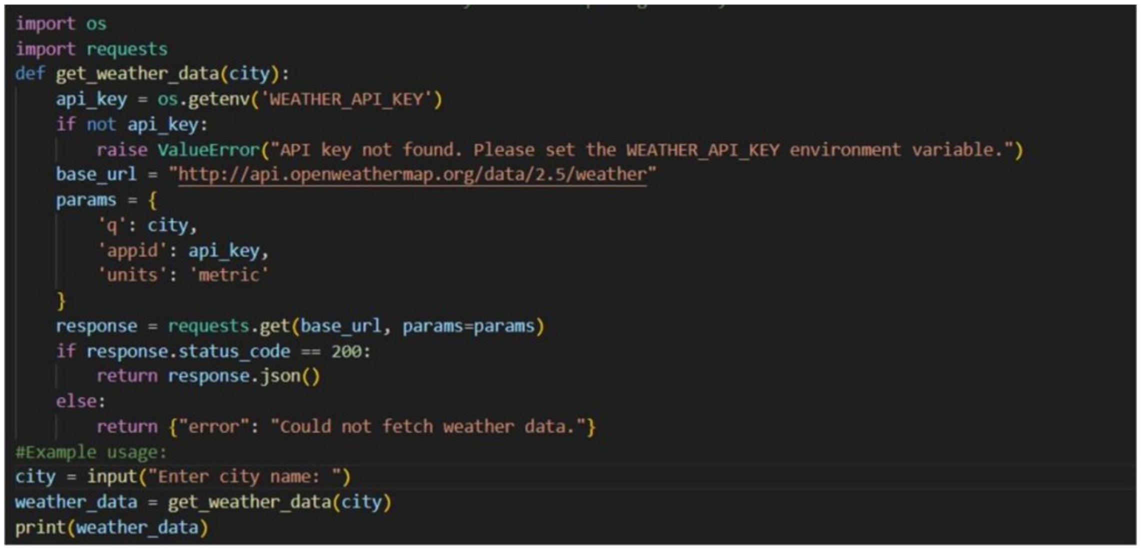
LAB ASSIGNMENT – 5.1

Name : Dasi Mahesh

Roll No: 2303A54021

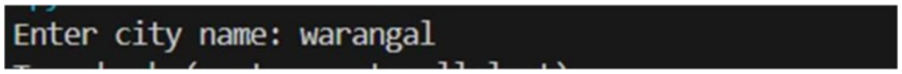
Batch – 47

1) Generate code to fetch weather data securely without exposing API keys in the code.



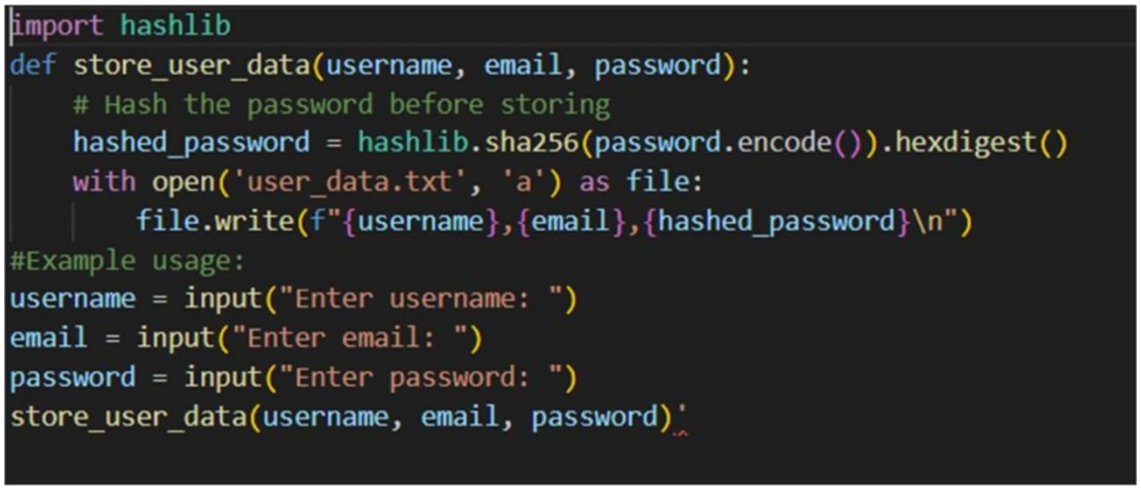
```python
import os
import requests
def get_weather_data(city):
    api_key = os.getenv('WEATHER_API_KEY')
    if not api_key:
        raise ValueError("API key not found. Please set the WEATHER_API_KEY environment variable.")
    base_url = "http://api.openweathermap.org/data/2.5/weather"
    params = {
        'q': city,
        'appid': api_key,
        'units': 'metric'
    }
    response = requests.get(base_url, params=params)
    if response.status_code == 200:
        return response.json()
    else:
        return {"error": "Could not fetch weather data."}
#Example usage:
city = input("Enter city name: ")
weather_data = get_weather_data(city)
print(weather_data)
```

