

## **ASSIGNMENT-5.1**

G.Srinidhi

2303A53023

B-46

### **Lab 5: Ethical Foundations – Responsible AI Coding Practices**

#### **Lab Objectives:**

- To explore the ethical risks associated with AI-generated code.
- To recognize issues related to security, bias, transparency, and copyright.
- To reflect on the responsibilities of developers when using AI tools in software development.
- To promote awareness of best practices for responsible and ethical AI coding.

#### **Lab Outcomes (LOs):**

After completing this lab, students will be able to:

- Identify and avoid insecure coding patterns generated by AI tools.
- Detect and analyze potential bias or discriminatory logic in AI-generated outputs.
- Evaluate originality and licensing concerns in reused AI-generated code.
- Understand the importance of explainability and transparency in AI-assisted programming.
- Reflect on accountability and the human role in ethical AI coding practices.

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

**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"Weather in {city}: {weather_data['weather'][0]
['description']}, Temperature:{weather_data['main']
['temp']}°C")

    except Exception as e:

        print(f"Error fetching weather data: {e}")
```

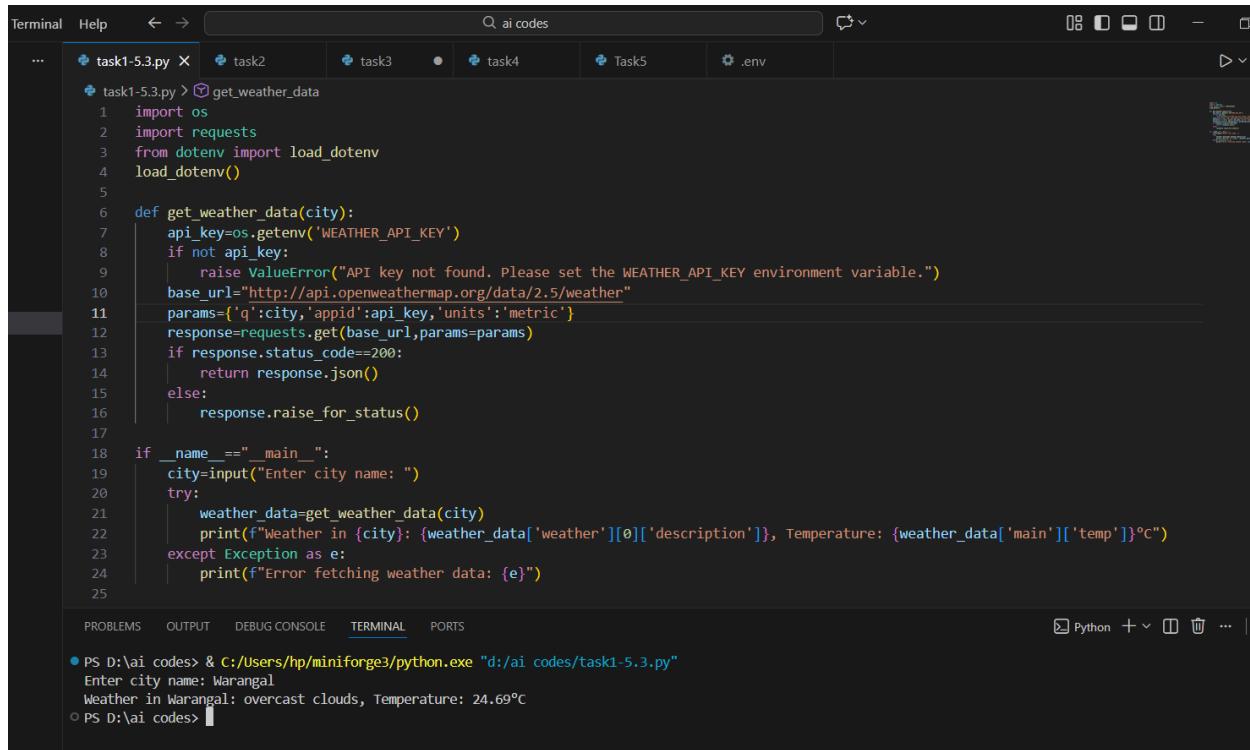
## **OUTPUT:**

Enter city name: warangal

Weather in warangal: clear sky, Temperature: 24.69C

## **Expected Output:**

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



