ASSIGNMENT – 2.4

Name: R.Shashideepika

H.no:2303A51871

B-13

Task 1: Book Class Generation

❖ Scenario:

You are building a simple library management module.

❖ Task:

Use Cursor AI to generate a Python class Book with attributes title,

author,and a summary() method.

❖ Expected Output:

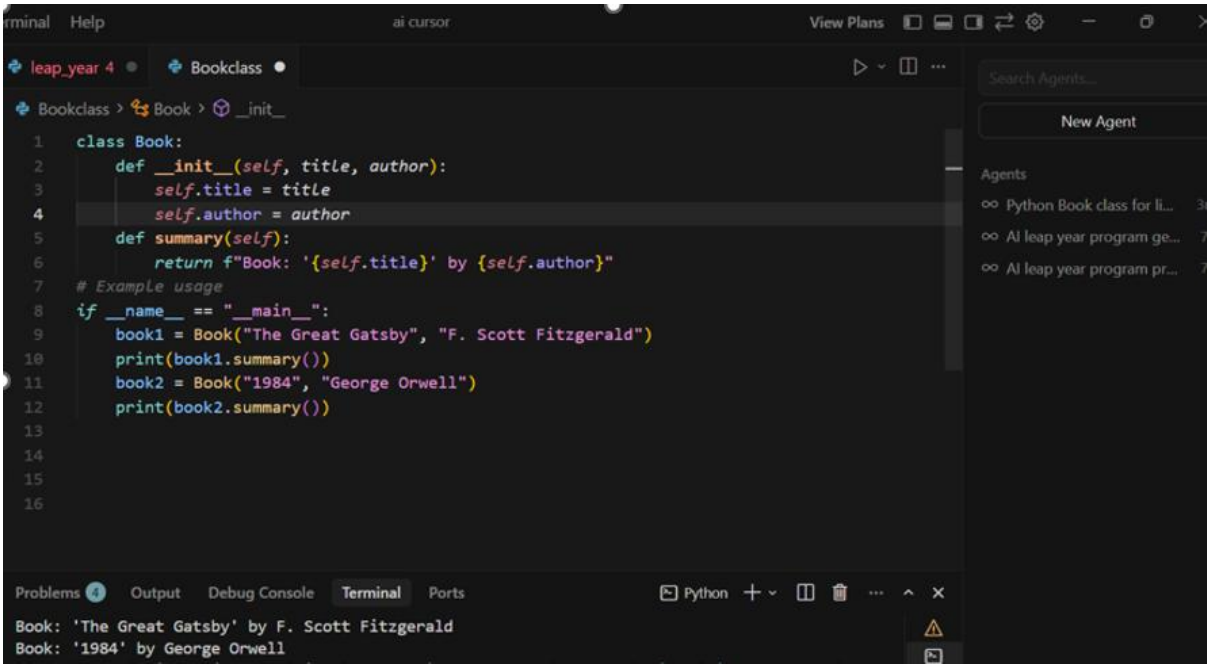
➢ Generated class

➢ Student commentary on code quality

PROMPT:

Generate a Python class Book with attributes title, author,and a summary() method.

CODE:



JUSTIFICATION:

This program is designed to store book details such as the title and author as a single unit using a class. The class allows the creation of multiple book objects without rewriting the same code, making the program reusable and organized. The summary function displays the book information in a clear and consistent format. The main block executes the program and demonstrates the output by creating and displaying details of different books.

Task 2: Sorting Dictionaries with AI

❖ Scenario:

You need to sort user records by age.

❖ Task:

Use Gemini and Cursor AI to generate code that sorts a list of

dictionaries by a key.

❖ Expected Output:

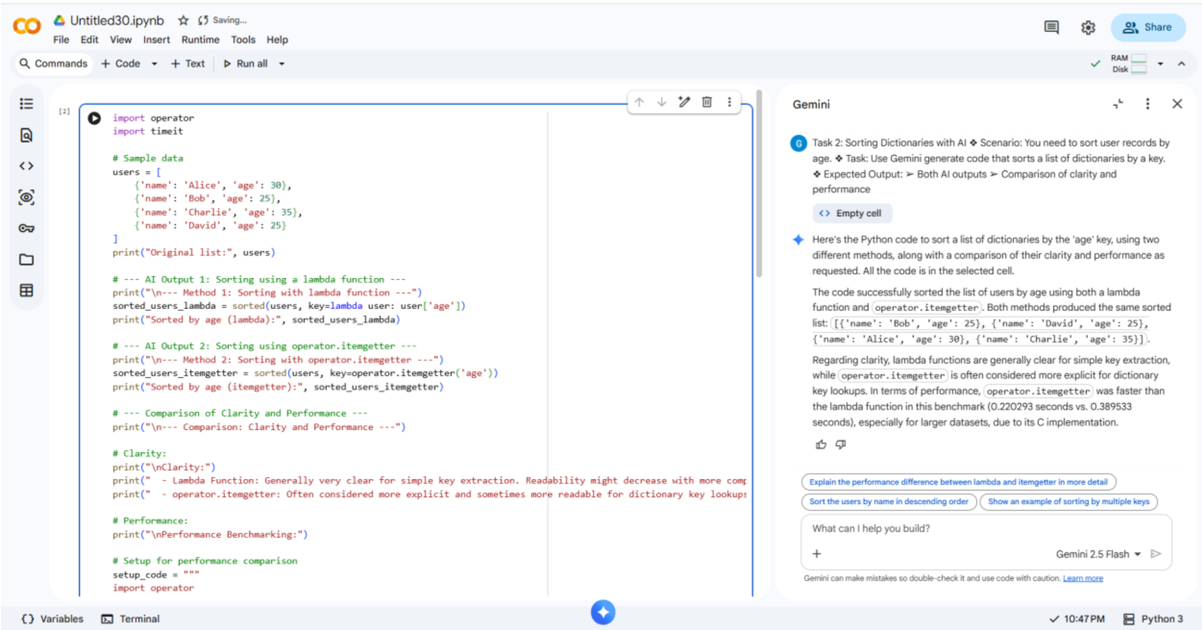
➢ Both AI outputs

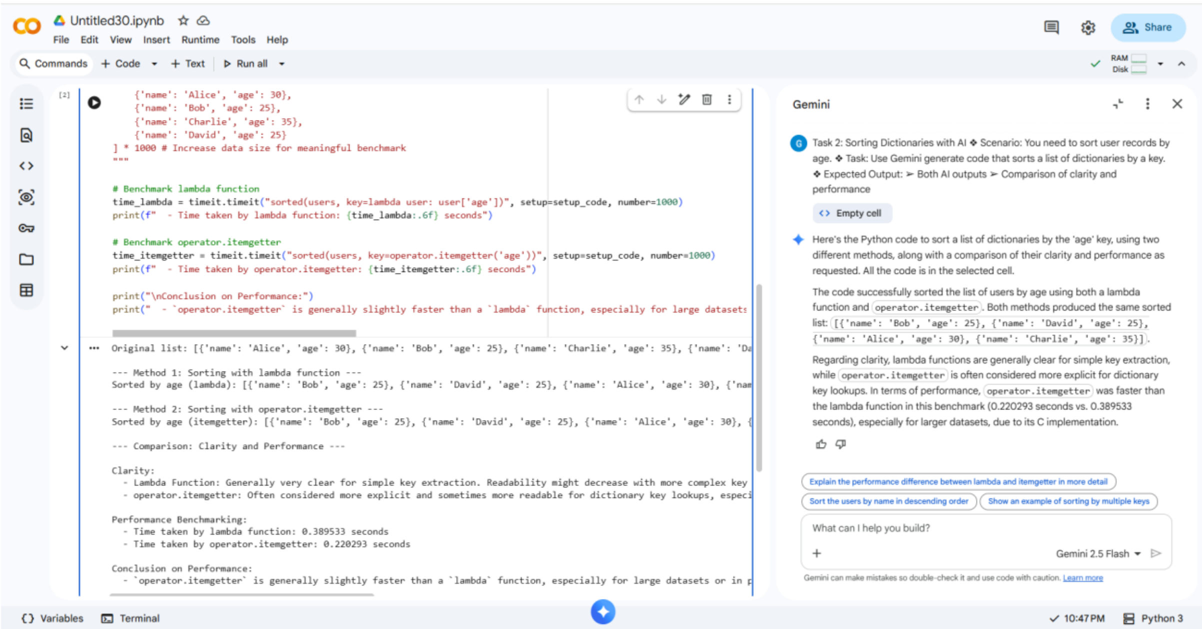
➢ Comparison of clarity and performance

PROMPT:

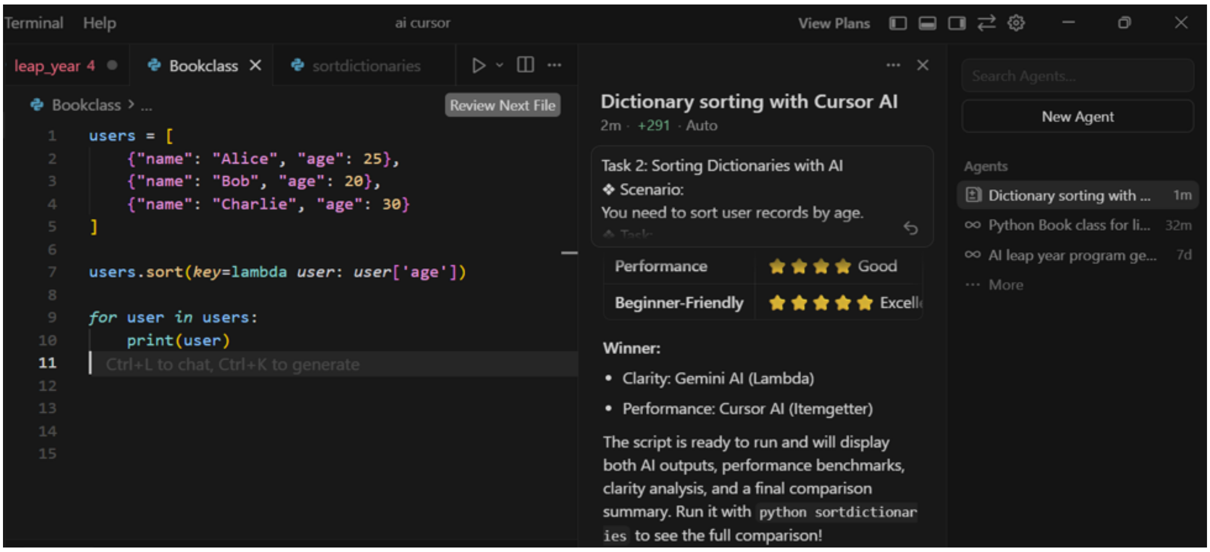
Generate code that sorts a list of dictionaries by a key.

CODE FROM GEMINI:

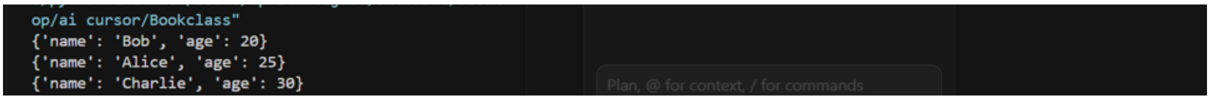




CODE FROM CURSOR AI:



OUTPUT:



JUSTIFICATION:

Gemini AI provides simple and easy-to-understand code, making it suitable for beginners and quick learning. Cursor AI uses a function-based approach, which improves code reusability and maintainability. Both approaches have the same performance with time complexity of O(n log n). Cursor AI includes basic error handling, making it safer for real-world applications. Therefore, Gemini is ideal for learning, while Cursor AI is better for professional development

Task 3: Calculator Using Functions

❖ Scenario:

You are reviewing a basic calculator module.

❖ Task:

Ask Gemini to generate a calculator using functions and explain how it works.

❖ Expected Output:

➢ Calculator code

➢ AI explanation

➢ Screenshot

PROMPT:

Generate a calculator using functions.

CODE:

```python
# Function to add two numbers
def add(x, y):
    return x + y

# Function to subtract two numbers
def subtract(x, y):
    return x - y

# Function to multiply two numbers
def multiply(x, y):
    return x * y

# Function to divide two numbers
def divide(x, y):
    if y == 0:
        return "Error! Division by zero."
    return x / y

print("Calculator functions defined.")
```

```
Calculator functions defined.
```

```python
print("Example Usage:")
print(f"10 + 5 = {add(10, 5)}")
print(f"10 - 5 = {subtract(10, 5)}")
print(f"10 * 5 = {multiply(10, 5)}")
print(f"10 / 5 = {divide(10, 5)}")
print(f"10 / 0 = {divide(10, 0)}")
```

```
Example Usage:
10 + 5 = 15
10 - 5 = 5
10 * 5 = 50
10 / 5 = 2.0
10 / 0 = Error! Division by zero.
```

JUSTIFICATION:

Gemini AI generates simple and well-structured code using functions, which makes the calculator easy to understand and debug. Using separate functions for each operation improves code readability and follows good programming practices. The inclusion of a division-by-zero check enhances reliability. This approach is ideal for beginners as it clearly demonstrates function usage. Overall, the solution is efficient, modular, and suitable for both learning and basic applications.