Output:

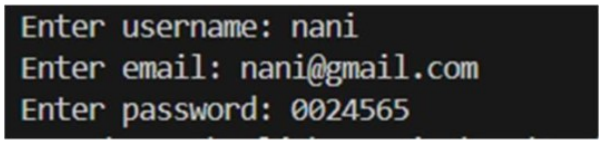


```
Enter city name: warangal
```

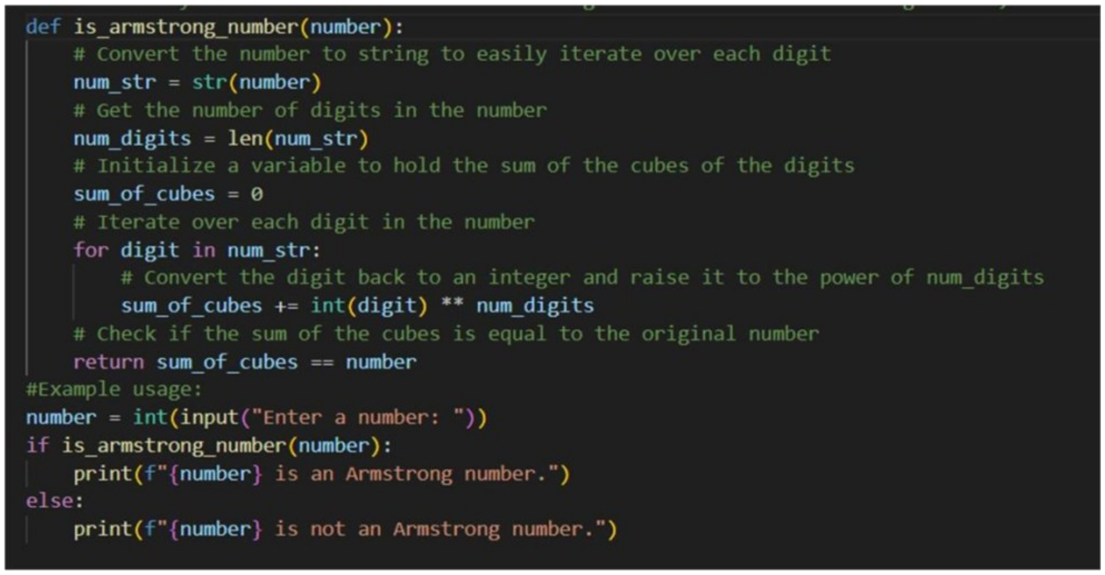
2) Generate a Python program that takes user name, email, and password as input, stores the data in a file, iden fies the security risk of storing passwords in plain text, and provides a revised version that securely stores passwords using hashing (hashlib or bcrypt).


```
import hashlib
def store_user_data(username, email, password):
    # Hash the password before storing
    hashed_password = hashlib.sha256(password.encode()).hexdigest()
    with open('user_data.txt', 'a') as file:
        file.write(f"{username},{email},{hashed_password}\n")
#Example usage:
username = input("Enter username: ")
email = input("Enter email: ")
password = input("Enter password: ")
store_user_data(username, email, password)'
```

