Lab Assignment-5.1

Syed Abdul Mateen

2303A51921

Batch-12

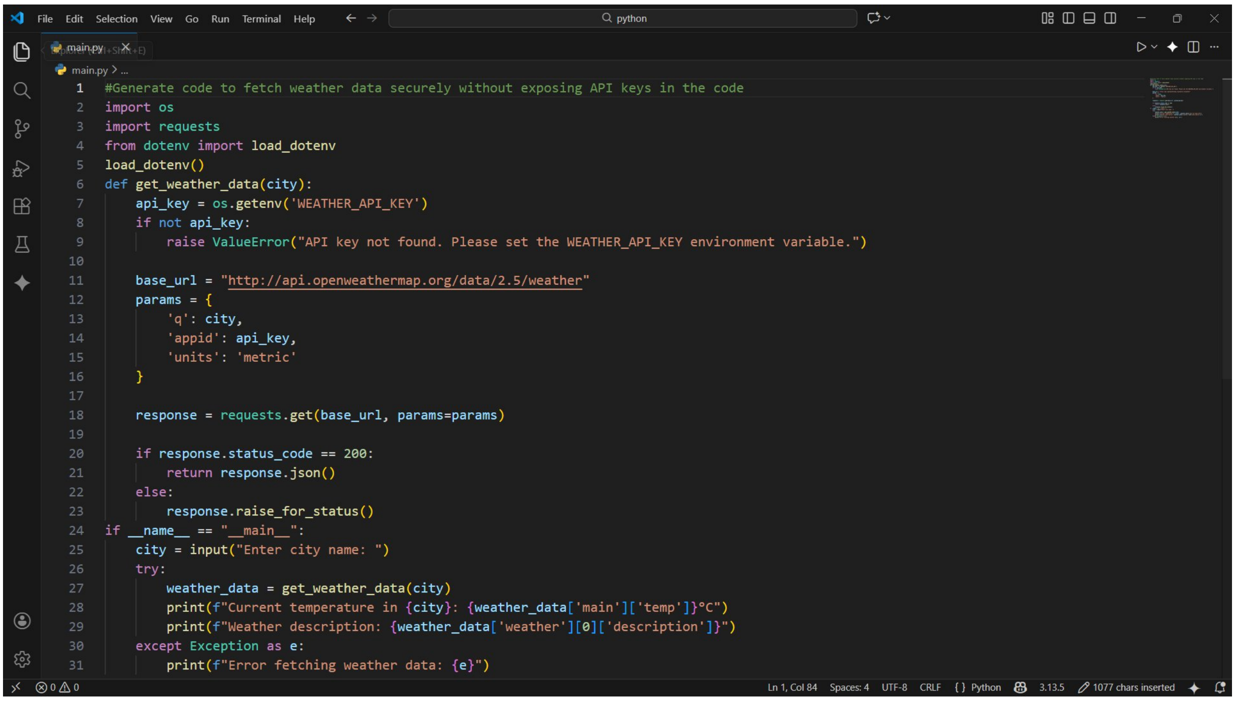
Task Description #1 (Privacy in API Usage):

Task: Use an AI tool to generate a Python program that connects to a weather API.

Prompt:

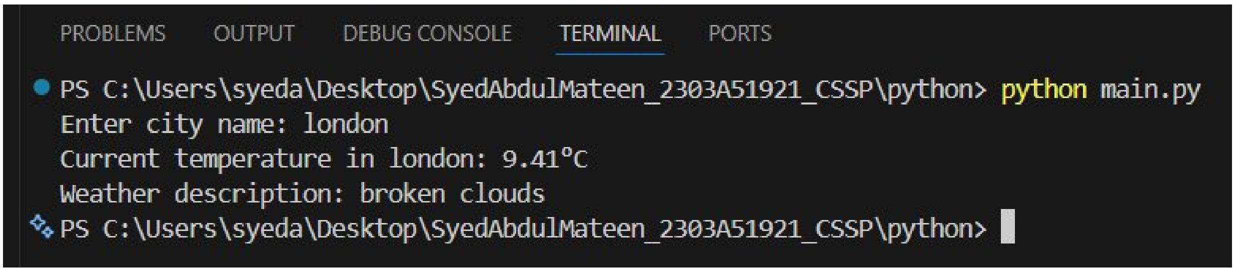
#Generate code to fetch weather data securely without exposing API keys in the code.

Code:



```python
#Generate code to fetch weather data securely without exposing API keys in the code
import os
import requests
from dotenv import load_dotenv
load_dotenv()
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:
        response.raise_for_status()
if __name__ == "__main__":
    city = input("Enter city name: ")
    try:
        weather_data = get_weather_data(city)
        print(f"Current temperature in {city}: {weather_data['main']['temp']}°C")
        print(f"Weather description: {weather_data['weather'][0]['description']}")
    except Exception as e:
        print(f"Error fetching weather data: {e}")
```

Output:



```
PS C:\Users\syeda\Desktop\SyedAbdulMateen_2303A51921_CSSP\python> python main.py
Enter city name: london
Current temperature in london: 9.41°C
Weather description: broken clouds
PS C:\Users\syeda\Desktop\SyedAbdulMateen_2303A51921_CSSP\python>
```

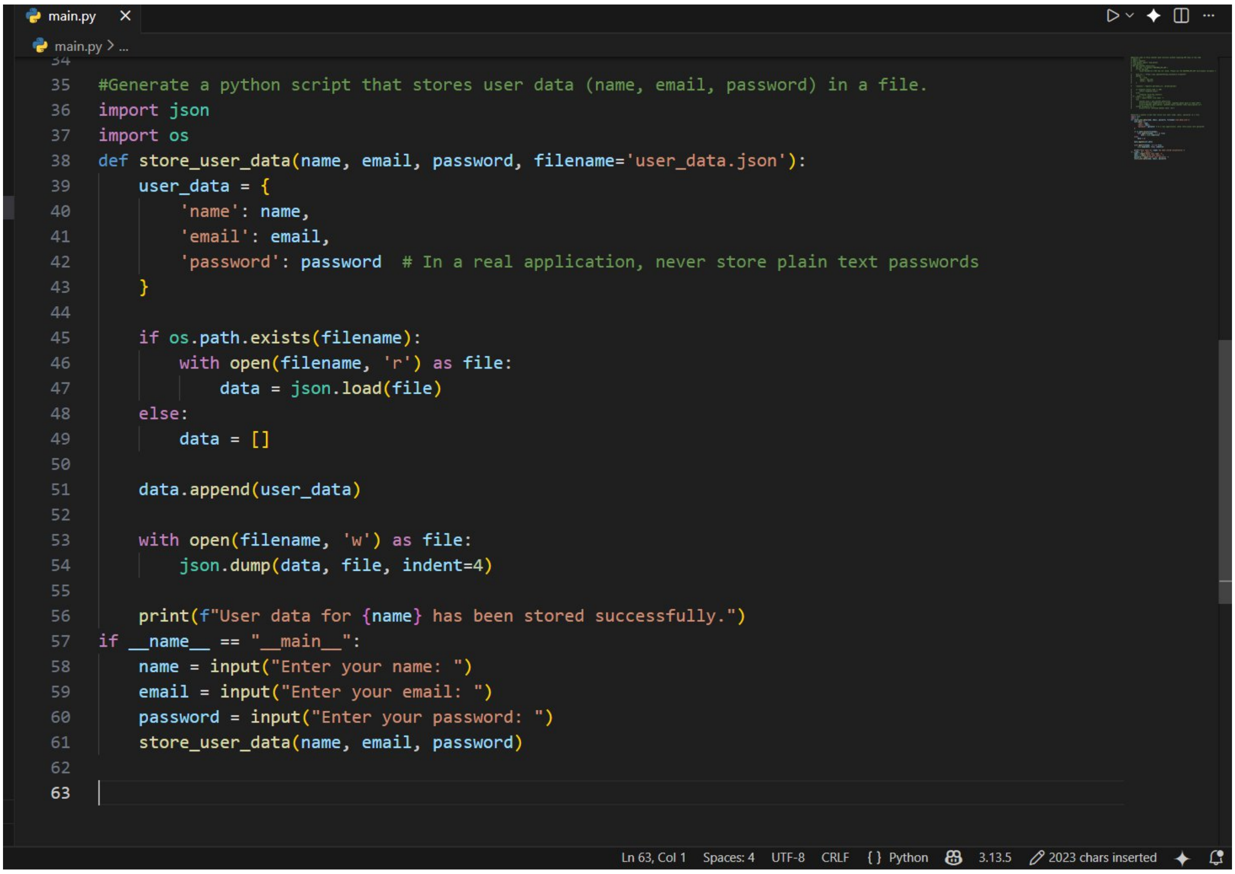
Explanation:

