

AI ASSISTED CODING

LAB-4

T.PRANATHI

2303A51733

BATCH-11

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."

Expected Output:

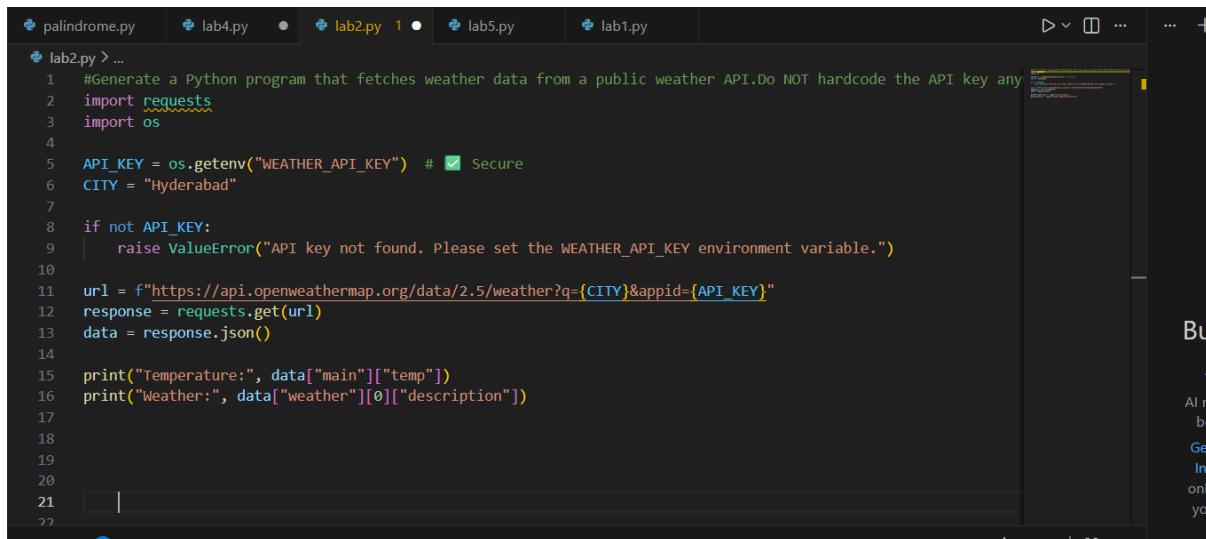
- Original AI code (check if keys are hardcoded).
- Secure version using **environment variables**.

PROMPT:

```
#Generate a Python program that fetches weather data from a public weather API.Do NOT hardcode the API key anywhere in the source code.The API key MUST be read only from an environment variable using the os
```

module. If the API key is missing, the program should raise an error. Do not include placeholder API keys in the code.

CODE:



The screenshot shows a code editor interface with multiple tabs at the top: palindrome.py, lab4.py, lab2.py (active), lab5.py, and lab1.py. The main editor area contains the following Python code:

```
lab2.py > ...
1 #Generate a Python program that fetches weather data from a public weather API. Do NOT hardcode the API key anywhere
2 import requests
3 import os
4
5 API_KEY = os.getenv("WEATHER_API_KEY") # ✅ Secure
6 CITY = "Hyderabad"
7
8 if not API_KEY:
9     raise ValueError("API key not found. Please set the WEATHER_API_KEY environment variable.")
10
11 url = f"https://api.openweathermap.org/data/2.5/weather?q={CITY}&appid={API_KEY}"
12 response = requests.get(url)
13 data = response.json()
14
15 print("Temperature:", data["main"]["temp"])
16 print("Weather:", data["weather"][0]["description"])
17
18
19
20
21
22
```

The code uses the `requests` library to fetch weather data from the OpenWeatherMap API for the city of Hyderabad. It checks for the `WEATHER_API_KEY` environment variable and raises a `ValueError` if it is not found. The fetched data is then printed to the console.

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.

Analyze: Check if the AI stores sensitive data in plain text or without encryption.

