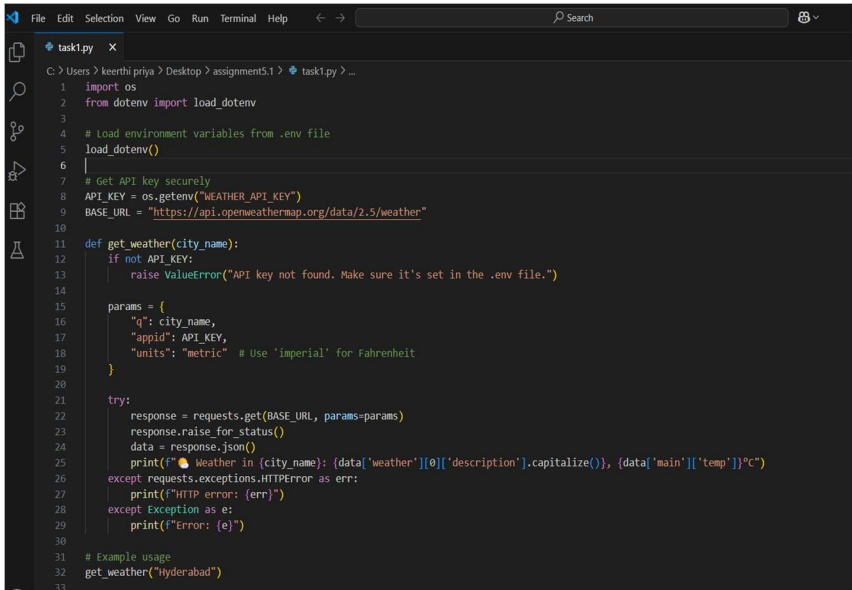
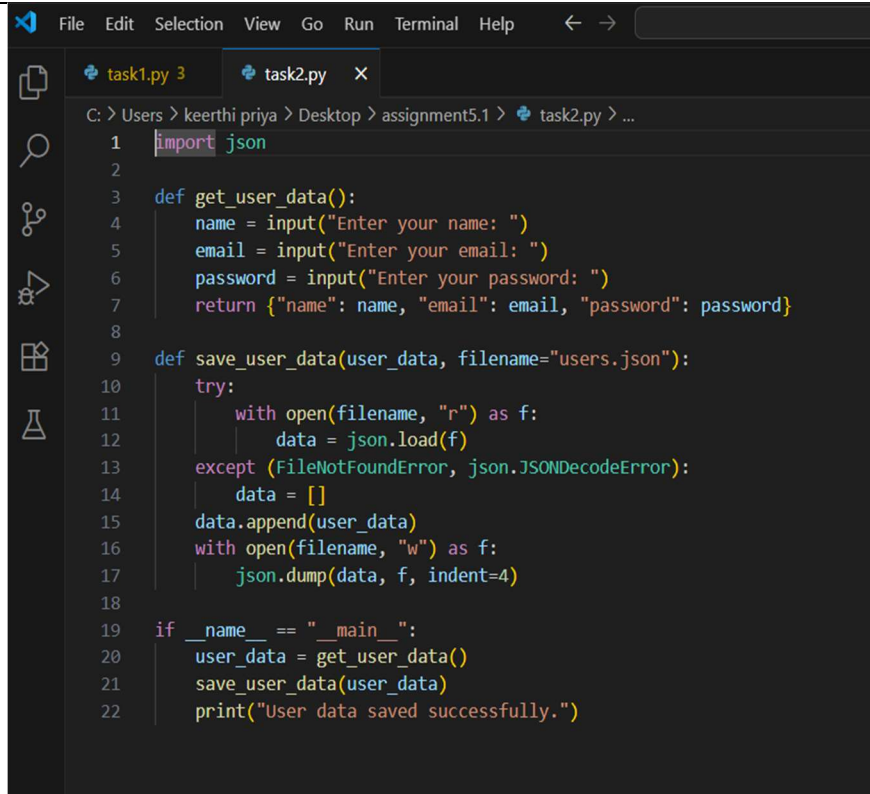