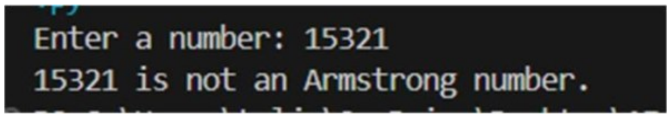
Output:


```
Enter username: nani
Enter email: nani@gmail.com
Enter password: 0024565
```

3) Generate a Python func on to check whether a given number is an Armstrong number, include clear comments in the code, explain the logic line-by-line, and ensure the explana on correctly matches the actual func onality of the code in a simple and easy-to-understand way.


```
def is_armstrong_number(number):
    # Convert the number to string to easily iterate over each digit
    num_str = str(number)
    # Get the number of digits in the number
    num_digits = len(num_str)
    # Initialize a variable to hold the sum of the cubes of the digits
    sum_of_cubes = 0
    # Iterate over each digit in the number
    for digit in num_str:
        # Convert the digit back to an integer and raise it to the power of num_digits
        sum_of_cubes += int(digit) ** num_digits
    # Check if the sum of the cubes is equal to the original number
    return sum_of_cubes == number
#Example usage:
number = int(input("Enter a number: "))
if is_armstrong_number(number):
    print(f"{number} is an Armstrong number.")
else:
    print(f"{number} is not an Armstrong number.")
```

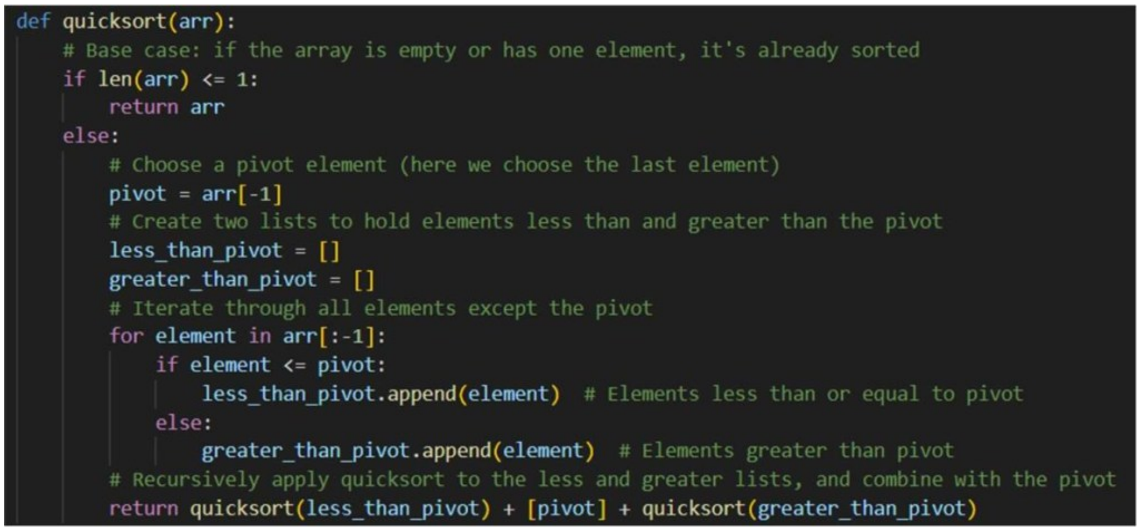
Output:



```
Enter a number: 15321
15321 is not an Armstrong number.
```