Expected Output:

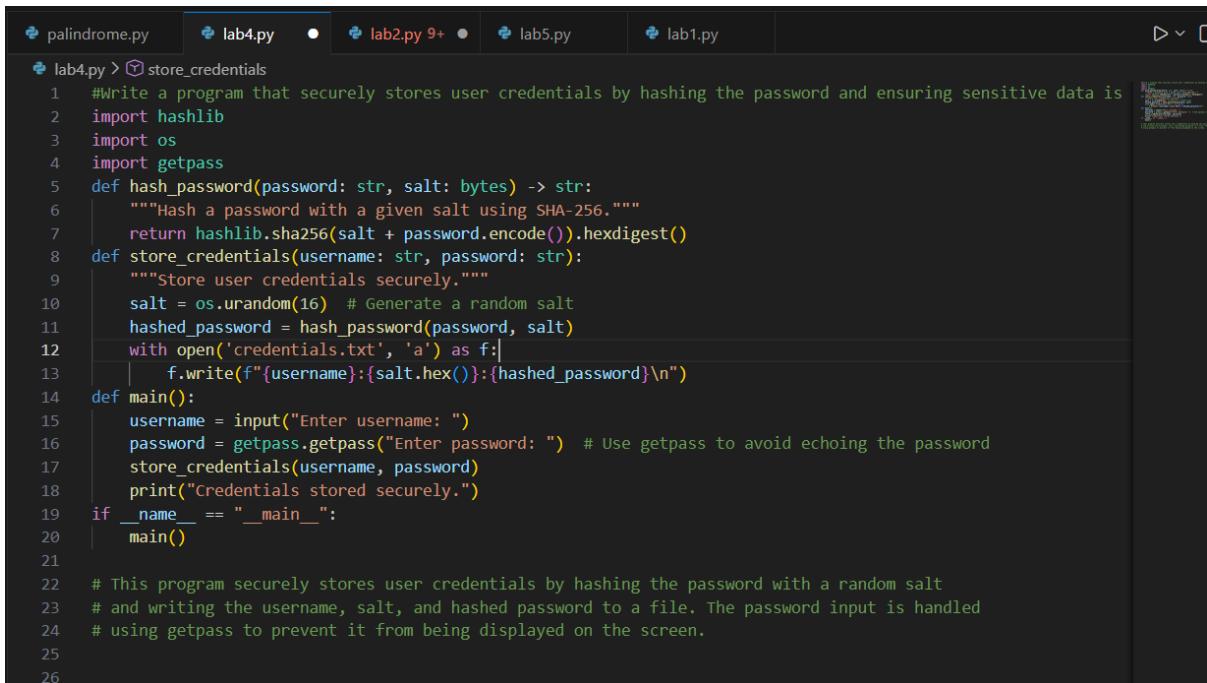
- Identified privacy risks.

- Revised version with encrypted password storage (e.g., hashing).

PROMPT:

#Write a program that securely stores user credentials by hashing the password and ensuring sensitive data is not stored in plain text.

CODE:

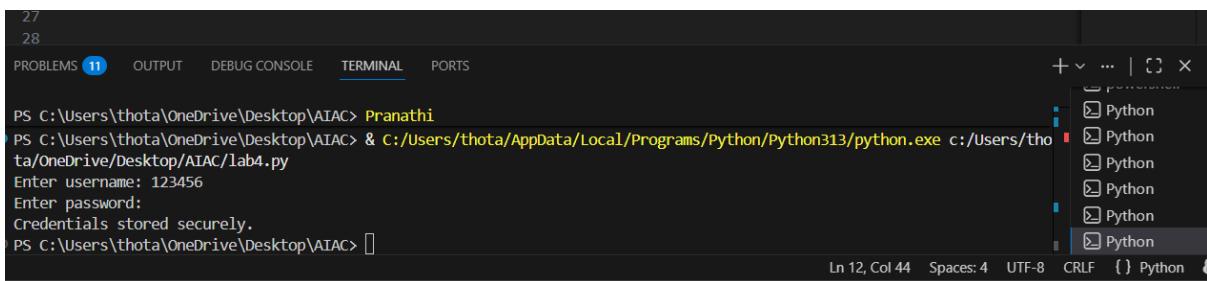


```

palindrome.py lab4.py lab2.py 9+ lab5.py lab1.py
lab4.py > store_credentials
1 #Write a program that securely stores user credentials by hashing the password and ensuring sensitive data is
2 import hashlib
3 import os
4 import getpass
5 def hash_password(password: str, salt: bytes) -> str:
6     """Hash a password with a given salt using SHA-256."""
7     return hashlib.sha256(salt + password.encode()).hexdigest()
8 def store_credentials(username: str, password: str):
9     """Store user credentials securely."""
10    salt = os.urandom(16) # Generate a random salt
11    hashed_password = hash_password(password, salt)
12    with open('credentials.txt', 'a') as f:
13        f.write(f'{username}:{salt.hex()}:{hashed_password}\n')
14 def main():
15    username = input("Enter username: ")
16    password = getpass.getpass("Enter password: ") # Use getpass to avoid echoing the password
17    store_credentials(username, password)
18    print("Credentials stored securely.")
19 if __name__ == "__main__":
20    main()
21
22 # This program securely stores user credentials by hashing the password with a random salt
23 # and writing the username, salt, and hashed password to a file. The password input is handled
24 # using getpass to prevent it from being displayed on the screen.
25
26