	<p style="text-align: center;">AI ASSISTED CODING</p> <p>NAME: J.KEERTHI PRIYA ROLL NO: 2403A510G4 ASSIGNMENT: 5.1</p>	
	<p>Task Description #1 (Privacy in API Usage)</p> <p>Task: Use an AI tool to generate a Python program that connects to a weather API.</p> <p>Prompt: <i>"Generate code to fetch weather data securely without exposing API keys in the code."</i></p> <p>Expected Output:</p> <ul style="list-style-type: none"> • Original AI code (check if keys are hardcoded). • Secure version using environment variables.  <p>Task Description #2 (Privacy & Security in File Handling)</p> <p>Task: Use an AI tool to generate a Python script that stores user data (name, email, password) in a file.</p> <p>Analyze: Check if the AI stores sensitive data in plain text or without encryption.</p> <p>Expected Output:</p> <ul style="list-style-type: none"> • Identified privacy risks. • Revised version with encrypted password storage (e.g., hashing). 	



```
File Edit Selection View Go Run Terminal Help
task1.py 3 task2.py X
C:\Users\keerthi priya\Desktop> assignment5.1> task2.py > ...
1 import json
2
3 def get_user_data():
4     name = input("Enter your name: ")
5     email = input("Enter your email: ")
6     password = input("Enter your password: ")
7     return {"name": name, "email": email, "password": password}
8
9 def save_user_data(user_data, filename="users.json"):
10     try:
11         with open(filename, "r") as f:
12             data = json.load(f)
13     except (FileNotFoundError, json.JSONDecodeError):
14         data = []
15     data.append(user_data)
16     with open(filename, "w") as f:
17         json.dump(data, f, indent=4)
18
19 if __name__ == "__main__":
20     user_data = get_user_data()
21     save_user_data(user_data)
22     print("User data saved successfully.")
```

Task Description #3 (Transparency in Algorithm Design)

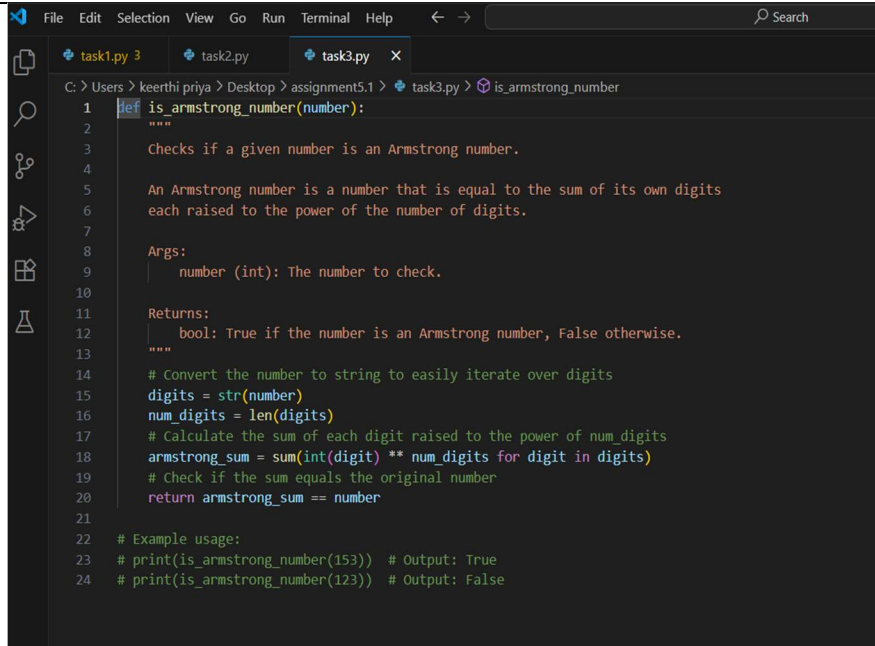
Objective: Use AI to generate an Armstrong number checking function with comments and explanations.

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.



```
File Edit Selection View Go Run Terminal Help
task1.py 3 task2.py task3.py x
C: > Users > keerthi priya > Desktop > assignment5.1 > task3.py > is_armstrong_number
1 def is_armstrong_number(number):
2     """
3     Checks if a given number is an Armstrong number.
4
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
8     Args:
9     | number (int): The number to check.
10
11     Returns:
12     | bool: True if the number is an Armstrong number, False otherwise.
13     """
14     # Convert the number to string to easily iterate over digits
15     digits = str(number)
16     num_digits = len(digits)
17     # Calculate the sum of each digit raised to the power of num_digits
18     armstrong_sum = sum(int(digit) ** num_digits for digit in digits)
19     # Check if the sum equals the original number
20     return armstrong_sum == number
21
22 # Example usage:
23 # print(is_armstrong_number(153)) # Output: True
24 # print(is_armstrong_number(123)) # Output: False
```