This Python script demonstrates how to securely fetch weather data for a user-specified city using the OpenWeatherMap API. It begins by loading environment variables from a .env file, ensuring the API key is not exposed directly in the code. The get_weather function constructs a request with the city name and API key, sending it to the weather API and returning the result as JSON if successful. The main block prompts the user to enter a city name, retrieves the weather data, and displays the weather description and temperature if found. If the city is not found or an error occurs, an appropriate error message is shown, making the script both secure and user-friendly.

Task Description #2 (Privacy & Security in File Handling)

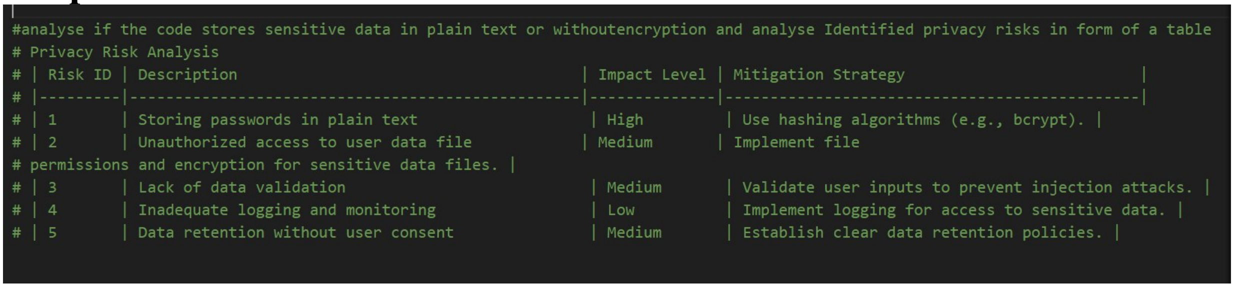
Task: Use an AI tool to generate a Python script that stores user data (name, email, password) in a file.

Prompt:



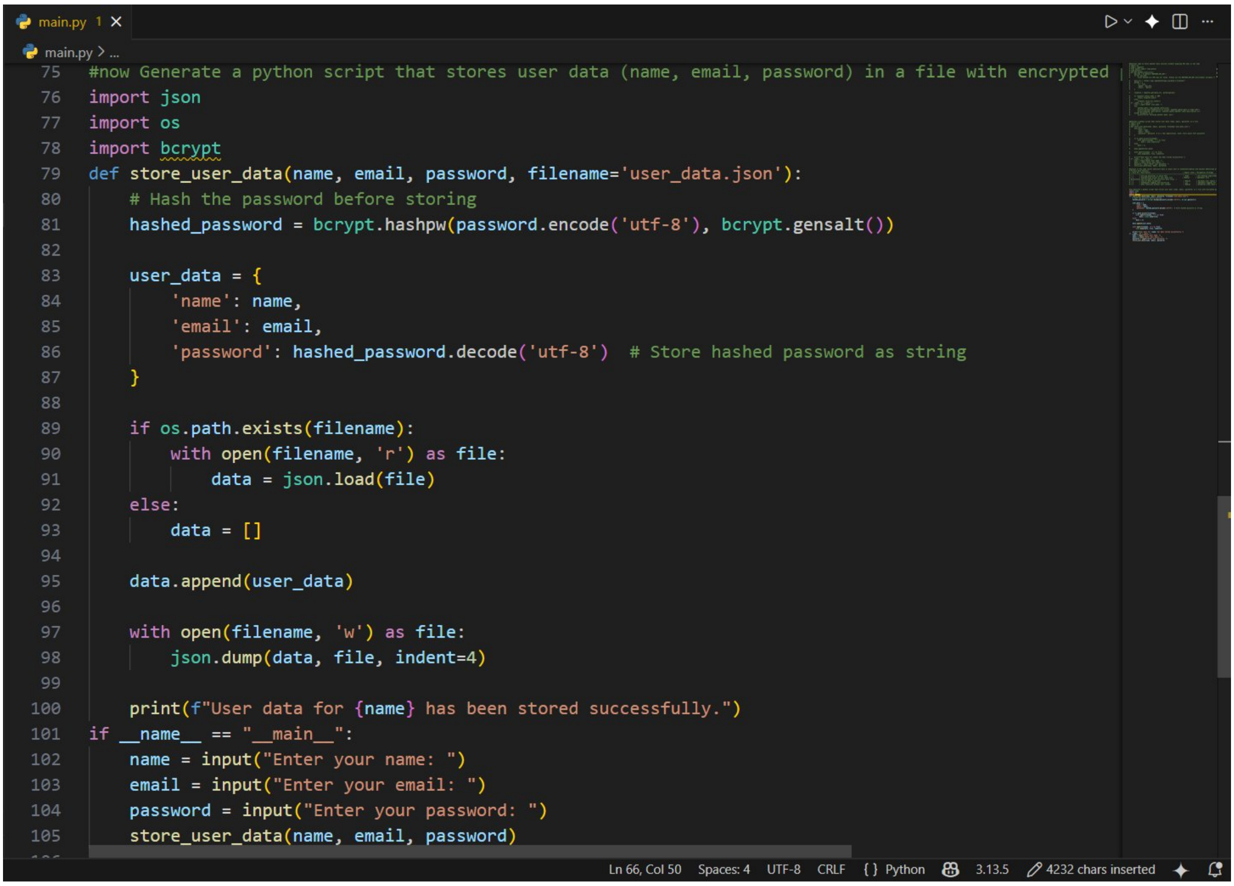
```python
#Generate a python script that stores user data (name, email, password) in a file.
import json
import os
def store_user_data(name, email, password, filename='user_data.json'):
    user_data = {
        'name': name,
        'email': email,
        'password': password  # In a real application, never store plain text passwords
    }

    if os.path.exists(filename):
        with open(filename, 'r') as file:
            data = json.load(file)
    else:
        data = []

    data.append(user_data)

    with open(filename, 'w') as file:
        json.dump(data, file, indent=4)

    print(f"User data for {name} has been stored successfully.")
if __name__ == "__main__":
    name = input("Enter your name: ")
    email = input("Enter your email: ")
    password = input("Enter your password: ")
    store_user_data(name, email, password)

```

Output:



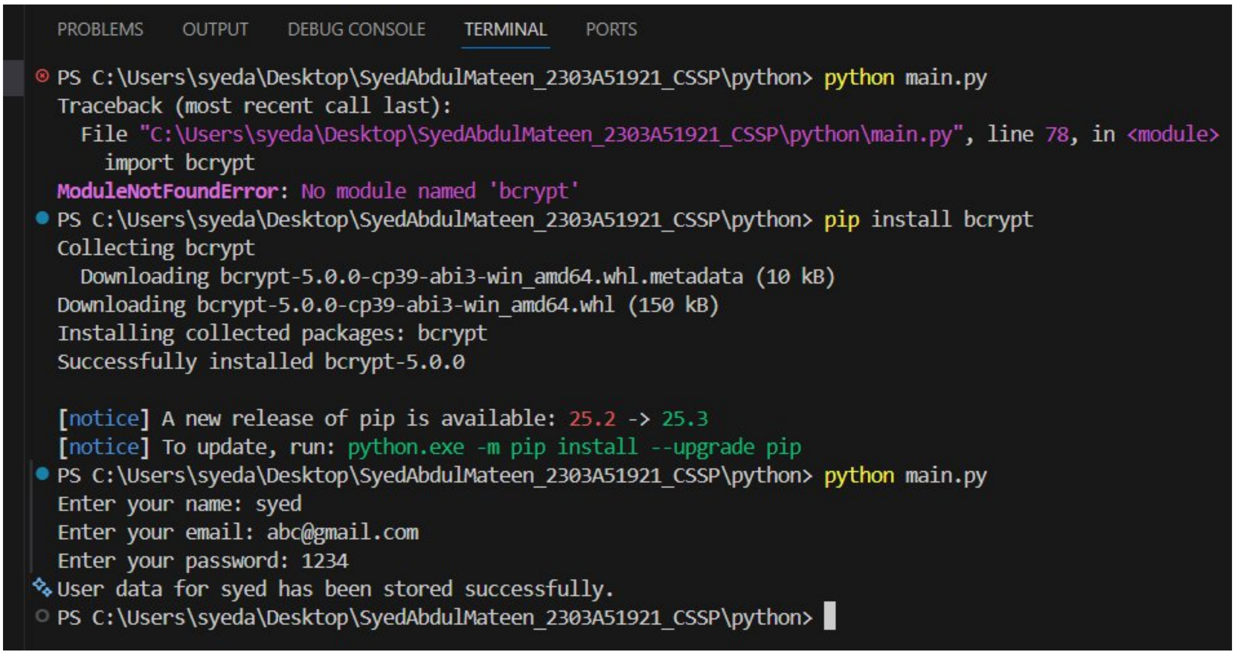
```
#analyse if the code stores sensitive data in plain text or withoutencryption and analyse Identified privacy risks in form of a table
# Privacy Risk Analysis
# | Risk ID | Description                                      | Impact Level | Mitigation Strategy                          |
# |---------|--------------------------------------------------|--------------|----------------------------------------------|
# | 1       | Storing passwords in plain text                   | High         | Use hashing algorithms (e.g., bcrypt). |
# | 2       | Unauthorized access to user data file            | Medium       | Implement file
# permissions and encryption for sensitive data files. |
# | 3       | Lack of data validation                           | Medium       | Validate user inputs to prevent injection attacks. |
# | 4       | Inadequate logging and monitoring                 | Low          | Implement logging for access to sensitive data. |
# | 5       | Data retention without user consent               | Medium       | Establish clear data retention policies. |
```

With Encrypted:

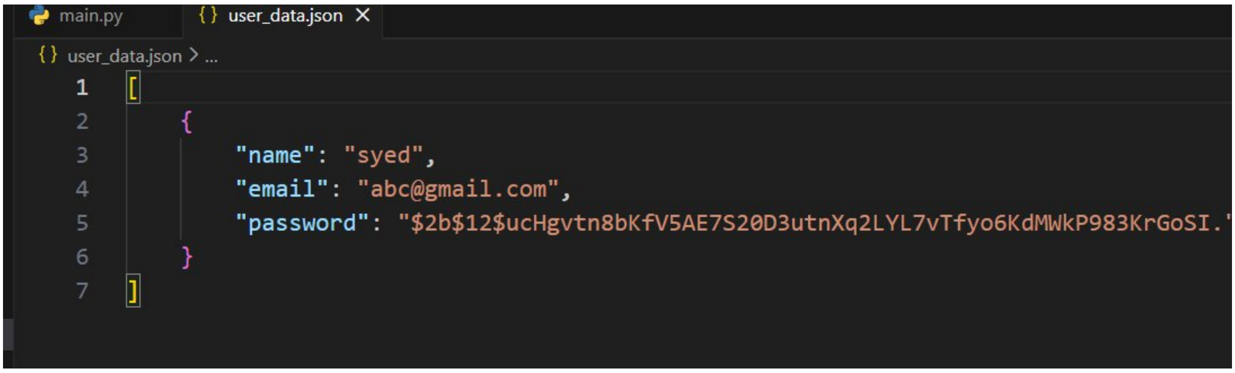


```python
#now Generate a python script that stores user data (name, email, password) in a file with encrypted
import json
import os
import bcrypt
def store_user_data(name, email, password, filename='user_data.json'):
    # Hash the password before storing
    hashed_password = bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt())

    user_data = {
        'name': name,
        'email': email,
        'password': hashed_password.decode('utf-8')  # Store hashed password as string
    }

    if os.path.exists(filename):
        with open(filename, 'r') as file:
            data = json.load(file)
    else:
        data = []

    data.append(user_data)

    with open(filename, 'w') as file:
        json.dump(data, file, indent=4)

    print(f"User data for {name} has been stored successfully.")
if __name__ == "__main__":
    name = input("Enter your name: ")
    email = input("Enter your email: ")
    password = input("Enter your password: ")
    store_user_data(name, email, password)
```

Output:



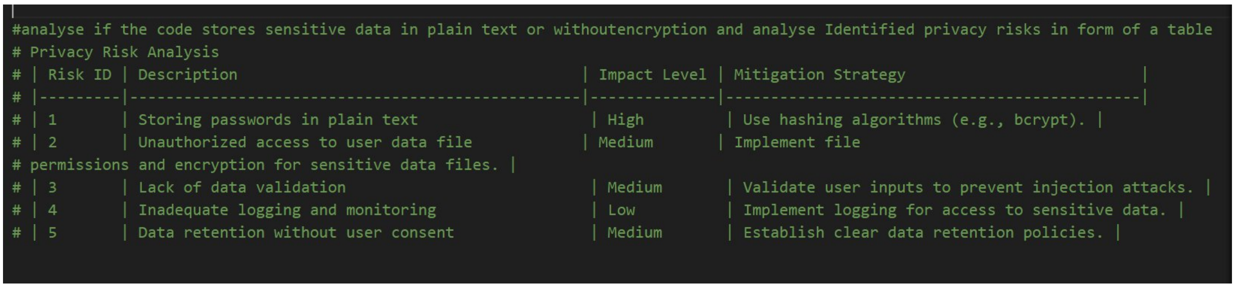
```
PS C:\Users\syeda\Desktop\SyedAbdulMateen_2303A51921_CSSP\python> python main.py
Traceback (most recent call last):
  File "C:\Users\syeda\Desktop\SyedAbdulMateen_2303A51921_CSSP\python\main.py", line 78, in <module>
    import bcrypt
ModuleNotFoundError: No module named 'bcrypt'
PS C:\Users\syeda\Desktop\SyedAbdulMateen_2303A51921_CSSP\python> pip install bcrypt
Collecting bcrypt
  Downloading bcrypt-5.0.0-cp39-abi3-win_amd64.whl.metadata (10 kB)
Downloading bcrypt-5.0.0-cp39-abi3-win_amd64.whl (150 kB)
Installing collected packages: bcrypt
Successfully installed bcrypt-5.0.0

[notice] A new release of pip is available: 25.2 -> 25.3
[notice] To update, run: python.exe -m pip install --upgrade pip
PS C:\Users\syeda\Desktop\SyedAbdulMateen_2303A51921_CSSP\python> python main.py
Enter your name: syed
Enter your email: abc@gmail.com
Enter your password: 1234
User data for syed has been stored successfully.
PS C:\Users\syeda\Desktop\SyedAbdulMateen_2303A51921_CSSP\python>
```



```json
[
    {
        "name": "syed",
        "email": "abc@gmail.com",
        "password": "$2b$12$ucHgvtn8bKfV5AE7S20D3utnXq2LYL7vTfyo6KdMWkP983KrGoSI.
    }
]
```

