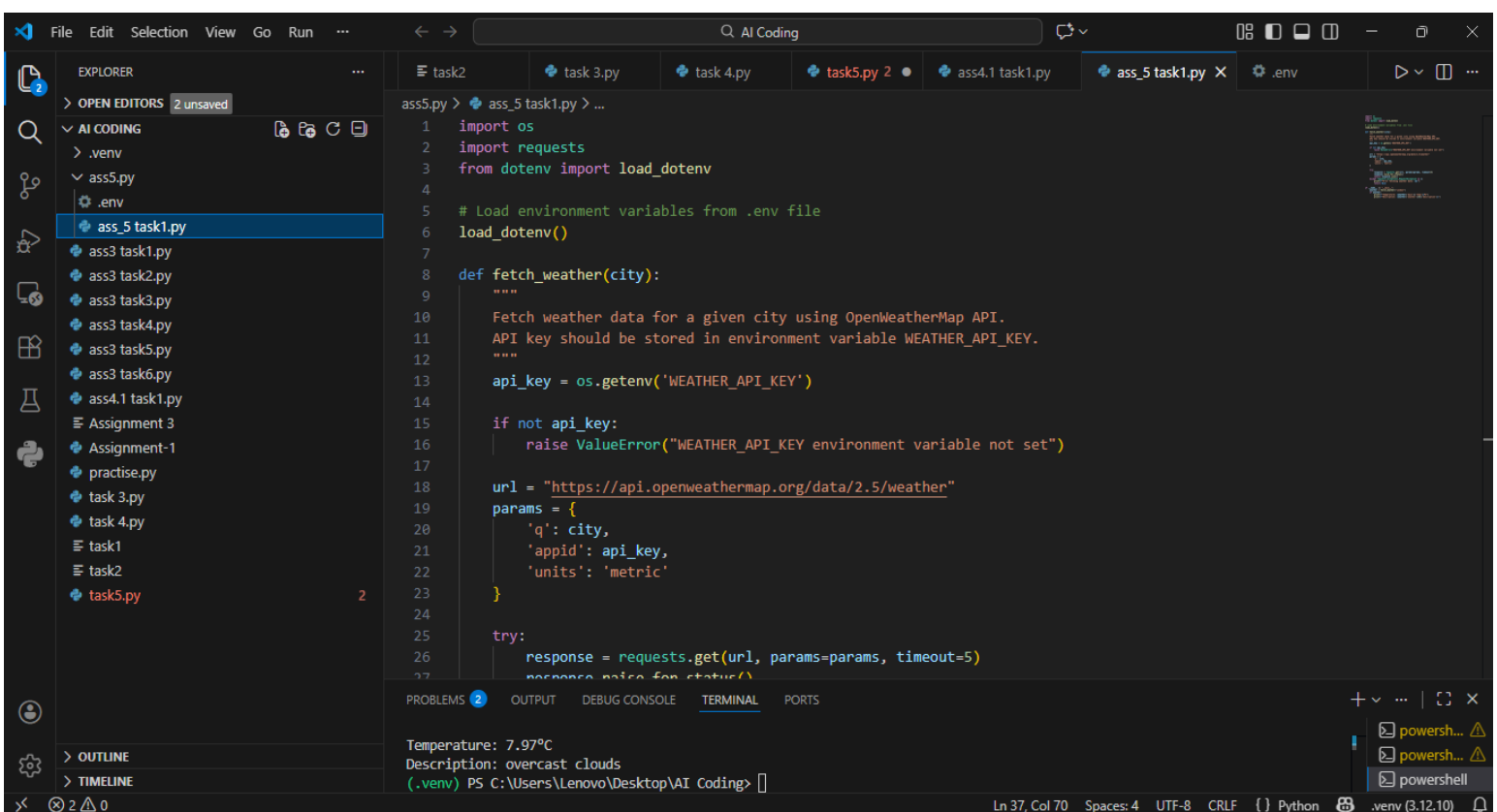


SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: B.Tech		Assignment Type: Lab	Academic Year: 2025-2026
Course Code	23CS002PC304	Course Title	AI Assisted Coding
Year/Sem	III/II	Regulation	R23
Date and Day of Assignment	Week 3 – Monday	Batch	23CSBTB48
Name	B.SUSHMA	Hall Ticket No	2303A54040
Assignment Number: 5.1			
Q.No.	Question		
	<p>Lab 5: Ethical Foundations – Responsible AI Coding Practices Lab</p> <p>Objectives:</p> <ul style="list-style-type: none"> To explore the ethical risk 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. <p>Lab Outcomes (LOs):</p> <p>After completing this lab, students will be able to:</p> <ul style="list-style-type: none"> 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. <p>Task Description #1 (Privacy in API Usage)</p> <p>Task: Use an AI tool to generate a Python program that connects to a weather API.</p> <p><i>Prompt:</i> <i>"Write a Python program using an AI tool to securely fetch weather data from an API without exposing sensitive API keys in the code."</i></p> <p>Expected Output:</p> <ul style="list-style-type: none"> Original AI code (check if keys are hard coded). Secure version using environment variables. 		



EXPLANATION : From this task, I understood that while using APIs in programs, **API keys should never be hardcoded** directly into the source code. If API keys are written openly in the program, anyone who accesses the file can misuse them, which can lead to security and privacy issues. This task helped me learn that a **secure approach is to store API keys in environment variables** and access them during runtime. This way, sensitive credentials remain hidden, and even if the code is shared or uploaded to a repository, the API key is protected. As a developer, it is my responsibility to ensure that confidential information is handled securely.

TaskDescription#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:

"Create a Python script using an AI tool to save user information (name, email, password) in a file. Evaluate the security of the password storage and update the script to store the password securely using hashing."

Code :