```
task1-5.3.py > get_weather_data
1 import os
2 import requests
3 from dotenv import load_dotenv
4 load_dotenv()
5
6 def get_weather_data(city):
7     api_key=os.getenv('WEATHER_API_KEY')
8     if not api_key:
9         raise ValueError("API key not found. Please set the WEATHER_API_KEY environment variable.")
10    base_url="http://api.openweathermap.org/data/2.5/weather"
11    params={'q':city,'appid':api_key,'units':'metric'}
12    response=requests.get(base_url,params=params)
13    if response.status_code==200:
14        return response.json()
15    else:
16        response.raise_for_status()
17
18 if __name__ == "__main__":
19    city=input("Enter city name: ")
20    try:
21        weather_data=get_weather_data(city)
22        print(f"Weather in {city}: {weather_data['weather'][0]['description']}, Temperature: {weather_data['main']['temp']}°C")
23    except Exception as e:
24        print(f"Error fetching weather data: {e}")
25
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS D:\ai codes> & C:/Users/hp/miniforge3/python.exe "d:/ai codes/task1-5.3.py"  
Enter city name: Warangal  
Weather in Warangal: overcast clouds, Temperature: 24.69°C  
PS D:\ai codes>

## 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:

dentified privacy risks.

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

### PROMPT:

## **#Generate a Python script to store user name, email, and password in a file**

### **CODE:**

```
import os

import json

def save_user_data(filename="user_data.json"):

    name = input("Enter your name: ")

    email = input("Enter your email: ")

    password = input("Enter your password: ")

    user_data = {

        "name": name,

        "email": email, "password": password}

    with open(filename, "w") as file:

        json.dump(user_data, file, indent=4)

        print(f"User data saved to {filename}")

def load_user_data(filename="user_data.json"):

    if not os.path.exists(filename):

        print(f"File {filename} not found")

        return None

    with open(filename, "r") as file:
```

```

user_data = json.load(file)

return user_data

if __name__ == "__main__":
    save_user_data()

    data = load_user_data()

    print("Loaded data:", data)

```

```

n Terminal Help ← → 🔍 ai codes 08 □ □
... task1-5.3.py task2 .env
task2 > ...
4 def save_user_data(filename="user_data.json"):
5
6     name = input("Enter your name: ")
7     email = input("Enter your email: ")
8     password = input("Enter your password: ")
9
10    user_data = {
11        "name": name,
12        "email": email,
13        "password": password
14    }
15
16    with open(filename, "w") as file:
17        json.dump(user_data, file, indent=4)
18
19    print(f"User data saved to {filename}")
20
21 def load_user_data(filename="user_data.json"):
22     """Load user data from file"""
23
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python + ⌂
● PS D:\ai codes> & C:/Users/hp/miniforge3/python.exe "d:/ai codes/task2"
Enter your name: Humera Mahaveen
Enter your email: humeramahaveen@gmail.com
Enter your password: humera20
User data saved to user_data.json
Loaded data: {'name': 'Humera Mahaveen', 'email': 'humeramahaveen@gmail.com', 'password': 'humera20'}
○ PS D:\ai codes> []