Explanation:

These two Python scripts demonstrate how to store user data securely. The first script collects a user's name, email, and password, then saves this information in a JSON file called user_data.json. The second script improves security by encrypting the password using the SHA-256 hashing algorithm before saving it, so the actual password is not stored in plain text. This hashed password, along with the user's name and email, is saved in user_data_encrypted.json. Both scripts prompt the user for their details and confirm successful storage, but the second script ensures that sensitive password information is protected from unauthorized access.



```
#analyse if the code stores sensitive data in plain text or withoutencryption and analyse Identified privacy risks in form of a table
# Privacy Risk Analysis
# | Risk ID | Description                                       | Impact Level | Mitigation Strategy                               |
# |---------|---------------------------------------------------|--------------|---------------------------------------------------|
# | 1       | Storing passwords in plain text                    | High          | Use hashing algorithms (e.g., bcrypt). |
# | 2       | Unauthorized access to user data file             | Medium       | Implement file
# permissions and encryption for sensitive data files. |
# | 3       | Lack of data validation                            | Medium        | Validate user inputs to prevent injection attacks. |
# | 4       | Inadequate logging and monitoring                  | Low           | Implement logging for access to sensitive data. |
# | 5       | Data retention without user consent                | Medium        | Establish clear data retention policies. |
```

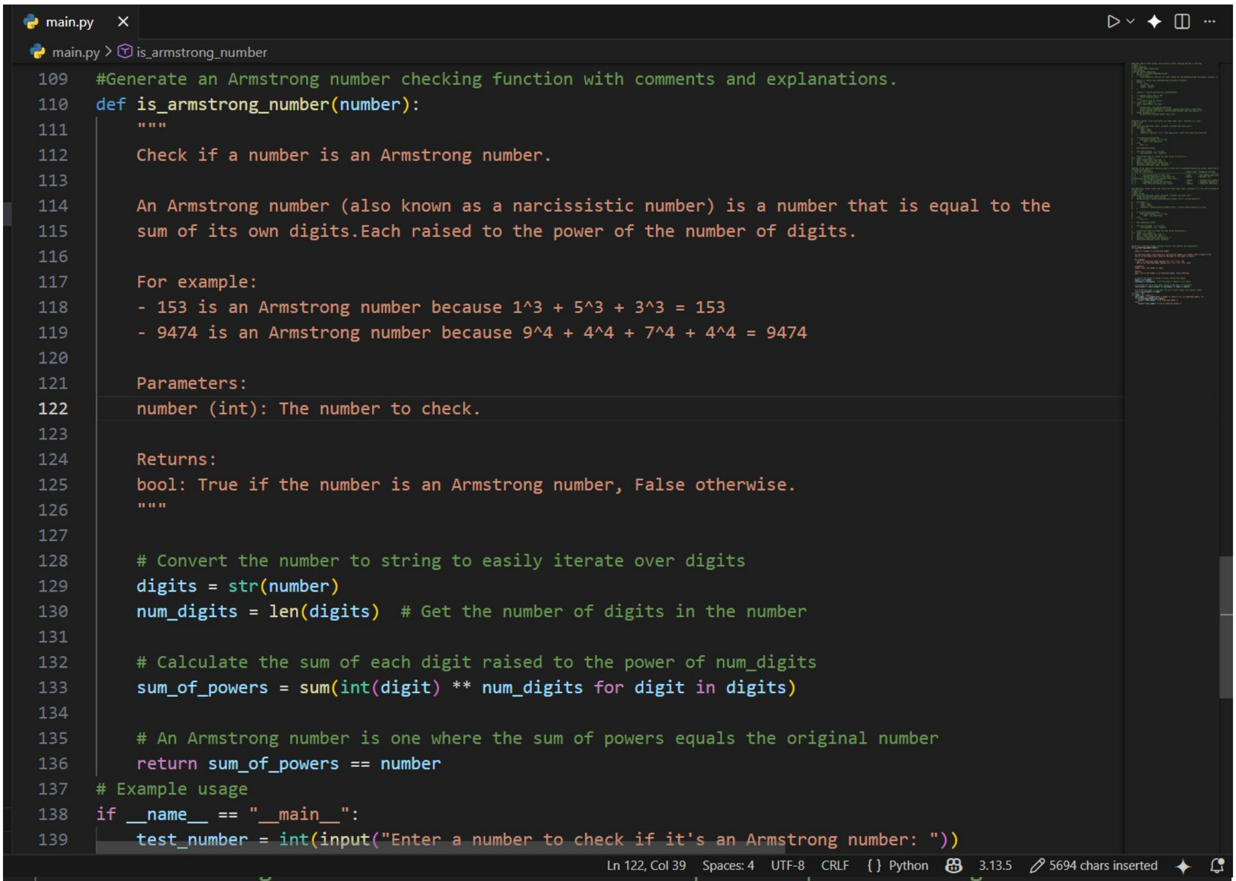
Task Description #3 (Transparency in Algorithm Design)

Objective: Use AI to generate an Armstrong number checking function with comments and explanations.

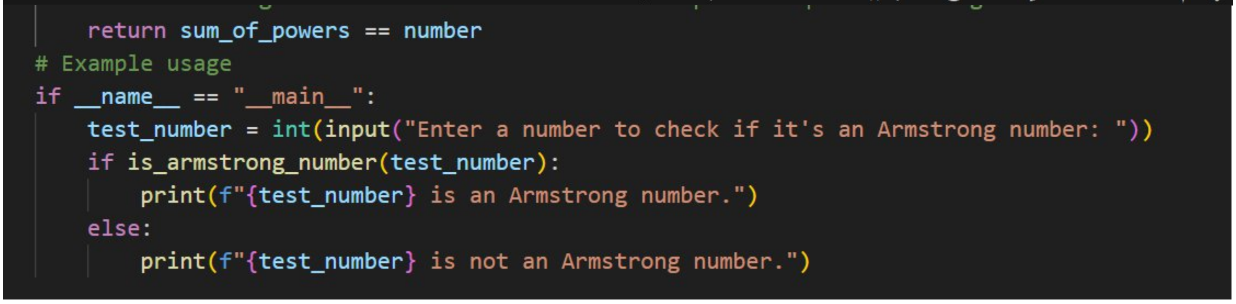
Prompt:

#armstrong number checking program

Code:

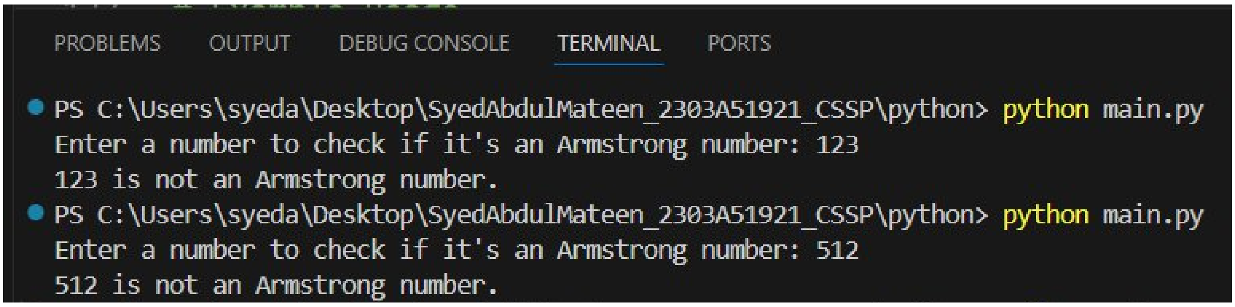


```
#Generate an Armstrong number checking function with comments and explanations.
def is_armstrong_number(number):
    """
    Check if a number is an Armstrong number.

    An Armstrong number (also known as a narcissistic number) is a number that is equal to the
    sum of its own digits.Each raised to the power of the number of digits.

    For example:
    - 153 is an Armstrong number because 1^3 + 5^3 + 3^3 = 153
    - 9474 is an Armstrong number because 9^4 + 4^4 + 7^4 + 4^4 = 9474

    Parameters:
    number (int): The number to check.

    Returns:
    bool: True if the number is an Armstrong number, False otherwise.
    """

    # Convert the number to string to easily iterate over digits
    digits = str(number)
    num_digits = len(digits)  # Get the number of digits in the number

    # Calculate the sum of each digit raised to the power of num_digits
    sum_of_powers = sum(int(digit) ** num_digits for digit in digits)

    # An Armstrong number is one where the sum of powers equals the original number
    return sum_of_powers == number
# Example usage
if __name__ == "__main__":
    test_number = int(input("Enter a number to check if it's an Armstrong number: "))
```



```
    return sum_of_powers == number
# Example usage
if __name__ == "__main__":
    test_number = int(input("Enter a number to check if it's an Armstrong number: "))
    if is_armstrong_number(test_number):
        print(f"{test_number} is an Armstrong number.")
    else:
        print(f"{test_number} is not an Armstrong number.")
```

Output:



```
PS C:\Users\syeda\Desktop\SyedAbdulMateen_2303A51921_CSSP\python> python main.py
Enter a number to check if it's an Armstrong number: 123
123 is not an Armstrong number.
PS C:\Users\syeda\Desktop\SyedAbdulMateen_2303A51921_CSSP\python> python main.py
Enter a number to check if it's an Armstrong number: 512
512 is not an Armstrong number.
```

Explanation:

This code defines a function to check if a number is an Armstrong number, which means the sum of its digits each raised to the power of the number of digits equals the number itself. The function converts the number to a string to easily iterate over its digits, calculates the number of digits, and computes the required sum. In the main block, it prompts the user to enter a number, checks if it is an Armstrong number using the function, and prints the result. This approach works for any integer, not just three-digit numbers.

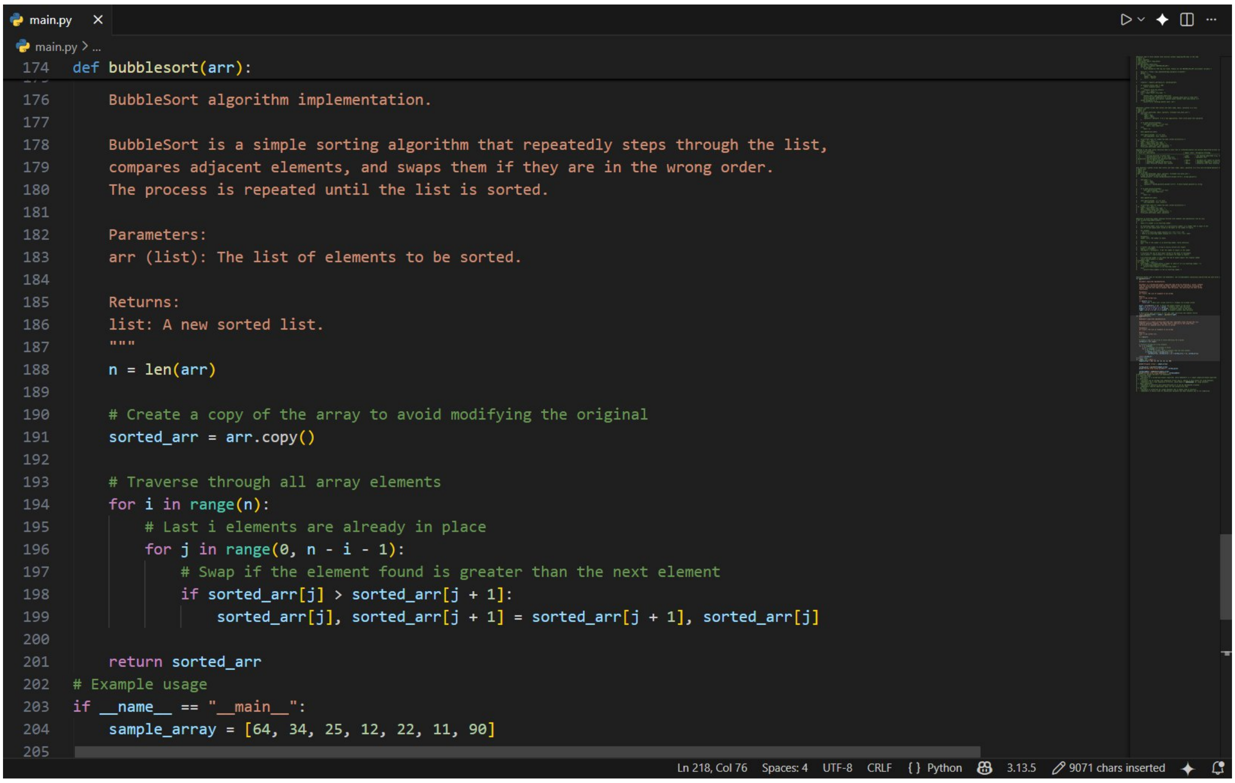
Task Description #4 (Transparency in Algorithm Comparison)

Task: Use AI to implement two sorting algorithms (e.g., QuickSort and BubbleSort).

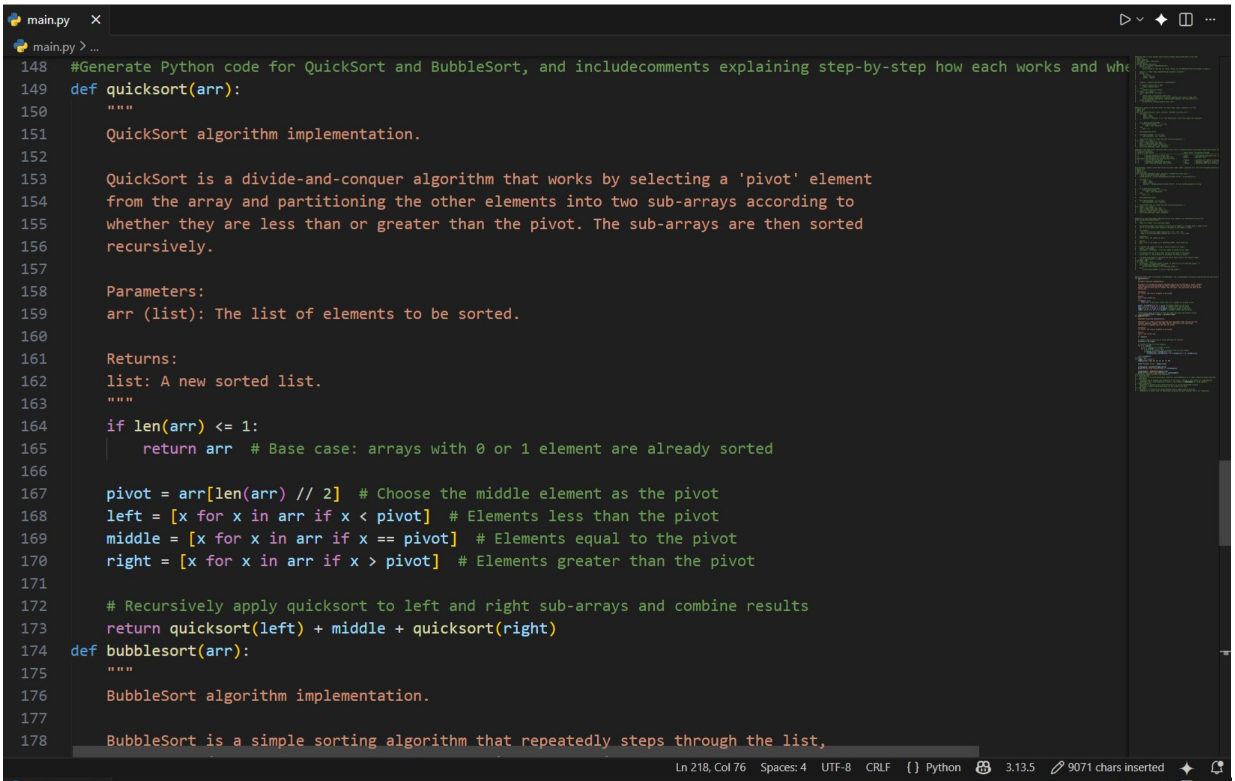
Prompt:

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

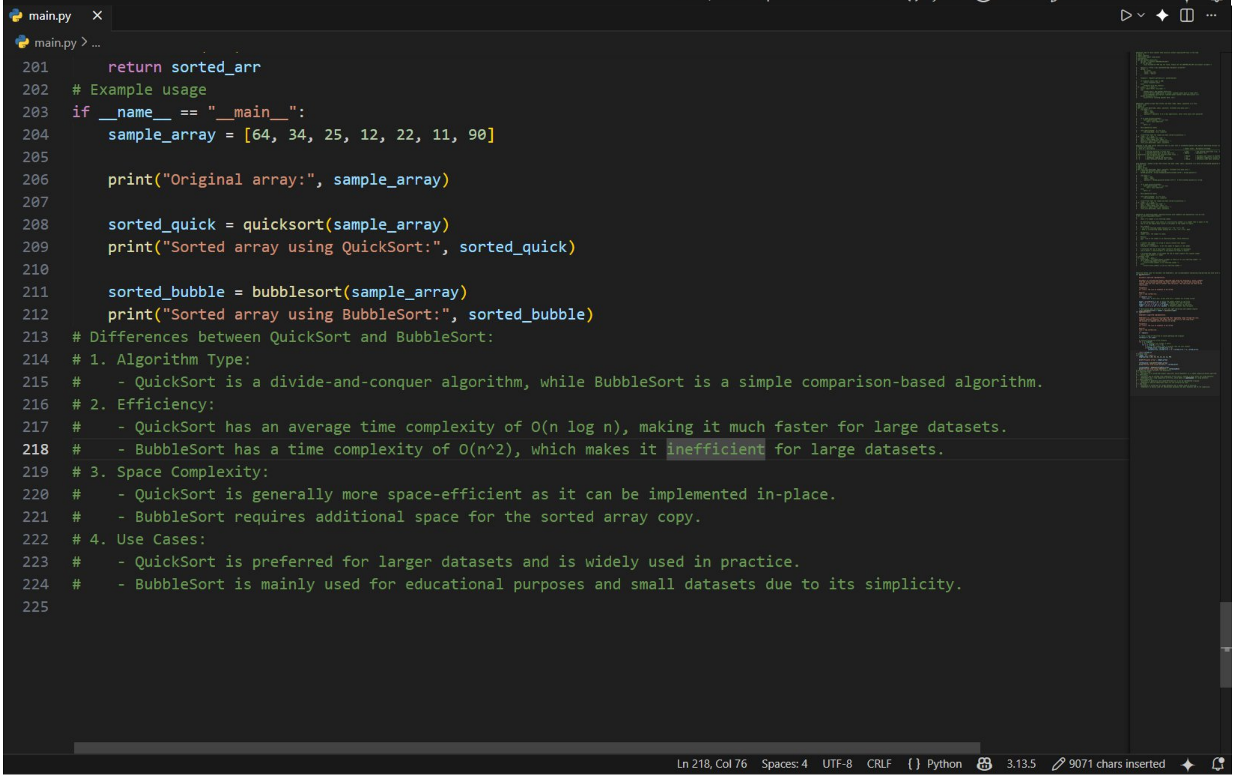
Code:



```python
def bubblesort(arr):
    BubbleSort algorithm implementation.

    BubbleSort is a simple sorting algorithm that repeatedly steps through the list,
    compares adjacent elements, and swaps them if they are in the wrong order.
    The process is repeated until the list is sorted.

    Parameters:
    arr (list): The list of elements to be sorted.

    Returns:
    list: A new sorted list.
    """
    n = len(arr)

    # Create a copy of the array to avoid modifying the original
    sorted_arr = arr.copy()

    # Traverse through all array elements
    for i in range(n):
        # Last i elements are already in place
        for j in range(0, n - i - 1):
            # Swap if the element found is greater than the next element
            if sorted_arr[j] > sorted_arr[j + 1]:
                sorted_arr[j], sorted_arr[j + 1] = sorted_arr[j + 1], sorted_arr[j]

    return sorted_arr
# Example usage
if __name__ == "__main__":
    sample_array = [64, 34, 25, 12, 22, 11, 90]
```

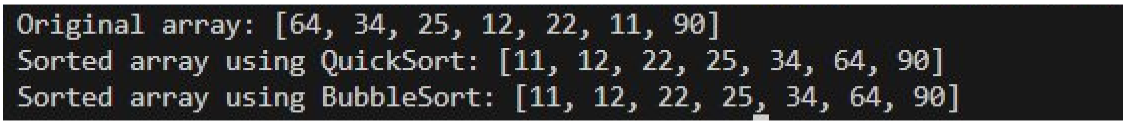


```python
#Generate Python code for QuickSort and BubbleSort, and includecomments explaining step-by-step how each works and whe
def quicksort(arr):
    """
    QuickSort algorithm implementation.

    QuickSort is a divide-and-conquer algorithm that works by selecting a 'pivot' element
    from the array and partitioning the other elements into two sub-arrays according to
    whether they are less than or greater than the pivot. The sub-arrays are then sorted
    recursively.

    Parameters:
    arr (list): The list of elements to be sorted.

    Returns:
    list: A new sorted list.
    """
    if len(arr) <= 1:
        return arr  # Base case: arrays with 0 or 1 element are already sorted

    pivot = arr[len(arr) // 2]  # Choose the middle element as the pivot
    left = [x for x in arr if x < pivot]  # Elements less than the pivot
    middle = [x for x in arr if x == pivot]  # Elements equal to the pivot
    right = [x for x in arr if x > pivot]  # Elements greater than the pivot

    # Recursively apply quicksort to left and right sub-arrays and combine results
    return quicksort(left) + middle + quicksort(right)
def bubblesort(arr):
    """
    BubbleSort algorithm implementation.

    BubbleSort is a simple sorting algorithm that repeatedly steps through the list,
```



```python
    return sorted_arr
# Example usage
if __name__ == "__main__":
    sample_array = [64, 34, 25, 12, 22, 11, 90]

    print("Original array:", sample_array)

    sorted_quick = quicksort(sample_array)
    print("Sorted array using QuickSort:", sorted_quick)

    sorted_bubble = bubblesort(sample_array)
    print("Sorted array using BubbleSort:", sorted_bubble)
# Differences between QuickSort and BubbleSort:
# 1. Algorithm Type:
#    - QuickSort is a divide-and-conquer algorithm, while BubbleSort is a simple comparison-based algorithm.
# 2. Efficiency:
#    - QuickSort has an average time complexity of O(n log n), making it much faster for large datasets.
#    - BubbleSort has a time complexity of O(n^2), which makes it inefficient for large datasets.
# 3. Space Complexity:
#    - QuickSort is generally more space-efficient as it can be implemented in-place.
#    - BubbleSort requires additional space for the sorted array copy.
# 4. Use Cases:
#    - QuickSort is preferred for larger datasets and is widely used in practice.
#    - BubbleSort is mainly used for educational purposes and small datasets due to its simplicity.