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.

```
File Edit Selection View Go Run Terminal Help ← → Search
task1.py 3 task2.py task3.py task4.py x
C:\Users\keerthi priya\Desktop> assignment5.1> task4.py > ...
1 # QuickSort and BubbleSort implementations in Python
2 # Both are sorting algorithms, but they differ in approach and efficiency.
3
4 # QuickSort Implementation
5 def quicksort(arr):
6     """
7     Sorts an array using the QuickSort algorithm.
8     QuickSort is a divide-and-conquer algorithm:
9     1. Select a 'pivot' element from the array.
10    2. Partition the array into two sub-arrays:
11        - Elements less than the pivot
12        - Elements greater than or equal to the pivot
13    3. Recursively apply the above steps to the sub-arrays.
14    4. Combine the sorted sub-arrays and the pivot.
15    """
16    if len(arr) <= 1:
17        # Base case: arrays with 0 or 1 element are already sorted
18        return arr
19    else:
20        pivot = arr[0] # Choose the first element as pivot
21        left = [x for x in arr[1:] if x < pivot] # Elements less than pivot
22        right = [x for x in arr[1:] if x >= pivot] # Elements greater or equal
23        # Recursively sort left and right, then combine
24        return quicksort(left) + [pivot] + quicksort(right)
25
26 # BubbleSort Implementation
27 def bubblesort(arr):
28     """
29     Sorts an array using the BubbleSort algorithm.
30     BubbleSort works by repeatedly swapping adjacent elements if they are in the wrong order:
31     1. Compare each pair of adjacent elements.
32     2. Swap them if they are in the wrong order.
33     3. Repeat the process for all elements until no swaps are needed.
34     4. Each pass 'bubbles' the largest unsorted element to its correct position.
35     """
36    n = len(arr)
37    for i in range(n):
38        # Track if any swaps are made in this pass
39        swapped = False
40        for j in range(0, n - i - 1):
41            if arr[j] > arr[j + 1]:
42                # Swap if elements are in wrong order
43                arr[j], arr[j + 1] = arr[j + 1], arr[j]
44                swapped = True
45        if not swapped:
46            # If no swaps, array is sorted
47            break
48    return arr
49
50 # Differences:
51 # - QuickSort uses recursion and divides the array, leading to faster average performance (O(n log n)).
52 # - BubbleSort uses repeated passes and swaps, making it slower (O(n^2)), but easier to understand and implement.
53
54 # Example usage:
55 if __name__ == "__main__":
56     data = [64, 34, 25, 12, 22, 11, 90]
57     print("Original:", data)
58     print("QuickSort:", quicksort(data.copy()))
59     print("BubbleSort:", bubblesort(data.copy()))
```

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.

```

1 import random
2
3 # Sample data: items and reasons
4 items = [
5     {"name": "The Pragmatic Programmer", "reason": "It offers practical advice for software development."},
6     {"name": "Clean Code", "reason": "It teaches principles for writing maintainable code."},
7     {"name": "Python Crash Course", "reason": "It's a beginner-friendly introduction to Python."},
8     {"name": "Automate the Boring Stuff with Python", "reason": "It shows how to use Python for everyday tasks."},
9     {"name": "Fluent Python", "reason": "It covers advanced Python features and best practices."}
10 ]
11
12 def recommend(n=3):
13     """Recommend n items with reasons."""
14     recommendations = random.sample(items, min(n, len(items)))
15     for item in recommendations:
16         print(f'Recommendation: {item["name"]}')
17         print(f'Reason: {item["reason"]}\n')
18
19 if __name__ == "__main__":
20     recommend()

```

```

1 import random
2
3 # Sample data: items and reasons
4 items = [
5     {"name": "The Pragmatic Programmer", "reason": "It offers practical advice for software development."},
6     {"name": "Clean Code", "reason": "It teaches principles for writing maintainable code."},
7     {"name": "Python Crash Course", "reason": "It's a beginner-friendly introduction to Python."},
8     {"name": "Automate the Boring Stuff with Python", "reason": "It shows how to use Python for everyday tasks."},
9     {"name": "Fluent Python", "reason": "It covers advanced Python features and best practices."}
10 ]
11
12 def recommend(n=3):
13     """Recommend n items with reasons."""
14     recommendations = random.sample(items, min(n, len(items)))
15     for item in recommendations:
16         print(f'Recommendation: {item["name"]}')
17         print(f'Reason: {item["reason"]}\n')
18
19 if __name__ == "__main__":
20     recommend()

```

Task Description #6 (Transparent Code Generation)

Task: Ask AI to generate a Python function for calculating factorial using recursion.

Prompt:

"Generate a recursive factorial function with comments that explain each line and a final summary of the algorithm's flow."

Expected Output:

- Fully commented code.
- Clear documentation of how recursion works.

Task Description #7 (Inclusiveness in Customer Support)

Code Snippet:

```
def support_reply(name, gender):
    if gender.lower() == "male":
        prefix = "Mr."
    else:
        prefix = "Mrs."
    return f"Dear {prefix} {name}, we have resolved your i
```

Task:

Regenerate the code so that support messages use neutral language (e.g., “Dear {name}”) and optionally accept preferred titles.

Expected Output:

- Neutral, user-friendly support responses.

```
def factorial(n):
    # Base case: if n is 0 or 1, return 1 (since 0! = 1! = 1)
    if n == 0 or n == 1:
        return 1
    # Recursive case: multiply n by factorial of (n-1)
    else:
        return n * factorial(n - 1)

# Summary:
# The function checks if n is 0 or 1 and returns 1.
# Otherwise, it calls itself with (n-1) and multiplies the result by n.
# This process repeats until the base case is reached, then the results are multiplied back up the call stack.
```

Note: Report should be submitted a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots

Evaluation Criteria:

Criteria	Max Marks
Transparency	1
Inclusiveness	0.5
Data security and Privacy	1
Total	2.5 Marks