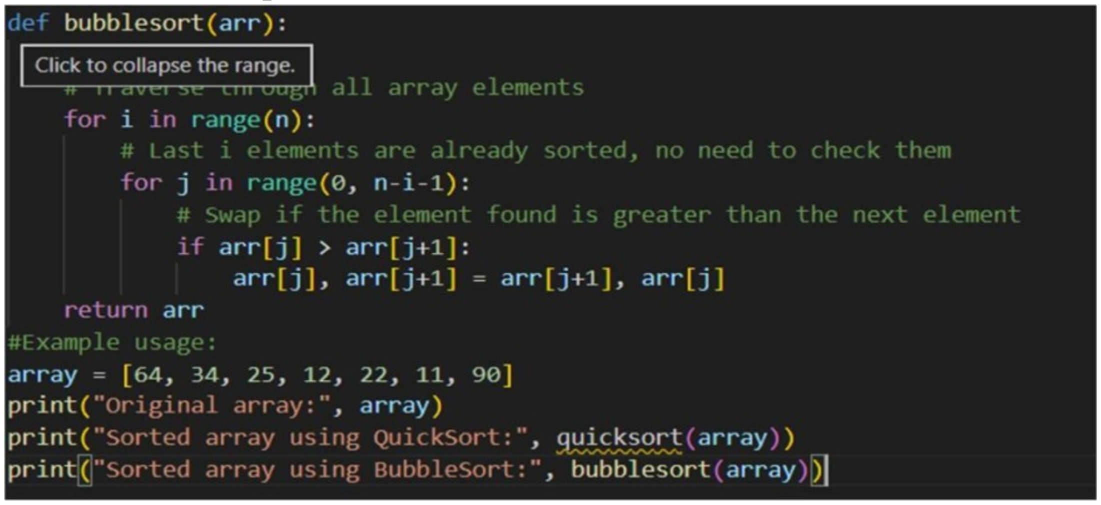
4) Generate Python code for QuickSort and BubbleSort, and include comments explaining step-by-step how each works and where they differ

□ Quick Sort implementa on



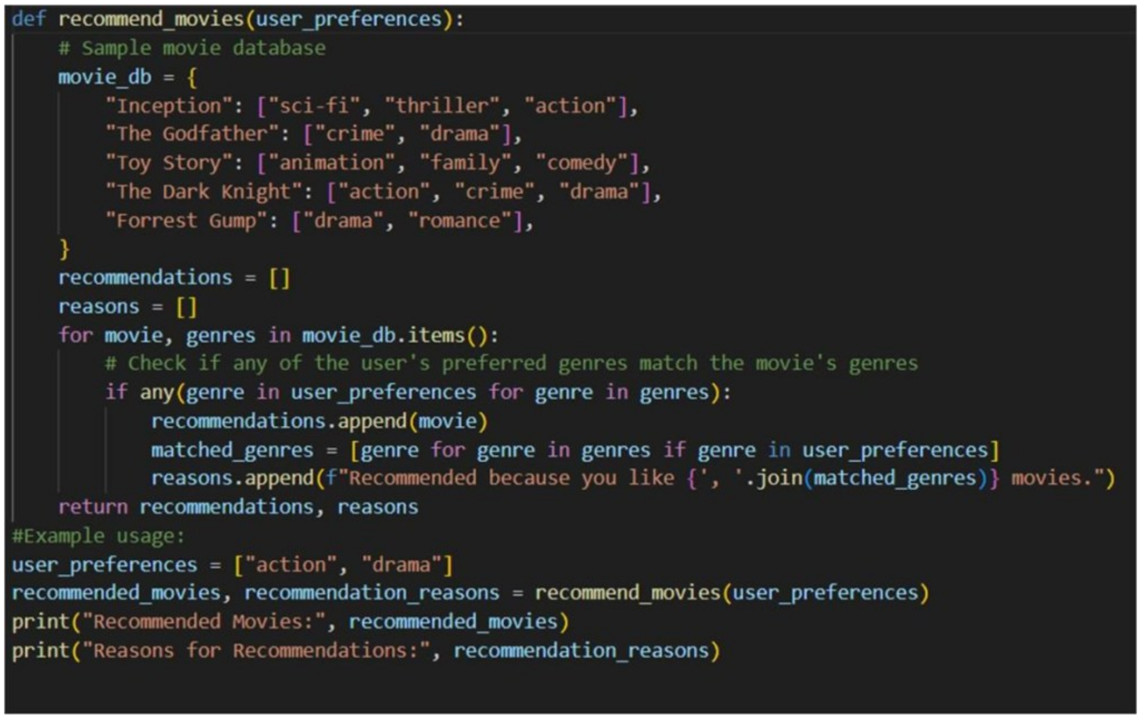
```python
def quicksort(arr):
    # Base case: if the array is empty or has one element, it's already sorted
    if len(arr) <= 1:
        return arr
    else:
        # Choose a pivot element (here we choose the last element)
        pivot = arr[-1]
        # Create two lists to hold elements less than and greater than the pivot
        less_than_pivot = []
        greater_than_pivot = []
        # Iterate through all elements except the pivot
        for element in arr[:-1]:
            if element <= pivot:
                less_than_pivot.append(element)  # Elements less than or equal to pivot
            else:
                greater_than_pivot.append(element)  # Elements greater than pivot
        # Recursively apply quicksort to the less and greater lists, and combine with the pivot
        return quicksort(less_than_pivot) + [pivot] + quicksort(greater_than_pivot)
```