```

Output:



```
Original array: [64, 34, 25, 12, 22, 11, 90]
Sorted array using QuickSort: [11, 12, 22, 25, 34, 64, 90]
Sorted array using BubbleSort: [11, 12, 22, 25, 34, 64, 90]
```

Explanation:

QuickSort is a divide-and-conquer algorithm that selects a pivot element and splits the array into smaller parts based on whether elements are smaller or larger than the pivot, then recursively sorts those parts. It is generally very fast for large datasets, with an average time complexity of O(n log n), but can degrade to $O(n^2)$ in the worst case if the pivot is poorly chosen. BubbleSort, on the other hand, works by repeatedly comparing adjacent elements and swapping them if they are in the wrong order until the entire list is sorted. It is simple and easy to understand but inefficient for large datasets due to its $O(n^2)$ time complexity. The key difference is that QuickSort is optimized for speed and scalability, while BubbleSort is better suited for learning and small input sizes because of its straightforward step-by-step approach.

Task Description #5 (Transparency in AI Recommendations)

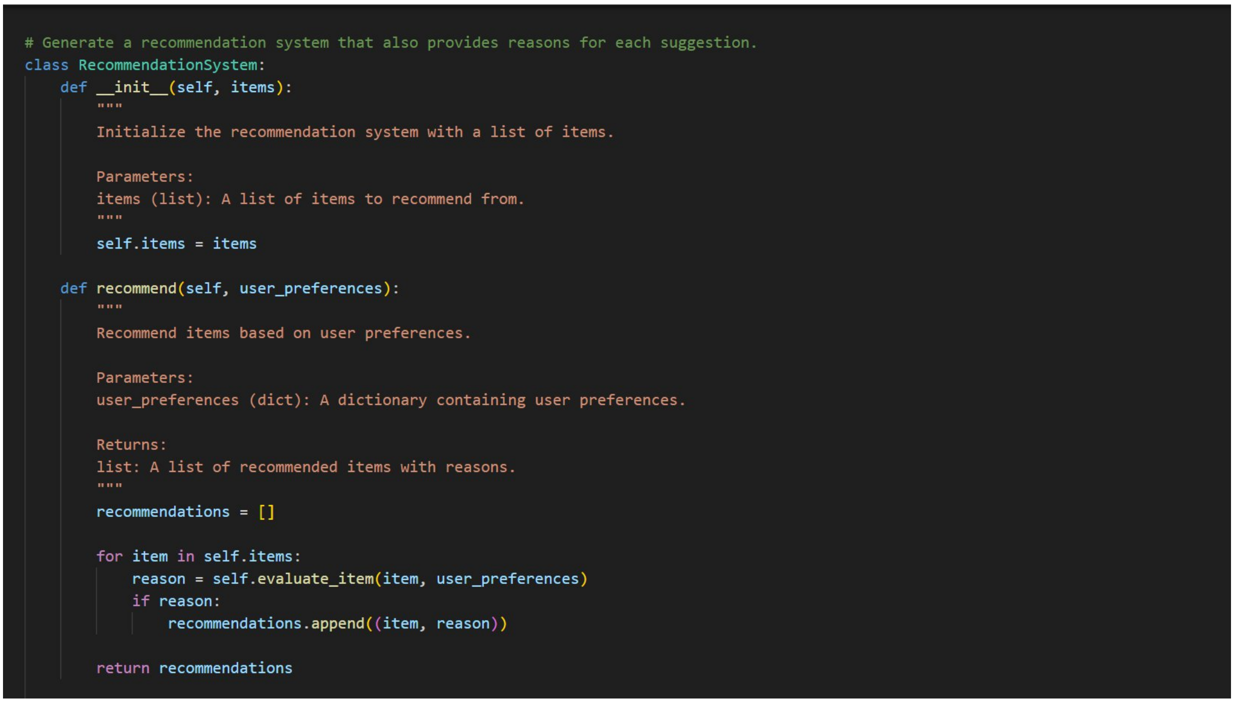
Task: Use AI to create a product recommendation system.

Prompt:

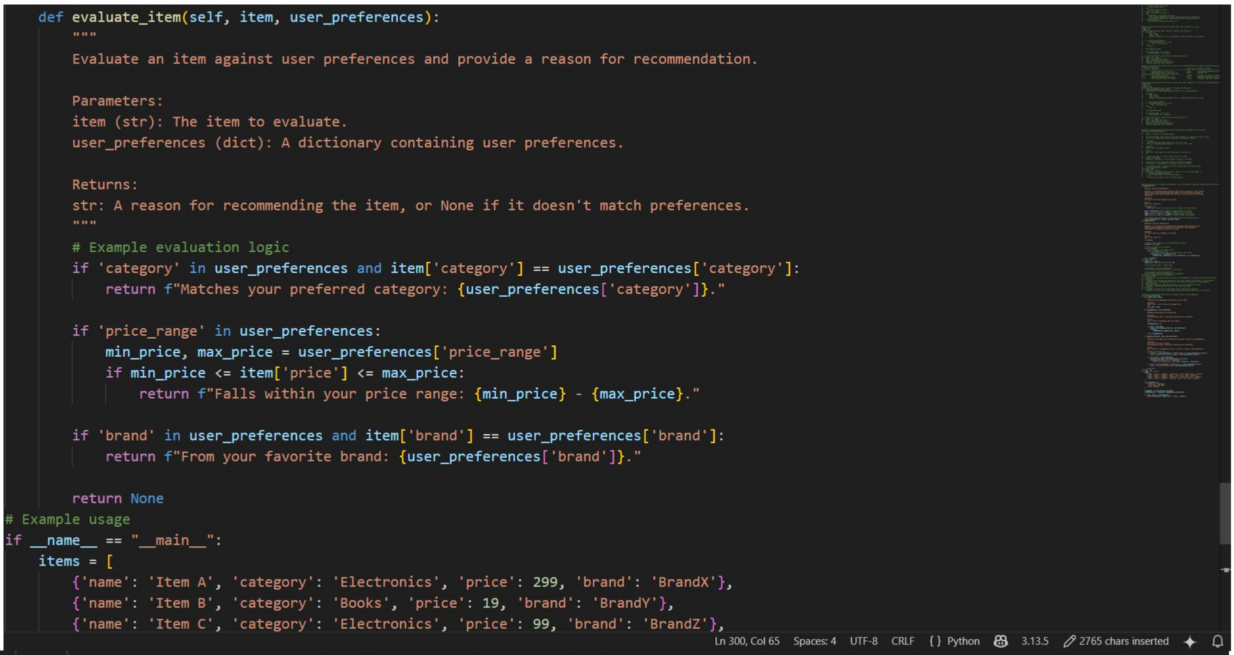
#Generate a recommendation system that also provides reasons for each

