1. **Write a python program to complete the given task using string methods:**
    a. **Find the number of characters in each product name and store it in a list.**
    b. **Convert all product names to uppercase.**
    c. **Find the number of times the word "sale" appears in the product descriptions.**
    d. **Replace all occurrences of the word "red" with "green" in the product descriptions.**
    e. **You need to perform the following tasks using string methods:**
    f. **Find the number of characters in each product name and store it in a list.**
    g. **Convert all product names to uppercase.**
    h. **Find the number of times the word "sale" appears in the product descriptions.**
    i. **Replace all occurrences of the word "red" with "green" in the product descriptions.**

```
# Sample data
product_names = ["Red Shirt", "Blue Jeans", "Green Hat"]
product_descriptions = ["Best red shirt on sale", "Stylish jeans on sale", "Comfortable green hat"]

# 1. Find the number of characters in each product name and store it in a list
char_counts = [len(name) for name in product_names]

# 2. Convert all product names to uppercase
uppercase_names = [name.upper() for name in product_names]

# 3. Find the number of times the word "sale" appears in the product descriptions
sale_count = sum(desc.lower().count("sale") for desc in product_descriptions)

# 4. Replace all occurrences of the word "red" with "green" in the product descriptions
updated_descriptions = [desc.replace("red", "green") for desc in product_descriptions]

# Output results
print("Character counts:", char_counts)
print("Uppercase names:", uppercase_names)
print("Sale count:", sale_count)
print("Updated descriptions:", updated_descriptions)
```

2. **Write a python program to complete the given task using List and its methods**

a. **Add five more products to the list and sort them in alphabetical order.**

b. **Create a new list with the prices of all the products and find the average price.**
c. **Count the number of products whose price is above 500.**
d. **Remove the last product from the list and add a new product at the beginning.**
e. **Find the index of the product with the maximum price and replace its price with a new value.**

```
# Initial product and price lists
products = ["Shirt", "Jeans", "Hat", "Jacket", "Shoes"]
prices = [300, 450, 700, 1000, 200]

# 1. Add five more products and sort them in alphabetical order
more_products = ["T-shirt", "Sweater", "Blazer", "Sneakers", "Coat"]
products.extend(more_products)  # Adding five more products
products.sort()  # Sorting alphabetically

# 2. Create a new list with the prices of all the products and find the average price
new_prices = [300, 450, 700, 1000, 200, 1500, 800, 600, 400, 1200]
average_price = sum(new_prices) / len(new_prices)  # Calculate average price

# 3. Count the number of products whose price is above 500
above_500_count = len([price for price in new_prices if price > 500])

# 4. Remove the last product from the list and add a new product at the beginning
products.pop()  # Removing the last product
products.insert(0, "Cap")  # Adding a new product at the beginning

# 5. Find the index of the product with the maximum price and replace its price with a new value
max_price_index = new_prices.index(max(new_prices))  # Find index of max price
new_prices[max_price_index] = 900  # Replace the max price with a new value

# Output results
print("Sorted products:", products)
print("Average price:", average_price)
print("Products with price above 500:", above_500_count)
print("Updated prices:", new_prices)
```

3. Write a python program to complete the given task using Dictionary and its methods

a. **Add three more students to the dictionary with their respective details.**
b. **Display all the keys in the dictionary.**
c. **Display all the values in the dictionary.**
d. **Update the address of one of the students in the dictionary.**

e. **Remove one of the students from the dictionary.**
f. **Sort the dictionary by keys and display the sorted dictionary.**
g. **Sort the dictionary by values and display the sorted dictionary.**
h. **Find the length of the dictionary.**
i. **Check if a key is present in the dictionary or not.**
j. **Clear the dictionary and display it.**

```
# Initial dictionary of students with their details
students = {
    "John": {"age": 20, "address": "123 Main St"},
    "Alice": {"age": 22, "address": "456 Oak St"},
    "Bob": {"age": 19, "address": "789 Pine St"}
}

# 1. Add three more students to the dictionary
students["Eve"] = {"age": 21, "address": "321 Maple St"}
students["Mike"] = {"age": 23, "address": "654 Elm St"}
students["Sophia"] = {"age": 20, "address": "987 Cedar St"}

# 2. Display all the keys in the dictionary
print("Keys:", students.keys())

# 3. Display all the values in the dictionary
print("Values:", students.values())

# 4. Update the address of one of the students
students["Alice"]["address"] = "999 Birch St"  # Updating Alice's address

# 5. Remove one of the students from the dictionary
students.pop("John")  # Removing John

# 6. Sort the dictionary by keys and display the sorted dictionary
sorted_by_keys = dict(sorted(students.items()))  # Sort by keys
print("Sorted by keys:", sorted_by_keys)

# 7. Sort the dictionary by values (by 'age') and display the sorted dictionary
sorted_by_values = dict(sorted(students.items(), key=lambda x: x[1]["age"]))  # Sort by values (age)
print("Sorted by values (age):", sorted_by_values)

# 8. Find the length of the dictionary
print("Length of dictionary:", len(students))

# 9. Check if a key is present in the dictionary
key_to_check = "Bob"
is_present = key_to_check in students
print(f"Is '{key_to_check}' present in the dictionary?", is_present)

# 10. Clear the dictionary and display it
students.clear()
print("Cleared dictionary:", students)
```