```

## Task Description #3 (Transparency in Algorithm Design)

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

### Instructions:

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

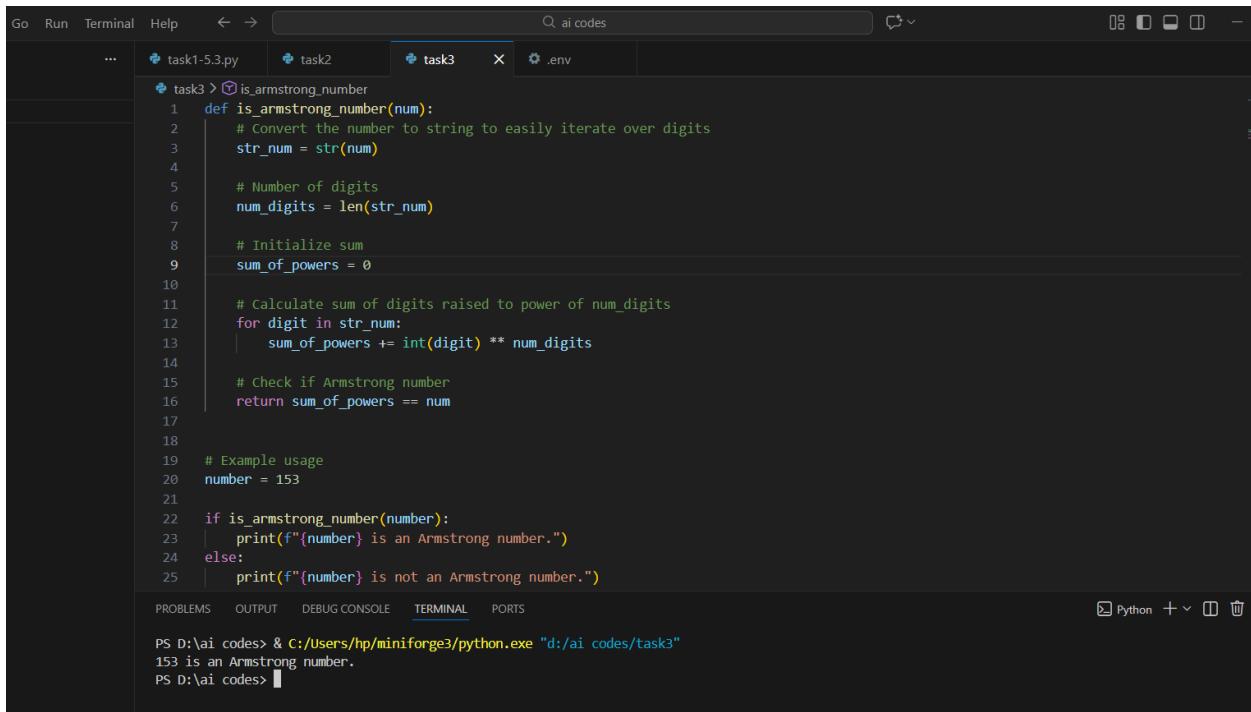
## Expected Output:

Transparent, commented code.

Correct, easy-to-understand explanation.

## PROMPT:

**#Generate a Python function to check if a number is an Armstrong number and explain the code line by line.**



The screenshot shows a code editor interface with a dark theme. The top menu bar includes 'Go', 'Run', 'Terminal', 'Help', and several icons. A search bar at the top right contains the text 'Q ai codes'. Below the menu, there are tabs for 'task1-5.3.py', 'task2', 'task3' (which is currently selected), and '.env'. The main code area displays the following Python script:

```
task3 > is_armstrong_number
1 def is_armstrong_number(num):
2     # Convert the number to string to easily iterate over digits
3     str_num = str(num)
4
5     # Number of digits
6     num_digits = len(str_num)
7
8     # Initialize sum
9     sum_of_powers = 0
10
11    # Calculate sum of digits raised to power of num_digits
12    for digit in str_num:
13        sum_of_powers += int(digit) ** num_digits
14
15    # Check if Armstrong number
16    return sum_of_powers == num
17
18
19    # Example usage
20    number = 153
21
22    if is_armstrong_number(number):
23        print(f"{number} is an Armstrong number.")
24    else:
25        print(f"{number} is not an Armstrong number.")

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
```

The terminal below the code shows the output of running the script with the number 153, confirming it is an Armstrong number. The Python extension icon is visible in the bottom right corner of the interface.

## Task Description #4 (Transparency in Algorithm Comparison)

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

## 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:**

```
def bubble_sort(arr):
    n=len(arr)
    for i in range(n):
        swapped=False
        for j in range(0,n-i-1):
            if arr[j]>arr[j+1]:
                arr[j],arr[j+1]=arr[j+1],arr[j]
                swapped=True
        if not swapped:
            Break
    return arr

def quick_sort(arr):
    if len(arr)<=1:
        return arr
    pivot=arr[len(arr)//2]
```

```
left=[x for x in arr if x<pivot]

middle=[x for x in arr if x==pivot]

right=[x for x in arr if x>pivot]

return quick_sort(left)+middle+quick_sort(right)

if name=="main":

test_arr=[64,34,25,12,22,11,90]

print("Original:",test_arr)

print("Bubble Sort:",bubble_sort(test_arr.copy()))

print("Quick Sort:",quick_sort(test_arr.copy()))
```