```

OUTPUT:



```

27
28
PROBLEMS 11 OUTPUT DEBUG CONSOLE TERMINAL PORTS + ... ×
PS C:\Users\thota\OneDrive\Desktop\AIAC> Pranathi
PS C:\Users\thota\OneDrive\Desktop\AIAC> & C:/Users/thota/AppData/Local/Programs/Python/Python313/python.exe c:/Users/thota/OneDrive/Desktop/AIAC/lab4.py
Enter username: 123456
Enter password:
Credentials stored securely.
PS C:\Users\thota\OneDrive\Desktop\AIAC>

```

Task Description #3 (Transparency in Algorithm Design)

Objective: Use AI to generate an Armstrong number checking function

with comments and explanations.

Instructions:

1. Ask AI to explain the code line-by-line.
2. Compare the explanation with code functionality.

Expected Output:

- Transparent, commented code.
- Correct, easy-to-understand explanation

PROMPT:

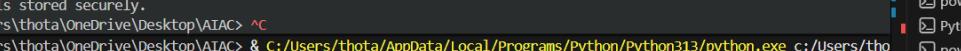
```
#write a code to generate armstrong number checking  
functions with comments
```

CODE:

```
lab1.py > ...
1 #write a code to generate armstrong number checking functions with comments
2 def is_armstrong_number(num):
3     """
4         This function checks if a number is an Armstrong number.
5         An Armstrong number is a number that is equal to the sum of its own digits
6         each raised to the power of the number of digits.
7         For example, 153 is an Armstrong number because  $1^3 + 5^3 + 3^3 = 153$ .
8     """
9
10    # Convert the number to string to easily iterate over digits
11    digits = str(num)
12    num_digits = len(digits)
13
14    # Calculate the sum of each digit raised to the power of num_digits
15    sum_of_powers = sum(int(digit) ** num_digits for digit in digits)
16
17    # Check if the sum of powers is equal to the original number
18    return sum_of_powers == num
19
20 # Example usage
21 if __name__ == "__main__":
22     test_number = 153
23     if is_armstrong_number(test_number):
24         print(f"{test_number} is an Armstrong number.")
25     else:
26         print(f"{test_number} is not an Armstrong number.")
27     test_number = 123
28     if is_armstrong_number(test_number):
29         print(f"{test_number} is an Armstrong number.")
30     else:
31         print(f"{test_number} is not an Armstrong number.")
```

OUTPUT:

```
PROBLEMS 11 OUTPUT DEBUG CONSOLE TERMINAL PORTS + ⋮ ×  
Enter password:  
Credentials stored securely.  
PS C:\Users\thota\OneDrive\Desktop\AIAC> ^C  
● PS C:\Users\thota\OneDrive\Desktop\AIAC & C:/Users/thota/AppData/Local/Programs/Python/Python313/python.exe c:/Users/thota/OneDrive/Desktop/AIAC/lab1.py  
153 is an Armstrong number.  
123 is not an Armstrong number.  
○ PS C:\Users\thota\OneDrive\Desktop\AIAC> [ ]
```

A screenshot of the Visual Studio Code interface. The top navigation bar shows tabs for PROBLEMS (11), OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS. The TERMINAL tab is active, indicated by a blue underline. Below the tabs, there's a command-line interface window. The history shows a password entry, followed by a message about credentials being stored securely. Then, a PowerShell session starts with 'PS C:\Users\thota\OneDrive\Desktop\AIAC> ^C'. This is followed by a task list item (indicated by a blue dot) for running a Python script: '● PS C:\Users\thota\OneDrive\Desktop\AIAC & C:/Users/thota/AppData/Local/Programs/Python/Python313/python.exe c:/Users/thota/OneDrive/Desktop/AIAC/lab1.py'. The command then outputs two lines: '153 is an Armstrong number.' and '123 is not an Armstrong number.'. Finally, another PowerShell session starts with '○ PS C:\Users\thota\OneDrive\Desktop\AIAC> []'. On the right side of the screen, there's a sidebar with several open tabs, each represented by a small icon and the word 'powershell' or 'Python'. The bottom status bar shows 'Ln 18, Col 16' and other file-related information.