4 **Create a tuple named products with the following information: product name, product ID, product price, and product quantity. Use the appropriate tuple method to perform the following tasks:**

a. **Display the entire tuple.**
b. **Display the length of the tuple.**
c. **Access the second element of the tuple.**
d. **Find the index of a particular element in the tuple.**
e. **Count the number of occurrences of a particular element in the tuple.**
f. **Concatenate two tuples.**
g. **Convert the tuple to a list.**
h. **Check if a particular element is present in the tuple or not.**
i. **Display the maximum value in the tuple.**
j. **Display the minimum value in the tuple.**

```
# Creating a tuple named products with product name, ID, price, and quantity
products = ("Laptop", 101, 75000, 5)

# 1. Display the entire tuple
print("Products tuple:", products)

# 2. Display the length of the tuple
print("Length of tuple:", len(products))

# 3. Access the second element of the tuple (product ID)
print("Second element (Product ID):", products[1])

# 4. Find the index of a particular element in the tuple (e.g., 75000)
index_of_element = products.index(75000)
print("Index of 75000 (Product Price):", index_of_element)

# 5. Count the number of occurrences of a particular element (e.g., 5)
occurrences_of_element = products.count(5)
print("Occurrences of 5 (Product Quantity):", occurrences_of_element)

# 6. Concatenate two tuples
additional_products = ("Mouse", 102, 1500, 10)
concatenated_tuple = products + additional_products
print("Concatenated tuple:", concatenated_tuple)

# 7. Convert the tuple to a list
products_list = list(products)
print("Tuple converted to list:", products_list)

# 8. Check if a particular element is present in the tuple (e.g., 'Laptop')
element_present = "Laptop" in products
print("Is 'Laptop' present in the tuple?", element_present)

# 9. Display the maximum value in the tuple (numeric values only)
numeric_values = [x for x in products if isinstance(x, (int, float))]
max_value = max(numeric_values)
```

```
print("Maximum value in the tuple:", max_value)

# 10. Display the minimum value in the tuple (numeric values only)
min_value = min(numeric_values)
print("Minimum value in the tuple:", min_value)
```

5. **Write a program that reads a file named input.txt, containing multiple lines of text. For each line in the file, the program should count the number of vowels (a, e, i, o, u) in the line and write the result to a new file named output.txt. Each line in the output file should be in the following format: "Line X: Y vowels", where X is the line number (starting from 1) and Y is the number of vowels in the line.**

```
# Function to count vowels in a given line
def count_vowels(line):
    vowels = "aeiouAEIOU"
    return sum(1 for char in line if char in vowels)

# Read from input.txt and write to output.txt
def process_file():
    try:
        # Open input.txt for reading and output.txt for writing
        with open('input.txt', 'r') as infile, open('output.txt', 'w') as outfile:
            line_number = 1

            # Read each line from input.txt
            for line in infile:
                # Count the number of vowels in the current line
                vowel_count = count_vowels(line)

                # Write the result to output.txt
                outfile.write(f"Line {line_number}: {vowel_count} vowels\n")

                # Move to the next line
                line_number += 1

        print("Vowel counting completed and written to output.txt.")

    except FileNotFoundError:
        print("Error: input.txt file not found.")

# Run the program
process_file()
```

