password
```

In [61]:

```python

#48Write a program for synchronization of Threads using RLOCK. Acc
import threading

# Creating a Reentrant Lock
lock = threading.RLock()

def factorial(number):
    lock.acquire()
    result = 1
    for i in range(1, number + 1):
        result *= i
    lock.release()
    return result

def calculate_factorial(number, name):
    print(f"Thread {name} calculating factorial of {number}")
    fact = factorial(number)
    print(f"Thread {name} - Factorial of {number} is {fact}")

def main():
    num1 = int(input("Enter first number: "))
    num2 = int(input("Enter second number: "))

    # Creating threads
    thread1 = threading.Thread(target=calculate_factorial, args=(n
    thread2 = threading.Thread(target=calculate_factorial, args=(n

    # Starting threads
    thread1.start()
    thread2.start()

    # Wait for both threads to complete
    thread1.join()
    thread2.join()

if __name__ == "__main__":
    main()

```

```
Enter first number: 5
Enter second number: 3
Thread 1 calculating factorial of 5
Thread 1 - Factorial of 5 is 120
Thread 2 calculating factorial of 3
Thread 2 - Factorial of 3 is 6
```

In [62]:

```python
#49 Write a multithreading program where one thread prints square 
import threading
import time

def print_square(number):
    print(f"Square of {number}: {number ** 2}")

def print_factorial(number):
    factorial = 1
    for i in range(1, number + 1):
        factorial *= i
    print(f"Factorial of {number}: {factorial}")

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

    start_time = time.time()

    # Creating threads
    thread1 = threading.Thread(target=print_square, args=(number,)
    thread2 = threading.Thread(target=print_factorial, args=(numbe

    # Starting threads
    thread1.start()
    thread2.start()

    # Wait for both threads to complete
    thread1.join()
    thread2.join()

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

# Run the main function
main()

```

```
Enter a number: 23
Square of 23: 529
Factorial of 23: 25852016738884976640000
Total time taken: 0.0049364566802978516 seconds
```

In [63]:

```python
#50 Write a Python program to perform following operation on Mongol
#i) Create collection "Book" with fields Book-name, Book-code, Bool
#ii)     Insert 5 documents
#iii)   Find the books whose price between 500-800
#iv)     Update price of book "Python programming" as 1000
#v)  Display all books in the order of publication Year
from pymongo import MongoClient

client = MongoClient('mongodb://localhost:27017/')

# Select or create the database and collection
db = client['Library']
collection = db['Book']

# i) Create collection "Book" with the fields: Book-name, Book-cod
# The collection and database are created automatically when we ins

# ii) Insert 5 documents
books = [
    {"Book-name": "Python Programming", "Book-code": "B001", "Book
    {"Book-name": "Data Science", "Book-code": "B002", "Book-Autho
    {"Book-name": "Machine Learning", "Book-code": "B003", "Book-A
    {"Book-name": "Artificial Intelligence", "Book-code": "B004",
    {"Book-name": "Database Systems", "Book-code": "B005", "Book-A
]

# Insert the books into the collection
collection.insert_many(books)

# iii) Find the books whose price is between 500 and 800
print("Books with price between 500 and 800:")
price_range_books = collection.find({"Book-Price": {"$gte": 500, "
for book in price_range_books:
    print(book)

# iv) Update price of book "Python Programming" to 1000
collection.update_one({"Book-name": "Python Programming"}, {"$set"
print("\nUpdated price for 'Python Programming' to 1000.")

# v) Display all books in the order of publication year
print("\nBooks sorted by publication year:")
sorted_books = collection.find().sort("Book-publication-year", 1)
for book in sorted_books:
    print(book)

# Close the connection
client.close()

```

```
---------------------------------------------------------------------
------
ModuleNotFoundError                       Traceback (most recent call
last)
~\AppData\Local\Temp\ipykernel_8848\3245841198.py in <module>
      5 #iv)     Update price of book "Python programming" as 1000
      6 #v)      Display all books in the order of publication Year
----> 7 from pymongo import MongoClient
      8
      9 client = MongoClient('mongodb://localhost:27017/')

ModuleNotFoundError: No module named 'pymongo'
```

In [ ]:

```python
#51. Write a python program to connect with MongoDB Database. Crea
#a. display all the documents in the collection restaurants
#b. display the fields restaurant_id, name, establishment_year and
#c. find the restaurants who achieved a score more than 90.
#d. arrange the name of the restaurants in ascending order
#e. Update restaurant score for the establishment year 2019

from pymongo import MongoClient

# Connect to MongoDB server (Assuming MongoDB is running locally)
client = MongoClient('mongodb://localhost:27017/')

# Create or select the database
db = client['RestaurantDB']

# Create the 'restaurants' collection
collection = db['restaurants']

# Sample data to insert into the collection (Assumed struct   ure)
restaurants_data = [
    {"restaurant_id": "R001", "name": "The Gourmet Kitchen", "esta

