AIAC Lab 5.1

Ethical Foundations – Responsible AI Coding Practices

CH UDAY

BATCH-11

Hall Ticket: 2303A510G2

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()
API_KEY = os.getenv('WEATHER_API_KEY')

def fetch_weather(city):
    # Check if API key is set
    if not API_KEY:
        return {"error": "API key not found. Please set WEATHER_API_KEY in your .env file"}

    url = f"http://api.openweathermap.org/data/2.5/weather?q={city}&appid={API_KEY}"

    try:
        response = requests.get(url)
        data = response.json()

        # Check for API errors in response
        if 'cod' in data and data['cod'] != 200:
            return {"error": f"API Error: {data.get('message', 'Unknown error')}"}

        return data
    except requests.exceptions.RequestException as e:
        return {"error": f"Request failed: {str(e)}"}

# Test the function
result1 = fetch_weather("London")
print("London:", result1)

result2 = fetch_weather("New York")
print("New York:", result2)
```

Observation:

1.API Key Is Not Hardcoded (Good Practice)

The program does not directly write the API key in the source code. Instead, it uses an environment variable (WEATHER_API_KEY), which helps prevent accidental exposure of sensitive credentials.

2.Use of .env File with dotenv Library

The code uses load_dotenv() to load environment variables from a .env file. This is a secure and widely accepted method to manage secrets separately from the codebase.

3.Potential Security Issue in URL Construction

Although the API key is stored securely, it is still appended directly to the request URL. This can expose the key in logs, browser history, or debugging tools.

4.Lack of HTTPS Protocol

The API URL uses http:// instead of https://. This is a security risk because the API key and data may be transmitted in plain text over the network.

5.Basic Error Handling Is Implemented

The program includes error handling for:

1.Missing API key

2.API errors returned by the server

3.Network-related request failures

4.This improves robustness and reliability.

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.

Prompt:

#write a python code to store the data of a person including name,email,password.

#Identified privacy risks.

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

Code:

```python
import os
import bcrypt
from dotenv import load_dotenv
load_dotenv()
class Person:
    def __init__(self, name, email, password):
        self.name = name
        self.email = email
        self.password_hash = self.hash_password(password)

    def hash_password(self, password):
        # Hash the password using bcrypt
        salt = bcrypt.gensalt()
        return bcrypt.hashpw(password.encode('utf-8'), salt)

    def check_password(self, password):
        # Check the hashed password
        return bcrypt.checkpw(password.encode('utf-8'), self.password_hash)
# Example usage
person = Person("John Doe", "john.doe@example.com", "secure_password")
print("Name:", person.name)
print("Email:", person.email)
print("Password Hash:", person.password_hash)
# Verify password
is_correct = person.check_password("secure_password")
print("Password correct:", is_correct)
```

Output:

```
Name: John Doe
Email: john.doe@example.com
Password Hash: b'$2b$12$CMJsWyScVS6soOSk1WwvzuVWBDlHwVxbrHHEIOQ/OBF08K6zArgfK'
Password correct: True
```

Observation:

The program securely stores passwords by hashing them using the bcrypt algorithm instead of storing plain text passwords.

A unique salt is generated for each password using bcrypt.gensalt(), which helps protect against rainbow table and brute-force attacks.

Password verification is implemented using bcrypt.checkpw(), ensuring safe and reliable comparison of hashed passwords.

The code does not hardcode sensitive information such as passwords or API keys, following secure and privacy-aware coding practices.

The hashed password is printed to the console, which may lead to information leakage and should be avoided in real-world applications.

The program imports unnecessary modules (os and dotenv) that are not used, reducing code clarity and violating clean coding principles.

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.

Prompt:

#generate an Armstrong number checking function with comments and explanations and explain code line by line.

#Compare the explanation with code functionality.

Code:

```
def is_armstrong_number(num):
    """
    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
    """
    # Convert the number to string to easily iterate over each digit
    digits = str(num)

    # Get the number of digits
    num_digits = len(digits)

    # Initialize sum to store the sum of each digit raised to the power of num_digits
    sum_of_powers = 0

    # Iterate over each digit in the number
    for digit in digits:
        # Convert digit back to integer and raise it to the power of num_digits, then add to sum_of_powers
        sum_of_powers += int(digit) ** num_digits

    # Check if the calculated sum of powers is equal to the original number
    return sum_of_powers == num
# Example usage
number = 153
if is_armstrong_number(number):
    print(f"{number} is an Armstrong number.")
else:
    print(f"{number} is not an Armstrong number.")
```

Output:

```
153 is an Armstrong number.
```

Observation:

1.The program correctly defines a function is_armstrong_number() to check whether a given number is an Armstrong (narcissistic) number.

2.The input number is converted into a string to easily iterate through each digit, improving code simplicity and readability.

3.The number of digits is calculated using len(), which is essential for raising each digit to the correct power.

4.The program uses a loop to compute the sum of each digit raised to the power of the total number of digits, following the Armstrong number definition.

5.A boolean comparison (sum_of_powers == num) is used to determine the result, ensuring accurate and efficient validation.

6.The output clearly indicates whether the given number is an Armstrong number, improving user understanding and program clarity.

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:**