6. Create a Python program to use a set to store the items in your inventory. Your program should perform the following tasks:

a. **Use the add() method to add items to the inventory set.**
b. **Use the remove() method to remove an item from the inventory set.**

c. **Use the clear() method to empty the inventory set.**
d. **Use the update() method to add multiple items to the inventory set.**
e. **Use the intersection() method to find the common items between two inventory sets.**
f. **Use the difference() method to find the items that are in one inventory set but not in the other.**
g. **Use the union() method to combine two inventory sets into one.**
h. **Use the copy() method to make a copy of the inventory set.**
i. **Use the pop() method to remove and return an arbitrary item from the inventory set.**
j. **Use the discard() method to remove an item from the inventory set if it is present, but do not raise an error if the item is not present.**

```
# Initialize an empty set for the inventory
inventory = set()

# 1. Use the add() method to add items to the inventory set
inventory.add("sword")
inventory.add("shield")
inventory.add("potion")
print("Inventory after adding items:", inventory)

# 2. Use the remove() method to remove an item from the inventory set
inventory.remove("potion")
print("Inventory after removing 'potion':", inventory)

# 3. Use the clear() method to empty the inventory set
inventory.clear()
print("Inventory after clearing:", inventory)

# 4. Use the update() method to add multiple items to the inventory set
inventory.update(["armor", "boots", "helmet"])
print("Inventory after adding multiple items:", inventory)

# Creating another inventory set for comparison
another_inventory = {"helmet", "ring", "amulet"}

# 5. Use the intersection() method to find common items between two inventory sets
common_items = inventory.intersection(another_inventory)
print("Common items between inventories:", common_items)

# 6. Use the difference() method to find items in one set but not in the other
difference_items = inventory.difference(another_inventory)
print("Items in inventory but not in another inventory:", difference_items)

# 7. Use the union() method to combine two inventory sets
combined_inventory = inventory.union(another_inventory)
print("Combined inventory:", combined_inventory)

# 8. Use the copy() method to make a copy of the inventory set
inventory_copy = inventory.copy()
```

```
print("Copied inventory:", inventory_copy)

# 9. Use the pop() method to remove and return an arbitrary item from the inventory set
popped_item = inventory.pop()
print("Popped item:", popped_item)
print("Inventory after popping an item:", inventory)

# 10. Use the discard() method to remove an item if it exists without raising an error
inventory.discard("boots")  # This will remove 'boots' if it exists
inventory.discard("spear")  # This won't raise an error if 'spear' doesn't exist
print("Inventory after discarding 'boots' and 'spear':", inventory)
```

7. **Write a Python program to take input from the user for their grades in five subjects and calculate their overall grade based on the following criteria:**

a. **Each subject has a weightage of 20% towards the overall grade.**
b. **If the overall grade is less than 40%, the student fails.**
c. **If the overall grade is between 40% and 60%, the student gets a C grade.**
d. **If the overall grade is between 60% and 80%, the student gets a B grade.**
e. **If the overall grade is above 80%, the student gets an**

A grade. Display the overall grade along with the grade letter to the user.

```
# Initialize an empty set for the inventory
inventory = set()

# 1. Use the add() method to add items to the inventory set
inventory.add("sword")
inventory.add("shield")
inventory.add("potion")
print("Inventory after adding items:", inventory)

# 2. Use the remove() method to remove an item from the inventory set
inventory.remove("potion")
print("Inventory after removing 'potion':", inventory)

# 3. Use the clear() method to empty the inventory set
inventory.clear()
print("Inventory after clearing:", inventory)

# 4. Use the update() method to add multiple items to the inventory set
inventory.update(["armor", "boots", "helmet"])
print("Inventory after adding multiple items:", inventory)

# Creating another inventory set for comparison
another_inventory = {"helmet", "ring", "amulet"}

# 5. Use the intersection() method to find common items between two inventory sets
common_items = inventory.intersection(another_inventory)
print("Common items between inventories:", common_items)
```

```
# 6. Use the difference() method to find items in one set but not in the other
difference_items = inventory.difference(another_inventory)
print("Items in inventory but not in another inventory:", difference_items)

# 7. Use the union() method to combine two inventory sets
combined_inventory = inventory.union(another_inventory)
print("Combined inventory:", combined_inventory)

# 8. Use the copy() method to make a copy of the inventory set
inventory_copy = inventory.copy()
print("Copied inventory:", inventory_copy)

# 9. Use the pop() method to remove and return an arbitrary item from the inventory set
popped_item = inventory.pop()
print("Popped item:", popped_item)
print("Inventory after popping an item:", inventory)

# 10. Use the discard() method to remove an item if it exists without raising an error
inventory.discard("boots")  # This will remove 'boots' if it exists
inventory.discard("spear")  # This won't raise an error if 'spear' doesn't exist
print("Inventory after discarding 'boots' and 'spear':", inventory)
```