□ Bubble Sort implementa on



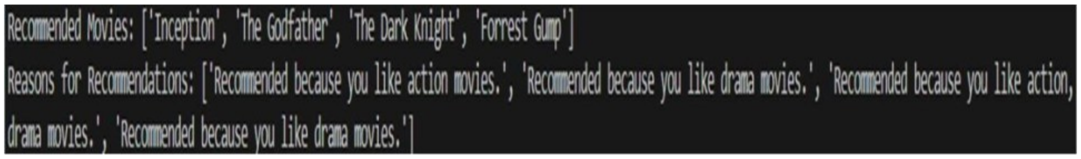
```python
def bubblesort(arr):
    # Traverse through all array elements
    for i in range(n):
        # Last i elements are already sorted, no need to check them
        for j in range(0, n-i-1):
            # Swap if the element found is greater than the next element
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
    return arr
#Example usage:
array = [64, 34, 25, 12, 22, 11, 90]
print("Original array:", array)
print("Sorted array using QuickSort:", quicksort(array))
print("Sorted array using BubbleSort:", bubblesort(array))
```

5) Generate a recommenda on system that also provides reasons for each sugges on.



```python
def recommend_movies(user_preferences):
    # Sample movie database
    movie_db = {
        "Inception": ["sci-fi", "thriller", "action"],
        "The Godfather": ["crime", "drama"],
        "Toy Story": ["animation", "family", "comedy"],
        "The Dark Knight": ["action", "crime", "drama"],
        "Forrest Gump": ["drama", "romance"],
    }
    recommendations = []
    reasons = []
    for movie, genres in movie_db.items():
        # Check if any of the user's preferred genres match the movie's genres
        if any(genre in user_preferences for genre in genres):
            recommendations.append(movie)
            matched_genres = [genre for genre in genres if genre in user_preferences]
            reasons.append(f"Recommended because you like {', '.join(matched_genres)} movies.")
    return recommendations, reasons
#Example usage:
user_preferences = ["action", "drama"]
recommended_movies, recommendation_reasons = recommend_movies(user_preferences)
print("Recommended Movies:", recommended_movies)
print("Reasons for Recommendations:", recommendation_reasons)
```

Output:



```
Recommended Movies: ['Inception', 'The Godfather', 'The Dark Knight', 'Forrest Gump']
Reasons for Recommendations: ['Recommended because you like action movies.', 'Recommended because you like drama movies.', 'Recommended because you like action,
drama movies.', 'Recommended because you like drama movies.']
```