

## **Assignment 5.1**

---

**Course Title: Ethical Foundations – Responsible AI**

**Lab Title: Responsible AI Coding Practices**

**Lab No: 5**

**Week: 3**

**Day: Monday**

---

**Student Name: Pochampalli Akshith**

**Roll No: 2303A54015**

---

### **Aim**

To understand ethical risks in AI-generated code and apply responsible AI coding practices focusing on privacy, security, transparency, and accountability.

---

### **Objectives**

- Identify insecure coding patterns generated by AI
  - Analyze privacy and security risks
  - Ensure transparency and explainability in algorithms
  - Understand developer responsibility in AI-assisted programming
- 

### **Tools Used**

- VS Code
- 

### **Task-Wise Implementation**

---

#### **Task 1: Privacy in API Usage**

##### **Problem Statement**

Generate a Python program to fetch weather data securely without exposing API keys.

##### **AI Prompt Used**

Generate a Python program to fetch weather data from a weather API without hardcoding the API key.

Use environment variables to store and access the API key securely.

Include basic error handling.

### Observation

- AI-generated code avoided hardcoded API keys
- Environment variables were used for security

### Secure Code

```
def get_weather_data(city_name):
    # Set the API key
    api_key = os.getenv("WEATHER_API_KEY", "your_api_key")
    url = f"https://api.weatherapi.com/v1/current.json?q={city_name}&key={api_key}"

    # Make the API request
    response = requests.get(url, params=params, timeout=5)

    # Check if the request was successful
    response.raise_for_status()

    # Parse and return the JSON response
    weather_data = response.json()
    return weather_data

except requests.exceptions.ConnectionError:
    print("Sorry, unable to connect to the weather API. Check your internet connection.")
    return None

# Argument parser
parser = argparse.ArgumentParser(description="Get weather data for a city")
parser.add_argument("city", type=str, help="City name")

if __name__ == "__main__":
    args = parser.parse_args()
    print(f"Getting weather for {args.city}...")

    # Check Armstrong numbers
    def is_armstrong(n):
        digits = [int(d) for d in str(n)]
        sum_of_powers = sum([d**len(digits) for d in digits])
        return sum_of_powers == n

    for i in range(1, 1000):
        if is_armstrong(i):
            print(f"{i} is an Armstrong number.")

    # Interactive mode
    while True:
        user_input = input("Enter a number to check (or 'quit' to exit): ")
        if user_input.lower() == "quit":
            break
        try:
            num = int(user_input)
            if is_armstrong(num):
                print(f"{num} is an Armstrong number!")
            else:
                print(f"{num} is not an Armstrong number.")
        except ValueError:
            print("Please enter a valid integer or 'quit' to exit.")
```

### Ethical Analysis

Using environment variables protects sensitive credentials and prevents unauthorized misuse.

---

## Task 2: Privacy & Security in File Handling

### Problem Statement

Store user data securely without exposing sensitive information.

### AI Prompt Used

Generate a Python program to store user details (name, email, password) securely.

Do not store passwords in plain text.

Use hashing for password storage and explain why this approach is secure.

### Privacy Risk Identified

- Plain-text password storage is insecure

### Secure Code

## Ethical Analysis

Hashing ensures passwords cannot be recovered even if the file is leaked.

### Task 3: Transparency in Algorithm Design

## Problem Statement

Design an Armstrong number checking program with clear explanation.

## AI Prompt Used

Generate a Python function to check whether a given number is an Armstrong number.

Add clear comments to every important line of code.

Also explain the code line-by-line in simple terms.