The screenshot displays the Visual Studio Code interface. The Explorer sidebar on the left shows a project structure with files like `task2.py`, `task3.py`, `task4.py`, `task5.py`, and `ass5_task2.py`. The main editor window shows the code for `ass5_task2.py`, which includes imports for `hashlib`, `json`, `os`, and `Path`. It defines a `hash_password` function using `hashlib.sha256` and a `save_user_details` function that writes user information to a `users.json` file. The terminal window at the bottom shows the command prompt output, including the activation of the `.venv` environment and the successful execution of the script, which prints "User John Doe saved securely!" and "User Jane Smith saved securely!".

```
1 import hashlib
2 import json
3 import os
4 from pathlib import Path
5
6 def hash_password(password):
7     """Hash a password using SHA-256"""
8     return hashlib.sha256(password.encode()).hexdigest()
9
10 def save_user_details(name, email, password, filename="users.json"):
11     """Save user details with hashed password to a JSON file"""
12     hashed_password = hash_password(password)
13
14     user_data = {
15         "name": name,
16         "email": email,
17         "password_hash": hashed_password
18     }
19
20     users = []
21     if os.path.exists(filename):
22         with open(filename, 'r') as f:
```

Terminal Output:

```
PS C:\Users\Lenovo\Desktop\AI Coding> & "C:\Users\Lenovo\Desktop\AI Coding\.venv\Scripts\Activate.ps1"
(.venv) PS C:\Users\Lenovo\Desktop\AI Coding> & "C:\Users\Lenovo\Desktop\AI Coding\.venv\Scripts\python.exe" "c:/Users/Leno
vo/Desktop/AI Coding/ass5.py/ass_5_task2.py"
User John Doe saved securely!
User Jane Smith saved securely!

Verifying password for John Doe:
False
(.venv) PS C:\Users\Lenovo\Desktop\AI Coding>
```

EXPLANATION : From this task, I learned that storing sensitive user information such as passwords in **plain text format** is **highly insecure**. If such files are accessed by unauthorized users, passwords can be easily read and misused, leading to serious security breaches. This task taught me the importance of using **password hashing techniques** instead of storing actual passwords. By converting passwords into hashed values with added salt, the original password cannot be retrieved even if the file is compromised. This ensures better privacy and protects users' data, highlighting the importance of secure coding practices when handling personal information.

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.