```
def quicksort(arr):
    """
    QuickSort is a divide-and-conquer algorithm that 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
    The sub-arrays are then sorted recursively.
    """
    if len(arr) <= 1:
        return arr  # Base case: arrays with 0 or 1 element are already sorted

    pivot = arr[len(arr) // 2]  # Choose the pivot element (middle element)
    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 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.
    """
    n = len(arr)
    for i in range(n):
        # Track if a swap was made; if no swaps occur, the array is sorted
        swapped = False
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:  # Compare adjacent elements
                arr[j], arr[j+1] = arr[j+1], arr[j]  # Swap if they are in the wrong order
                swapped = True
        if not swapped:
            break  # If no swaps were made, the array is sorted
    return arr
# Example usage
array1 = [64, 34, 25, 12, 22, 11, 90]
sorted_quick = quicksort(array1)
print("QuickSort Result:", sorted_quick)
array2 = [64, 34, 25, 12, 22, 11, 90]
sorted_bubble = bubblesort(array2)
print("BubbleSort Result:", sorted_bubble)
```

Output:

```
QuickSort Result: [11, 12, 22, 25, 34, 64, 90]
BubbleSort Result: [11, 12, 22, 25, 34, 64, 90]
```

Observation:

1.The program correctly implements QuickSort using the divide-and-conquer approach by selecting a pivot and partitioning the array into smaller sub-arrays.

2.Recursion is effectively used in QuickSort to sort the left and right sub-arrays until the base case of one or zero elements is reached.

3.The program also implements Bubble Sort, which repeatedly compares and swaps adjacent elements to sort the array.

4.An optimization is included in Bubble Sort using a swapped flag, allowing the algorithm to terminate early if the array becomes sorted.

5.Both sorting algorithms produce the same correctly sorted output, verifying the correctness of the implementations.

6.The program clearly prints the results of QuickSort and Bubble Sort, enabling easy comparison of both algorithms.

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
class RecommendationSystem:
    def __init__(self, user_preferences, item_database):
        """
        Initialize the recommendation system with user preferences and item database.

        :param user_preferences: A dictionary containing user preferences (e.g., {'genre': 'sci-fi', 'price_range': 'low'})
        :param item_database: A list of dictionaries representing items (e.g., [{'name': 'Item1', 'genre': 'sci-fi', 'price': 10}, ...])
        """
        self.user_preferences = user_preferences
        self.item_database = item_database

    def recommend(self):
        """
        Generate recommendations based on user preferences.

        :return: A list of recommended items with reasons for each suggestion.
        """
        recommendations = []

        for item in self.item_database:
            reasons = []
            # Check genre preference
            if 'genre' in self.user_preferences and item['genre'] == self.user_preferences['genre']:
                reasons.append(f"Matches preferred genre: {item['genre']}")
            # Check price range preference
            if 'price_range' in self.user_preferences:
                if self.user_preferences['price_range'] == 'low' and item['price'] < 20:
                    reasons.append(f"Within preferred price range: ${item['price']}")
                elif self.user_preferences['price_range'] == 'high' and item['price'] >= 20:
                    reasons.append(f"Within preferred price range: ${item['price']}")

            # If there are reasons, add the item to recommendations
            if reasons:
                recommendations.append({
                    'item': item,
                    'reasons': reasons
                })

        return recommendations

# Example usage
user_preferences = {
    'genre': 'sci-fi',
    'price_range': 'low'
}
item_database = [
    {'name': 'Space Adventure', 'genre': 'sci-fi', 'price': 15},
    {'name': 'Romantic Getaway', 'genre': 'romance', 'price': 25},
    {'name': 'Sci-Fi Epic', 'genre': 'sci-fi', 'price': 30},
    {'name': 'Budget Sci-Fi', 'genre': 'sci-fi', 'price': 10}
]
rec_system = RecommendationSystem(user_preferences, item_database)
recommendations = rec_system.recommend()
for rec in recommendations:
    print(f"Recommended Item: {rec['item']['name']}")
    for reason in rec['reasons']:
        print(f" - Reason: {reason}")
```

Output:

```
Recommended Item: Space Adventure
 - Reason: Matches preferred genre: sci-fi
 - Reason: Within preferred price range: $15
Recommended Item: Sci-Fi Epic
 - Reason: Matches preferred genre: sci-fi
Recommended Item: Budget Sci-Fi
 - Reason: Matches preferred genre: sci-fi
 - Reason: Within preferred price range: $10
```

Observation:

1.The program implements a rule-based recommendation system that suggests items based on user preferences such as genre and price range.
2.User preferences and item details are stored using dictionaries and lists, making the data structure simple and easy to understand.
3.The recommendation logic checks multiple conditions (genre and price) to filter relevant items from the database.
4.For every recommended item, the system provides clear reasons for the suggestion, improving transparency and explainability.
5.Items are added to the recommendation list only when at least one preference rule is satisfied, ensuring relevant outputs.
6.The final output displays recommended items along with their matching reasons, enhancing user clarity and trust in the system.