{"restaurant_id": "R002", "name": "Spicy Delights", "establishment_
    {"restaurant_id": "R003", "name": "Sushi World", "establishmen
    {"restaurant_id": "R004", "name": "Burgers and Fries", "establ
    {"restaurant_id": "R005", "name": "Tacos and More", "establish
]

# Insert data into the collection
collection.insert_many(restaurants_data)

# a) Display all the documents in the collection
print("All restaurants in the collection:")
for restaurant in collection.find():
    print(restaurant)

# b) Display the fields restaurant_id, name, establishment_year, a
print("\nRestaurant details (id, name, establishment_year, cuisine
for restaurant in collection.find({}, {"restaurant_id": 1, "name":
    print(restaurant)

# c) Find restaurants with a score more than 90
print("\nRestaurants with a score greater than 90:")
for restaurant in collection.find({"score": {"$gt": 90}}):
    print(restaurant)

# d) Arrange the name of the restaurants in ascending order
print("\nRestaurants arranged by name (ascending order):")
for restaurant in collection.find().sort("name", 1):  # 1 for asce
    print(restaurant["name"])

# e) Update the score of restaurants established in 2019
collection.update_many({"establishment_year": 2019}, {"$set": {"sc
print("\nUpdated scores for restaurants established in 2019:")
for restaurant in collection.find({"establishment_year": 2019}):
    print(restaurant)

# Close the connection
client.close()
```

```
60
```

In [ ]:

```python
#52 Write a python program to connect with MongoDB Database. Creat
#a. Get all documents
#b. Get all document with director set to “Raj Kapoor”
#c. get all documents where actors include "Amitabh Bachchan"
#d. get all movies released in the 90s
#e. get all movies released before the year 2000 or after 2010
#f. Update some documents by adding some extra fields
#g. Delete movie “movie_name”

from pymongo import MongoClient
from datetime import datetime

# Connect to MongoDB server (Assuming MongoDB is running locally)
client = MongoClient('mongodb://localhost:27017/')

# Create or select the database
db = client['MovieDB']

# Create or select the 'movies' collection
collection = db['movies']

# Sample movie data (for illustration)
movies_data = [
    {"title": "Sholay", "writer": "Salim-Javed", "year": 1975, "ac
    {"title": "Kabhi Kabhie", "writer": "Yes, it’s a love story",
    {"title": "Mera Naam Joker", "writer": "Raj Kapoor", "year": 1
    {"title": "Dilwale Dulhania Le Jayenge", "writer": "Aditya Cho
    {"title": "Lagaan", "writer": "Ashutosh Gowariker", "year": 20
    {"title": "3 Idiots", "writer": "Chetan Bhagat", "year": 2009,
    {"title": "Barfi!", "writer": "Anurag Basu", "year": 2012, "ac
    {"title": "Dabangg", "writer": "Abhinav Kashyap", "year": 2010
]

# Insert sample movies data into the collection
collection.insert_many(movies_data)

# a) Get all documents
print("All movies:")
for movie in collection.find():
    print(movie)

# b) Get all documents where director is "Raj Kapoor"
print("\nMovies directed by Raj Kapoor:")
for movie in collection.find({"director": "Raj Kapoor"}):
    print(movie)

# c) Get all documents where actors include "Amitabh Bachchan"
print("\nMovies featuring Amitabh Bachchan:")
for movie in collection.find({"actors": "Amitabh Bachchan"}):
    print(movie)

# d) Get all movies released in the 90s
print("\nMovies released in the 90s:")
for movie in collection.find({"year": {"$gte": 1990, "$lt": 2000}}
    print(movie)

# e) Get all movies released before 2000 or after 2010
print("\nMovies released before 2000 or after 2010:")
for movie in collection.find({"year": {"$lt": 2000, "$gt": 2010}})
    print(movie)

```

```python
# f) Update some documents by adding extra fields (e.g., 'genre' a
collection.update_many({}, {"$set": {"genre": "Drama", "rating": 8
print("\nUpdated movies with 'genre' and 'rating':")
for movie in collection.find():
    print(movie)

# g) Delete a movie by its name (e.g., "Sholay")
collection.delete_one({"title": "Sholay"})
print("\nDeleted movie 'Sholay' from the collection:")
for movie in collection.find():
    print(movie)

# Close the connection
client.close()

```