## Implementation

```
#!/usr/bin/env python3
# ARMSTRONG NUMBER CHECKER
# A program to check if a given number is an Armstrong number or not.
# An Armstrong number is a number that is equal to the sum of its own digits each raised to the power of the number of digits.

def is_armstrong_number(num):
    result = 0
    status = "Not Armstrong"
    for digit in str(num):
        result += int(digit) ** len(str(num))
    if result == num:
        status = "Armstrong"
    print(f"\n{status} - Checking individual numbers")
    print(f"\n{num} is {status} if result else 'Not Armstrong'")

def main():
    print("1. Armstrong analysis")
    print("2. Detailed analysis for 1000")
    print("3. Detailed analysis for 10000")
    print("4. Detailed analysis for 100000")
    print("5. Interactive Mode")
    choice = input("Enter a number to check (or 'quit' to exit): ")
    if choice == '1':
        is_armstrong_number(int(choice))
    elif choice == '2':
        is_armstrong_number(1000)
    elif choice == '3':
        is_armstrong_number(10000)
    elif choice == '4':
        is_armstrong_number(100000)
    elif choice == '5':
        while True:
            user_input = input("Enter a number to check (or 'quit' to exit): ")
            if user_input == 'quit':
                break
            is_armstrong_number(int(user_input))

if __name__ == "__main__":
    main()
```

## Transparency Evaluation

The explanation clearly matches the program logic, ensuring understandability.

## Task 4: Transparency in Algorithm Comparison

### Problem Statement

Implement and compare two sorting algorithms.

### AI Prompt Used

Generate Python code for Bubble Sort and Quick Sort.

Include step-by-step comments explaining how each algorithm works.

Compare both algorithms in terms of logic, time complexity, and efficiency.

```
#!/usr/bin/env python3
# BUBBLE SORT vs QUICK SORT
# A program to compare the performance of Bubble Sort and Quick Sort on various datasets.

def test_sorting_algorithm():
    """Test both algorithms with various datasets."""
    print("1. Test case 1: Random array")
    print("2. Test case 2: Already sorted")
    print("3. Test case 3: Reverse sorted")
    print("4. Test case 4: All NaNs sorted array (test4)"""
    print("5. Interactive Mode")
    choice = input("Enter a number to check (or 'quit' to exit): ")
    if choice == '1':
        arr1 = [42, 13, 25, 33, 45, 56]
        arr2 = [13, 25, 33, 45, 42]
        print("Original array: ", arr1)
        print("Bubble sort result: ", bubble_sort(arr1))
        print("Quick sort result: ", quick_sort(arr1))
    elif choice == '2':
        arr1 = [1, 2, 3, 4, 5]
        arr2 = [5, 4, 3, 2, 1]
        print("Original array: ", arr2)
        print("Bubble sort result: ", bubble_sort(arr2))
        print("Quick sort result: ", quick_sort(arr2))
    elif choice == '3':
        arr1 = [5, 4, 3, 2, 1]
        arr2 = [1, 2, 3, 4, 5]
        print("Original array: ", arr1)
        print("Bubble sort result: ", bubble_sort(arr1))
        print("Quick sort result: ", quick_sort(arr1))
    elif choice == '4':
        arr1 = [float('NaN'), float('NaN'), float('NaN'), float('NaN'), float('NaN')]
        arr2 = [float('NaN'), float('NaN'), float('NaN'), float('NaN'), float('NaN')]
        print("Original array: ", arr1)
        print("Bubble sort result: ", bubble_sort(arr1))
        print("Quick sort result: ", quick_sort(arr1))
    elif choice == '5':
        while True:
            user_input = input("Enter a number to check (or 'quit' to exit): ")
            if user_input == 'quit':
                break
            arr1 = [42, 13, 25, 33, 45, 56]
            arr2 = [13, 25, 33, 45, 42]
            print("Original array: ", arr1)
            print("Bubble sort result: ", bubble_sort(arr1))
            print("Quick sort result: ", quick_sort(arr1))

if __name__ == "__main__":
    test_sorting_algorithm()
```

## Comparison

### Algorithm Time Complexity Efficiency

Bubble Sort  $O(n^2)$  Low

Quick Sort  $O(n \log n)$  High

---

## Task 5: Transparency in AI Recommendations

### Problem Statement

Create an explainable recommendation system.