8**.Write a Python program that uses regular expressions to extract this information from the text data.The text data consists of a long string that includes multiple phone numbers, email addresses, and postal addresses. Your program should extract the following information from the text data:**

a. **All phone numbers, formatted as (+XX) XXXXXXXXXX**
b. **All email addresses, which follow the standard format of username@domain.com**
c. **All postal addresses, which consist of a street address, city, state, and ZIP code, with each part separated by a comma**

```
import re

# Sample text data containing phone numbers, emails, and postal addresses
text_data = """
Contact us at (+91) 9876543210 or (+44) 1234567890.
Send your queries to support@example.com or hr@company.co.uk.
Visit us at 123 Elm Street, Springfield, IL, 62704 or 456 Maple Avenue, Chicago, IL, 60605.
"""

# 1. Extract all phone numbers formatted as (+XX) XXXXXXXXXX
phone_pattern = r"\(\+\d{2}\)\s\d{10}"
phone_numbers = re.findall(phone_pattern, text_data)
print("Phone numbers:", phone_numbers)

# 2. Extract all email addresses
email_pattern = r"[a-zAZ0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}"
```

```
emails = re.findall(email_pattern, text_data)
print("Email addresses:", emails)

# 3. Extract all postal addresses formatted as street address, city, state, ZIP code
address_pattern = r"\d{1,4}\s[\w\s]+\s(?:Street|Avenue|Rd|Blvd),\s\w+,\s\w{2},\s\d{5}"
addresses = re.findall(address_pattern, text_data)
print("Postal addresses:", addresses)
```

-

9. Write a python program to demonstrate the concept of keyword argument function and default argument function.

```
# Function with default arguments
def greet(name="Guest", age=25):
    """
    This function takes name and age as arguments.
    If no value is provided for either, the default values "Guest" and 25 are used.
    """
    print(f"Hello, {name}! You are {age} years old.")

# Function with keyword arguments
def calculate_total(price, tax_rate=0.1, discount=0):
    """
    This function calculates the total price after applying tax and discount.
    Keyword arguments can be used to specify tax_rate and discount.
    """
    total = price + (price * tax_rate) - discount
    print(f"Total price: ${total:.2f}")

# Demonstrating default arguments
print("Demonstrating default arguments:")
greet()  # No arguments passed, will use default values
greet("Alice")  # Only 'name' passed, age will use default
greet("Bob", 30)  # Both arguments passed

# Demonstrating keyword arguments
print("\nDemonstrating keyword arguments:")
calculate_total(100)  # Only price passed, tax_rate and discount will use defaults
calculate_total(100, tax_rate=0.15)  # Custom tax rate, default discount
calculate_total(100, tax_rate=0.15, discount=10)  # Custom tax rate and discount
```

10. Write a Python program to create a shopping cart for a customer. The program should have at least three functions:

a. **add_item(item_name, item_price): This function should add an item and its price to the shopping cart.**

b. **remove_item(item_name): This function should remove an item from the shopping cart.**

c. view_cart(): This function should display the items in the shopping cart and the total cost.

```
# Initialize an empty shopping cart (a dictionary to store item names and prices)
shopping_cart = {}

# Function to add an item to the shopping cart
def add_item(item_name, item_price):
    if item_name in shopping_cart:
        print(f"'{item_name}' is already in the cart.")
    else:
        shopping_cart[item_name] = item_price
        print(f"'{item_name}' has been added to the cart.")

# Function to remove an item from the shopping cart
def remove_item(item_name):
    if item_name in shopping_cart:
        del shopping_cart[item_name]
        print(f"'{item_name}' has been removed from the cart.")
    else:
        print(f"'{item_name}' is not in the cart.")

# Function to view the shopping cart and total cost
def view_cart():
    if not shopping_cart:
        print("Your cart is empty.")
    else:
        print("\nYour Shopping Cart:")
        total_cost = 0
        for item_name, item_price in shopping_cart.items():
            print(f"{item_name}: ${item_price:.2f}")
            total_cost += item_price
        print(f"\nTotal Cost: ${total_cost:.2f}")

# Example usage of the shopping cart functions
add_item("Laptop", 899.99)
add_item("Mouse", 19.99)
add_item("Keyboard", 49.99)

view_cart()

remove_item("Mouse")
view_cart()

add_item("Monitor", 199.99)
view_cart()
```