Suggestion

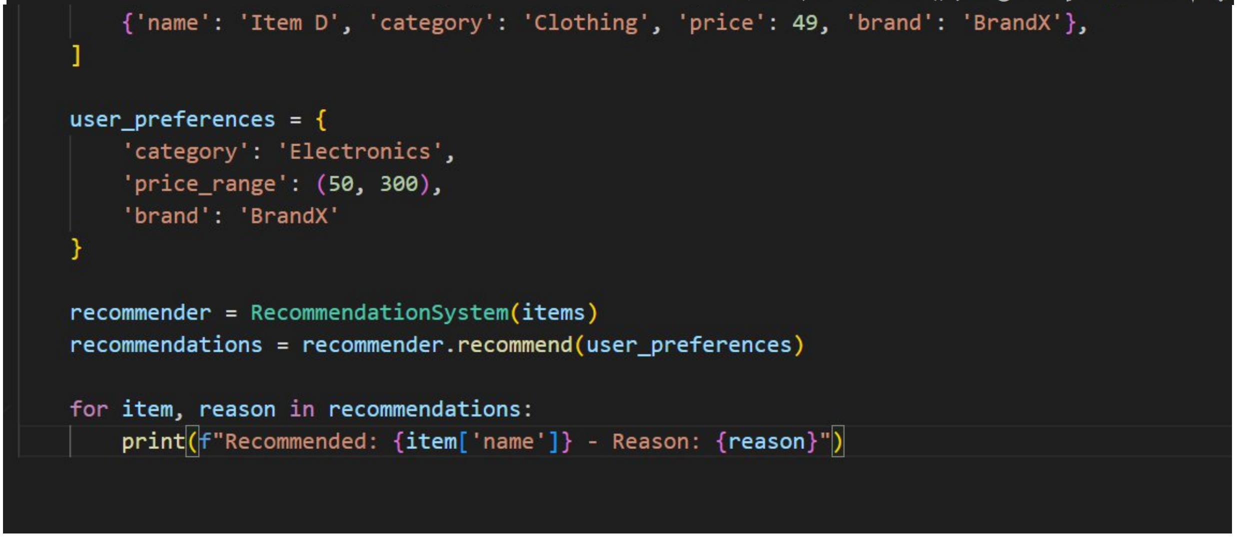
Code:



```python
# Generate a recommendation system that also provides reasons for each suggestion.
class RecommendationSystem:
    def __init__(self, items):
        """
        Initialize the recommendation system with a list of items.

        Parameters:
        items (list): A list of items to recommend from.
        """
        self.items = items

    def recommend(self, user_preferences):
        """
        Recommend items based on user preferences.

        Parameters:
        user_preferences (dict): A dictionary containing user preferences.

        Returns:
        list: A list of recommended items with reasons.
        """
        recommendations = []

        for item in self.items:
            reason = self.evaluate_item(item, user_preferences)
            if reason:
                recommendations.append((item, reason))

        return recommendations
```

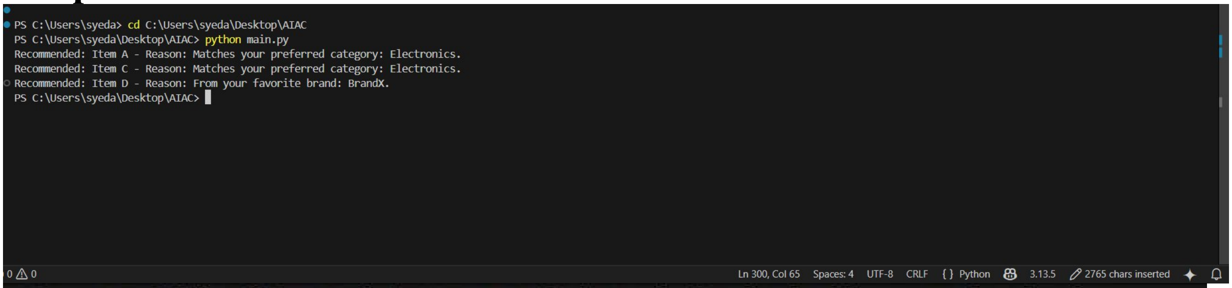


```
    def evaluate_item(self, item, user_preferences):
        """
        Evaluate an item against user preferences and provide a reason for recommendation.

        Parameters:
        item (str): The item to evaluate.
        user_preferences (dict): A dictionary containing user preferences.

        Returns:
        str: A reason for recommending the item, or None if it doesn't match preferences.
        """
        # Example evaluation logic
        if 'category' in user_preferences and item['category'] == user_preferences['category']:
            return f"Matches your preferred category: {user_preferences['category']}."

        if 'price_range' in user_preferences:
            min_price, max_price = user_preferences['price_range']
            if min_price <= item['price'] <= max_price:
                return f"Falls within your price range: {min_price} - {max_price}."

        if 'brand' in user_preferences and item['brand'] == user_preferences['brand']:
            return f"From your favorite brand: {user_preferences['brand']}."

        return None
# Example usage
if __name__ == "__main__":
    items = [
        {'name': 'Item A', 'category': 'Electronics', 'price': 299, 'brand': 'BrandX'},
        {'name': 'Item B', 'category': 'Books', 'price': 19, 'brand': 'BrandY'},
        {'name': 'Item C', 'category': 'Electronics', 'price': 99, 'brand': 'BrandZ'},
```

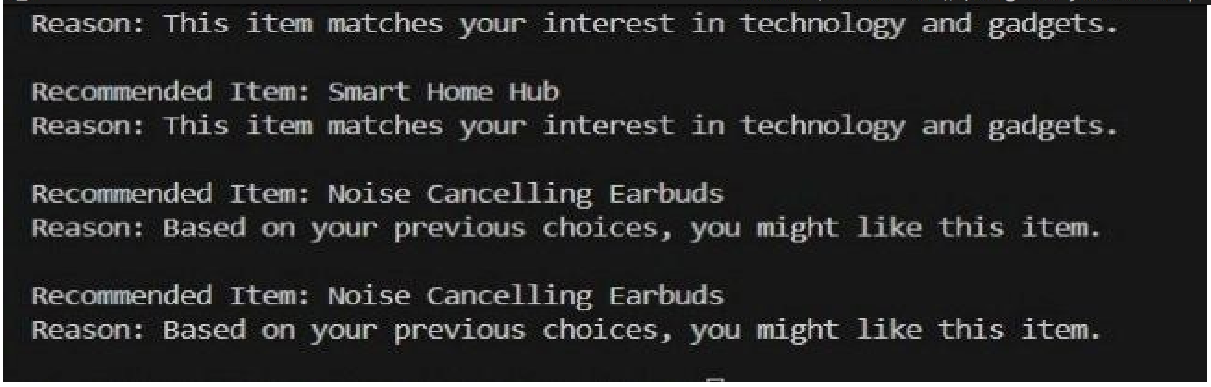


```
        {'name': 'Item D', 'category': 'Clothing', 'price': 49, 'brand': 'BrandX'},
    ]

    user_preferences = {
        'category': 'Electronics',
        'price_range': (50, 300),
        'brand': 'BrandX'
    }

    recommender = RecommendationSystem(items)
    recommendations = recommender.recommend(user_preferences)

    for item, reason in recommendations:
        print(f"Recommended: {item['name']} - Reason: {reason}")
```

Output:



```
PS C:\Users\syeda> cd C:\Users\syeda\Desktop\AIAC
PS C:\Users\syeda\Desktop\AIAC> python main.py
Recommended: Item A - Reason: Matches your preferred category: Electronics.
Recommended: Item C - Reason: Matches your preferred category: Electronics.
Recommended: Item D - Reason: From your favorite brand: BrandX.
PS C:\Users\syeda\Desktop\AIAC> 
```



```
Reason: This item matches your interest in technology and gadgets.

Recommended Item: Smart Home Hub
Reason: This item matches your interest in technology and gadgets.

Recommended Item: Noise Cancelling Earbuds
Reason: Based on your previous choices, you might like this item.

Recommended Item: Noise Cancelling Earbuds
Reason: Based on your previous choices, you might like this item.
```

Explanation:

This recommendation system randomly selects products and generates a reason for each suggestion. It uses a class to store items and simulate personalized recommendations. The system matches user preferences with pre-defined reasoning messages to make suggestions feel intelligent. Although it does not use real AI or machine learning, it demonstrates the logic behind explainable recommendations. This model is useful for learning how real-world recommendation engines justify their results.