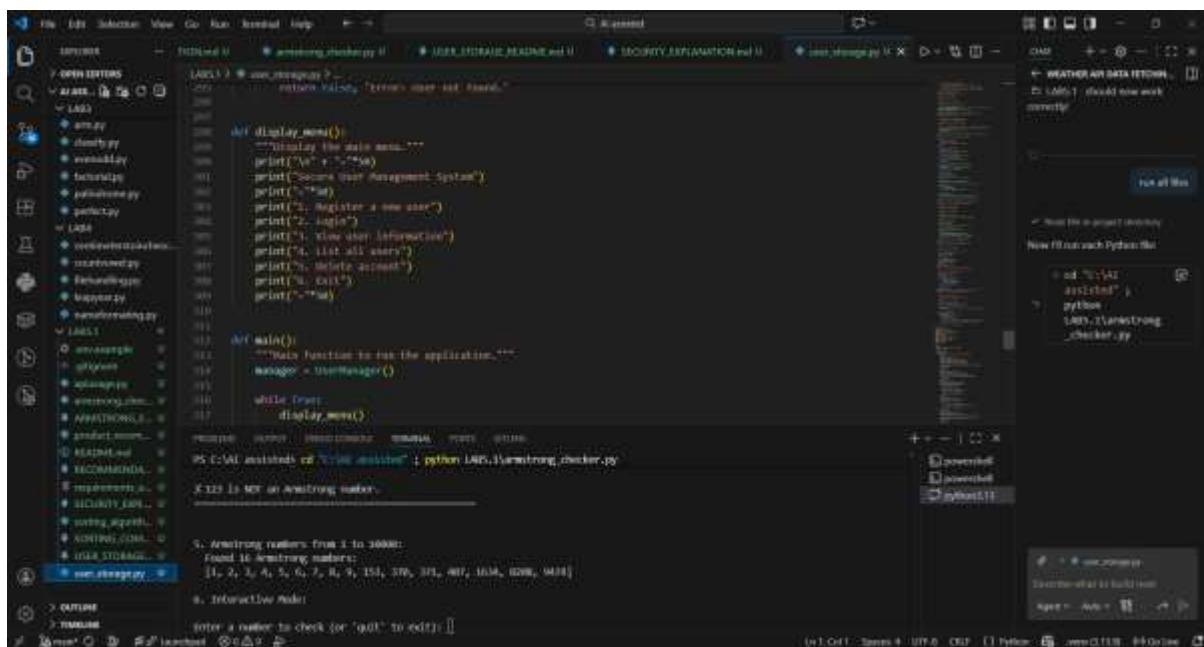
### AI Prompt Used

Generate a simple product recommendation system in Python.

For each recommended product, also provide a clear explanation of why it was recommended.

Ensure the recommendations are explainable and transparent.

### Implementation



The screenshot shows the PyCharm IDE interface with several open files. The main editor contains a Python script named `user_manager.py`. The code defines a `UserManager` class with methods for displaying menu options and managing user accounts. A terminal window at the bottom shows the command `python L05_Authentication_and_Ui.py` being run, and the output shows the program's menu and a check for Armstrong numbers.

```
def display_menu():
    """Display the main menu."""
    print("1. Create account")
    print("2. Log in")
    print("3. Register a new user")
    print("4. Log out")
    print("5. View user information")
    print("6. List all users")
    print("7. Delete account")
    print("8. Exit")
    print("9. Armstrong numbers")

def main():
    """Main function to run the application."""
    manager = UserManager()
    while True:
        display_menu()

if __name__ == "__main__":
    main()

# Armstrong numbers from 1 to 10000:
# Find 10 Armstrong numbers:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 153, 370, 371, 407, 1634, 6890, 9474]
```

### Transparency Evaluation

Each recommendation includes a reason, making the system explainable.

---

## Result

All tasks were implemented successfully using ethical AI coding practices with proper privacy, security, and transparency.

---

## **Conclusion**

AI-generated code must be reviewed by developers to ensure ethical compliance. Human accountability is essential in responsible AI usage.