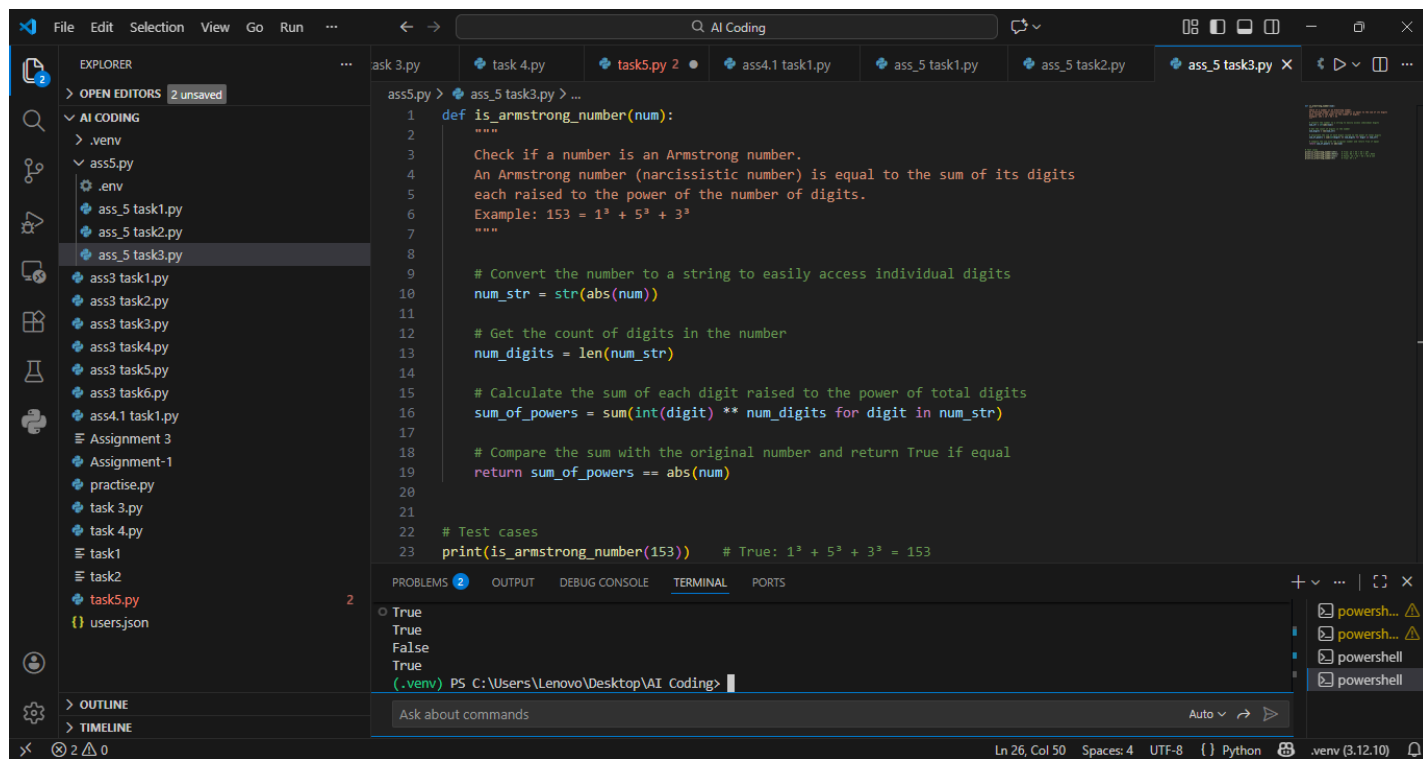
3. Expected Output:

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

Prompt:

"Ask an AI tool to generate a Python function that checks if a number is an Armstrong number. Include detailed comments explaining each line of the code and verify if the explanation matches the functionality."

Code :



```
1 def is_armstrong_number(num):
2     """
3     Check if a number is an Armstrong number.
4     An Armstrong number (narcissistic number) is equal to the sum of its digits
5     each raised to the power of the number of digits.
6     Example: 153 = 1³ + 5³ + 3³
7     """
8
9     # Convert the number to a string to easily access individual digits
10    num_str = str(abs(num))
11
12    # Get the count of digits in the number
13    num_digits = len(num_str)
14
15    # Calculate the sum of each digit raised to the power of total digits
16    sum_of_powers = sum(int(digit) ** num_digits for digit in num_str)
17
18    # Compare the sum with the original number and return True if equal
19    return sum_of_powers == abs(num)
20
21
22 # Test cases
23 print(is_armstrong_number(153)) # True: 1³ + 5³ + 3³ = 153
```

True
True
False
True

(.venv) PS C:\Users\Lenovo\Desktop\AI Coding>

EXPLANATION : From this task, I understood the importance of **transparency and explainability** in AI-generated code. Simply generating working code is not enough; the logic behind each step must be clear and understandable to humans. By generating an Armstrong number program with line-by-line comments, I learned how each part of the code contributes to the final result. This task helped me verify that the explanation correctly matches the actual functionality of the code. Clear explanations make the program easier to debug, maintain, and trust, especially when AI tools are used in development.