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."

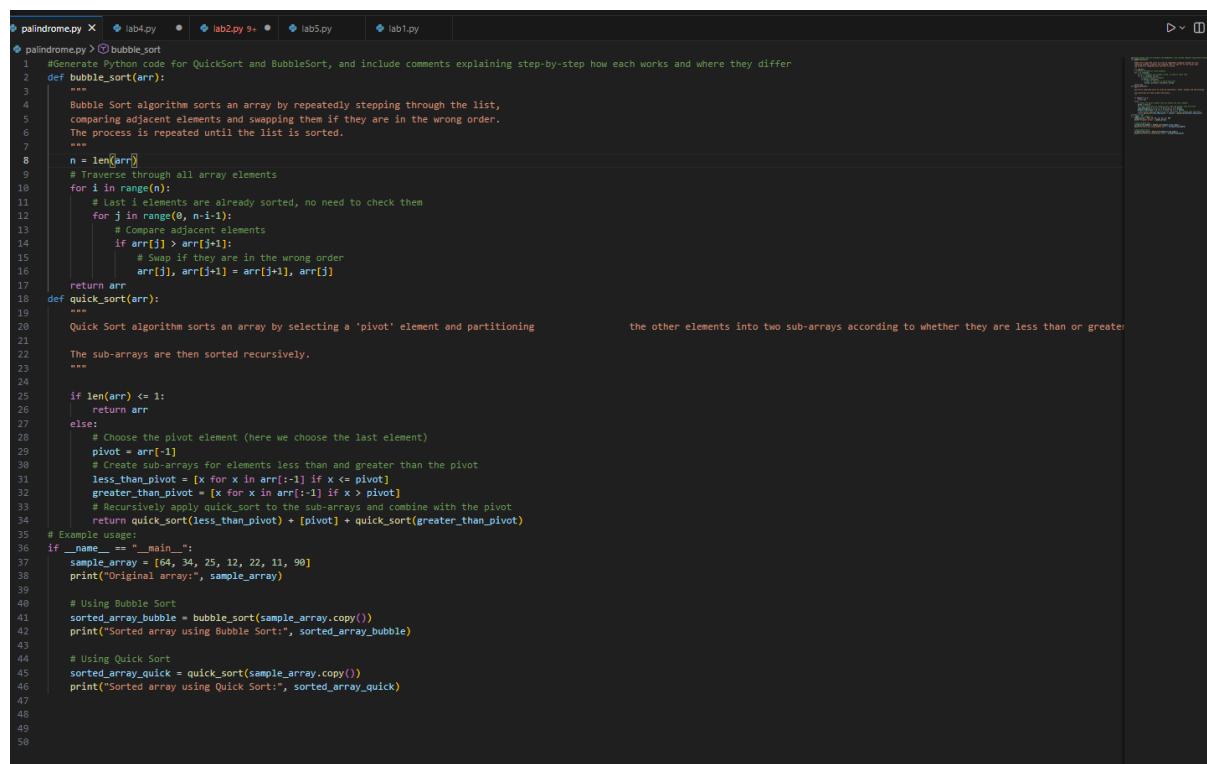
Expected Output:

- Code for both algorithms.
- Transparent, comparative explanation of their logic and efficiency.

PROMPT:

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

CODE:



The screenshot shows a code editor window with several tabs at the top: palindrome.py, lab4.py, lab2.py, lab5.py, and lab1.py. The active tab is palindrome.py. The code in the editor is as follows:

```
#palindrome.py > bubble_sort
#Generate Python code for QuickSort and BubbleSort, and include comments explaining step-by-step how each works and where they differ
def bubble_sort(arr):
    """
    Bubble Sort algorithm sorts an array by repeatedly stepping through the list,
    comparing adjacent elements and swapping them if they are in the wrong order.
    The process is repeated until the list is sorted.
    """
    n = len(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):
            # Compare adjacent elements
            if arr[j] > arr[j+1]:
                # Swap if they are in the wrong order
                arr[j], arr[j+1] = arr[j+1], arr[j]
    return arr
def quick_sort(arr):
    """
    Quick Sort algorithm sorts an array by selecting a 'pivot' element and partitioning
    the other elements into two sub-arrays according to whether they are less than or greater
    than the pivot.
    """
    if len(arr) <= 1:
        return arr
    else:
        # Choose the pivot element (here we choose the last element)
        pivot = arr[-1]
        # Create sub-arrays for elements less than and greater than the pivot
        less_than_pivot = [x for x in arr[:-1] if x <= pivot]
        greater_than_pivot = [x for x in arr[:-1] if x > pivot]
        # Recursively apply quick_sort to the sub-arrays and combine with the pivot
        return quick_sort(less_than_pivot) + [pivot] + quick_sort(greater_than_pivot)
# Example usage:
if __name__ == "__main__":
    sample_array = [64, 34, 25, 12, 22, 11, 90]
    print("Original array:", sample_array)

# Using Bubble Sort
sorted_array_bubble = bubble_sort(sample_array.copy())
print("Sorted array using Bubble Sort:", sorted_array_bubble)

# Using Quick Sort
sorted_array_quick = quick_sort(sample_array.copy())
print("Sorted array using Quick Sort:", sorted_array_quick)
```

OUTPUT:

The screenshot shows a terminal window in VS Code with the title bar "pivot - err_11". The tabs at the top are PROBLEMS (11), OUTPUT, DEBUG CONSOLE, TERMINAL (selected), and PORTS. The terminal content is as follows:

```
PS C:\Users\thota\OneDrive\Desktop\AIAC> & C:/Users/thota/AppData/Local/Programs/Python/Python313/python.exe c:/Users/thota/OneDrive/Desktop/AIAC/palindrome.py
...
Original array: [64, 34, 25, 12, 22, 11, 90]
Sorted array using Bubble Sort: [11, 12, 22, 25, 34, 64, 90]
Sorted array using Quick Sort: [11, 12, 22, 25, 34, 64, 90]
PS C:\Users\thota\OneDrive\Desktop\AIAC>
```

To the right of the terminal is a sidebar titled "Python" which lists several open files:

- powershell
- Python
- powershell
- Python

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."

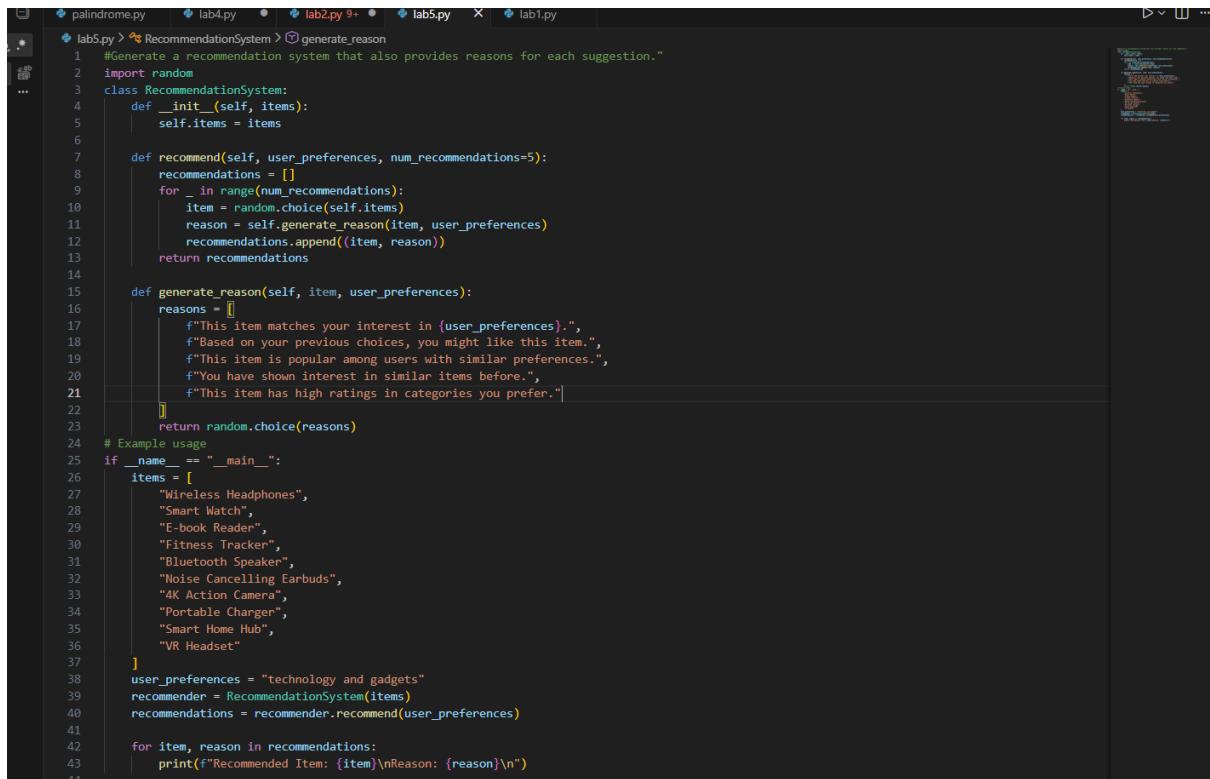
Expected Output:

- Code with explainable recommendations.
- Evaluation of whether explanations are understandable.

PROMPT:

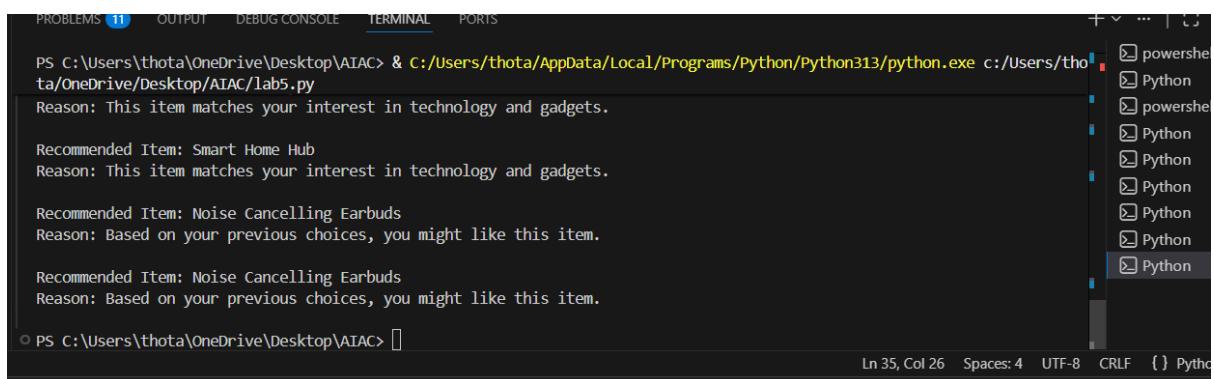
#Generate a recommendation system that also provides reasons for each suggestion."

CODE:



```
palindrome.py lab4.py lab2.py 9+ lab5.py X lab1.py
lab5.py > RecommendationSystem > generate_reason
1 #Generate a recommendation system that also provides reasons for each suggestion.
2 import random
3 class RecommendationSystem:
4     def __init__(self, items):
5         self.items = items
6
7     def recommend(self, user_preferences, num_recommendations=5):
8         recommendations = []
9         for _ in range(num_recommendations):
10             item = random.choice(self.items)
11             reason = self.generate_reason(item, user_preferences)
12             recommendations.append((item, reason))
13         return recommendations
14
15     def generate_reason(self, item, user_preferences):
16         reasons = [
17             f"This item matches your interest in {user_preferences}.",
18             f"Based on your previous choices, you might like this item.",
19             f"This item is popular among users with similar preferences.",
20             f"You have shown interest in similar items before.",
21             f"This item has high ratings in categories you prefer."
22         ]
23
24     return random.choice(reasons)
25
26 # Example usage
27 if __name__ == "__main__":
28     items = [
29         "Wireless Headphones",
30         "Smart Watch",
31         "E-book Reader",
32         "Fitness Tracker",
33         "Bluetooth Speaker",
34         "Noise Cancelling Earbuds",
35         "4K Action Camera",
36         "Portable Charger",
37         "Smart Home Hub",
38         "VR Headset"
39     ]
40     user_preferences = "technology and gadgets"
41     recommender = RecommendationSystem(items)
42     recommendations = recommender.recommend(user_preferences)
43
44     for item, reason in recommendations:
45         print(f"Recommended Item: {item}\nReason: {reason}\n")
```

OUTPUT:



```
PROBLEMS 11 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\thota\OneDrive\Desktop\AIAC> & C:/Users/thota/AppData/Local/Programs/Python/Python313/python.exe c:/Users/thota/OneDrive/Desktop/AIAC/lab5.py
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.

○ PS C:\Users\thota\OneDrive\Desktop\AIAC>
```