The screenshot shows a code editor with multiple tabs at the top. The active tab is 'task4' which contains the following Python code:

```
def bubble_sort(arr):
    """
    Bubble Sort: Repeatedly steps through the list, compares adjacent elements,
    and swaps them if they're in wrong order. Largest elements "bubble" to end.
    Time: O(n^2), Space: O(1)
    """
    n = len(arr)
    for i in range(n):
        swapped = False
        # Each pass moves the largest unsorted element to its position
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
                swapped = True
        # Optimization: stop if no swaps occur
        if not swapped:
            break
    return arr

def quick_sort(arr):
    """
    Quick Sort: Divide-and-conquer algorithm that partitions array around a pivot,
    then recursively sorts left and right partitions. Much faster in practice.
    Time: O(n log n) average, O(n^2) worst case, Space: O(log n)
    """
```

Below the code editor, there is a navigation bar with tabs: PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is underlined), and PORTS. The TERMINAL tab shows the following command-line output:

- PS D:\ai codes> & C:/Users/hp/miniforge3/python.exe "d:/ai codes/task4"
- Original: [64, 34, 25, 12, 22, 11, 90]
- Bubble Sort: [11, 12, 22, 25, 34, 64, 90]
- Quick Sort: [11, 12, 22, 25, 34, 64, 90]

At the bottom left, there is a status bar with the text: PS D:\ai codes> [ ]

## **Task Description #5 (Transparency in AI Recommendations)**

**Task:** Use AI to create a product recommendation system.

## **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:**

```
import random
```

```
def recommend_items(user_preferences,items):
```

```
recommendations=[]

for item in items:

    score=0

    reasons=[]

    for preference in user_preferences:

        if preference in item['tags']:

            score+=1

            reasons.append(f"Matches your preference for {preference}.")

    if score>0:

        recommendations.append({'item':item['name'],'score':score,'reasons':reasons})

recommendations.sort(key=lambda x:x['score'],reverse=True)

return recommendations

user_preferences=['action','comedy','sci-fi']

items=[

    {'name':'Movie A','tags':['action','thriller']},

    {'name':'Movie B','tags':['comedy','romance']},

    {'name':'Movie C','tags':['sci-fi','adventure']},

    {'name':'Movie D','tags':['drama','biography']}]

]
```

```
recommendations=recommend_items(user_preferences,items)
```

```
for rec in recommendations:
```

```
    print(f"Recommended: {rec['item']} (Score: {rec['score']})")
```

```
    for reason in rec['reasons']:
```

```
        print(f" - {reason}")
```

The screenshot shows a code editor interface with a dark theme. The top navigation bar includes 'Terminal', 'Help', and several tabs labeled 'task1-5.3.py', 'task2', 'task3', 'task4', 'Task5' (which is the active tab), and '.env'. Below the tabs is a search bar with the placeholder 'ai codes'. The main workspace contains the following Python code:

```
task1-5.3.py task2 task3 task4 Task5 .env
...
# Task5 > recommend_items
1 import random
2 def recommend_items(user_preferences,items):
3     recommendations=[]
4     for item in items:
5         score=0
6         reasons=[]
7         for preference in user_preferences:
8             if preference in item['tags']:
9                 score+=1
10            reasons.append(f"Matches your preference for {preference}.")
11        if score>0:
12            recommendations.append({'item':item['name'], 'score':score, 'reasons':reasons})
13    recommendations.sort(key=lambda x:x['score'],reverse=True)
14    return recommendations
15 user_preferences=[{'action','comedy','sci-fi']}
16 items=[
17     {'name': 'Movie A', 'tags': ['action', 'thriller']},
18     {'name': 'Movie B', 'tags': ['comedy', 'romance']},
19     {'name': 'Movie C', 'tags': ['sci-fi', 'adventure']},
20     {'name': 'Movie D', 'tags': ['drama', 'biography']}
21 ]
22 recommendations=recommend_items(user_preferences,items)
23 for rec in recommendations:
24     print(f"Recommended: {rec['item']} (Score: {rec['score']})")
25     for reason in rec['reasons']:
26         print(f" - {reason}")

```

The terminal below the code editor shows the execution of the script and its output:

```
PS D:\ai codes> & C:/Users/hp/miniforge3/python.exe "d:/ai codes/Task5"
Recommended: Movie A (Score: 1)
 - Matches your preference for action.
Recommended: Movie B (Score: 1)
 - Matches your preference for comedy.
Recommended: Movie C (Score: 1)
 - Matches your preference for sci-fi.
```