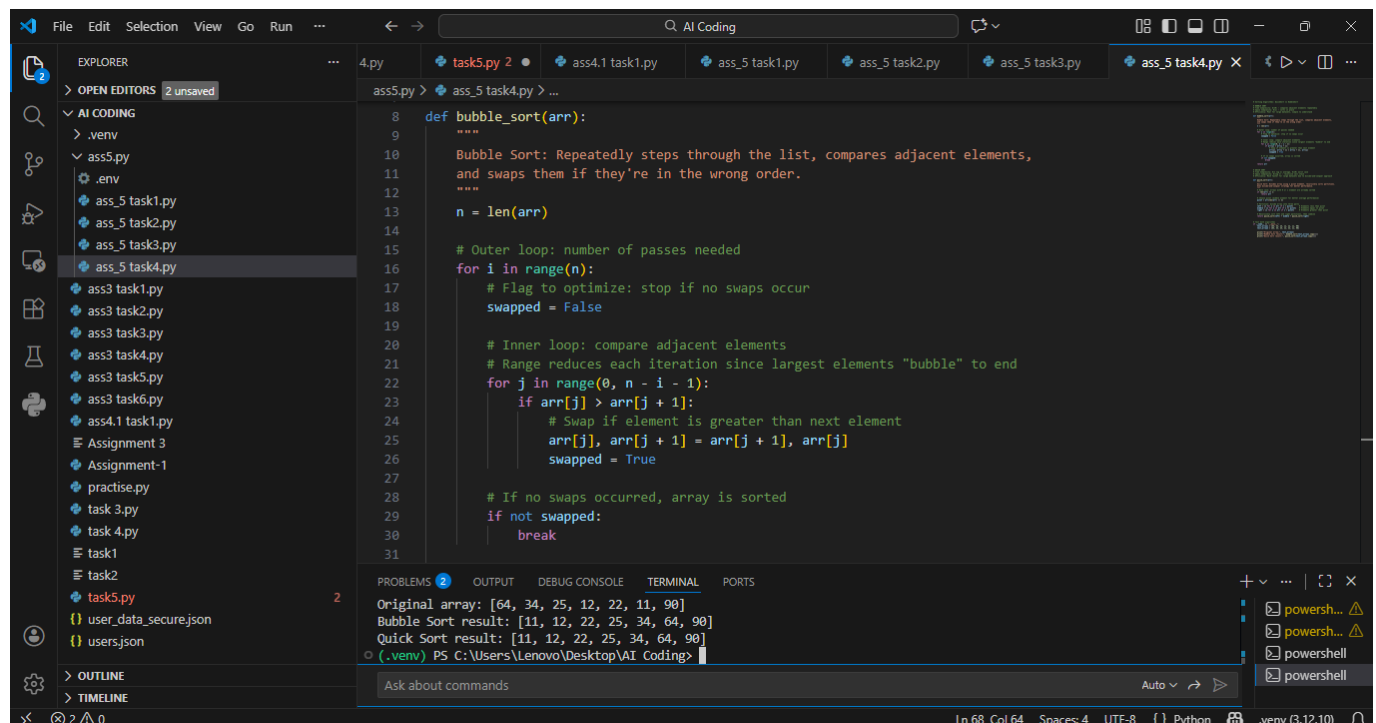
Task Description #4 (Transparency in Algorithm Comparison)

Task: Use AI to implement two sorting algorithms (e.g., Quick Sort and Bubble Sort).

Prompt: "Use an AI tool to implement two sorting algorithms, QuickSort and BubbleSort, in Python. Add comments to explain each step and highlight the differences in their logic and efficiency."

Expected Output:

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



The screenshot shows a VS Code editor with a file explorer on the left containing various Python files. The main editor window displays the code for a `bubble_sort` function. The code includes detailed comments explaining the algorithm's steps: repeatedly stepping through the list, comparing adjacent elements, and swapping them if they are in the wrong order. It also includes an optimization flag to stop if no swaps occur. The function is tested with an array `[64, 34, 25, 12, 22, 11, 90]`. The output at the bottom shows the original array, the result after Bubble Sort (`[11, 12, 22, 25, 34, 64, 90]`), and the result after Quick Sort (`[11, 12, 22, 25, 34, 64, 90]`).

```
8 def bubble_sort(arr):
9     """
10    Bubble Sort: Repeatedly steps through the list, compares adjacent elements,
11    and swaps them if they're in the wrong order.
12    """
13    n = len(arr)
14
15    # Outer loop: number of passes needed
16    for i in range(n):
17        # Flag to optimize: stop if no swaps occur
18        swapped = False
19
20        # Inner loop: compare adjacent elements
21        # Range reduces each iteration since largest elements "bubble" to end
22        for j in range(0, n - i - 1):
23            if arr[j] > arr[j + 1]:
24                # Swap if element is greater than next element
25                arr[j], arr[j + 1] = arr[j + 1], arr[j]
26                swapped = True
27
28        # If no swaps occurred, array is sorted
29        if not swapped:
30            break
31
```

Original array: [64, 34, 25, 12, 22, 11, 90]
Bubble Sort result: [11, 12, 22, 25, 34, 64, 90]
Quick Sort result: [11, 12, 22, 25, 34, 64, 90]

EXPLANATION : From this task, I learned that different algorithms can solve the same problem but with **different efficiency and performance**. Bubble Sort is simple and easy to understand, but it is inefficient for large datasets because it repeatedly compares adjacent elements. On the other hand, Quick Sort uses a **divide-and-conquer approach**, making it much faster and more efficient for large inputs. By comparing both algorithms with explanations, I understood how transparency in algorithm design helps developers choose the right solution based on performance requirements and use cases.

Task Description #5 (Transparency in AIR ecommendations)

Task: Use AI to create a product recommendation system.

Prompt: "Develop a product recommendation system using an AI tool that provides suggestions based on user preferences. Ensure the system explains the reasoning behind each recommendation."

Expected Output:

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

Code :

The screenshot shows a Visual Studio Code editor with a Python file named `ass_5_task5.py` open. The code defines a `ProductRecommendationSystem` class with a list of products and a `get_recommendations` method. The terminal output shows the execution of the script, which returns a recommendation for a Wireless Mouse with a rating of 4.2/5, explaining that it is within the budget and has a high rating.

```
1 import json
2 from typing import List, Dict
3
4 class ProductRecommendationSystem:
5     def __init__(self):
6         self.products = [
7             {"id": 1, "name": "Laptop", "category": "electronics", "price": 1000, "rating": 4.5},
8             {"id": 2, "name": "Wireless Mouse", "category": "electronics", "price": 30, "rating": 4.2},
9             {"id": 3, "name": "USB-C Cable", "category": "electronics", "price": 15, "rating": 4.0},
10            {"id": 4, "name": "Desk Lamp", "category": "furniture", "price": 50, "rating": 4.3},
11            {"id": 5, "name": "Ergonomic Chair", "category": "furniture", "price": 300, "rating": 4.6},
12            {"id": 6, "name": "Notebook", "category": "stationery", "price": 10, "rating": 4.1},
13        ]
14
15    def get_recommendations(self, preferences: Dict, num_recommendations: int = 3) -> List[Dict]:
16        """
17        Generate product recommendations based on user preferences.
18
19        Args:
20            preferences: Dict with keys like 'category', 'max_price', 'min_rating'
21            num_recommendations: Number of recommendations to return
22        """
```

Terminal Output:

```
(.venv) PS C:\Users\Lenovo\Desktop\AI Coding> & "C:\Users\Lenovo\Desktop\AI Coding\Scripts\python.exe" "c:/Users/Lenovo/Desktop/AI Coding/ass5.py/ass_5_task5.py"
Rating: 4.2/5
Why: Within your budget ($30) | High rating: 4.2/5
```

EXPLANATION : From this task, I understood that AI recommendation systems should not only suggest items but also **explain why a particular recommendation was made**. Without explanations, users may not trust or understand AI decisions. This task helped me realize the importance of **explainable AI**, where recommendations are based on user preferences, past behavior, or similar users' choices. Providing reasons improves transparency, builds user trust, and allows users to better understand how AI systems work, making AI-assisted applications more reliable and ethical.