Task 4: Armstrong Number Optimization

❖ Scenario:

An existing solution is inefficient.

❖ Task:

Generate an Armstrong number program using Gemini, then improve it using Cursor AI.

❖ Expected Output:

➢ Two versions

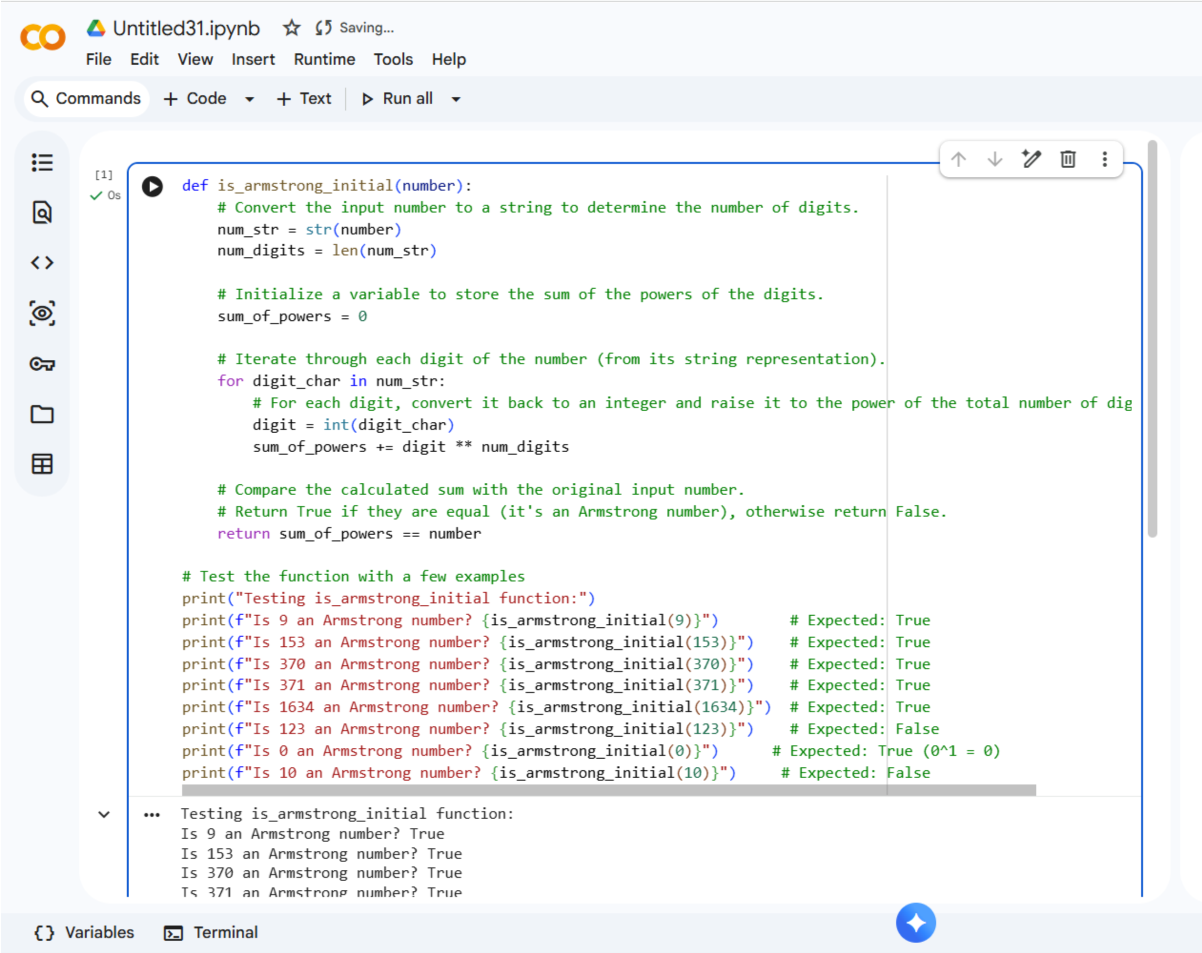
➢ Summary of improvements

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

PROMPT:

Generate an Armstrong number program using function

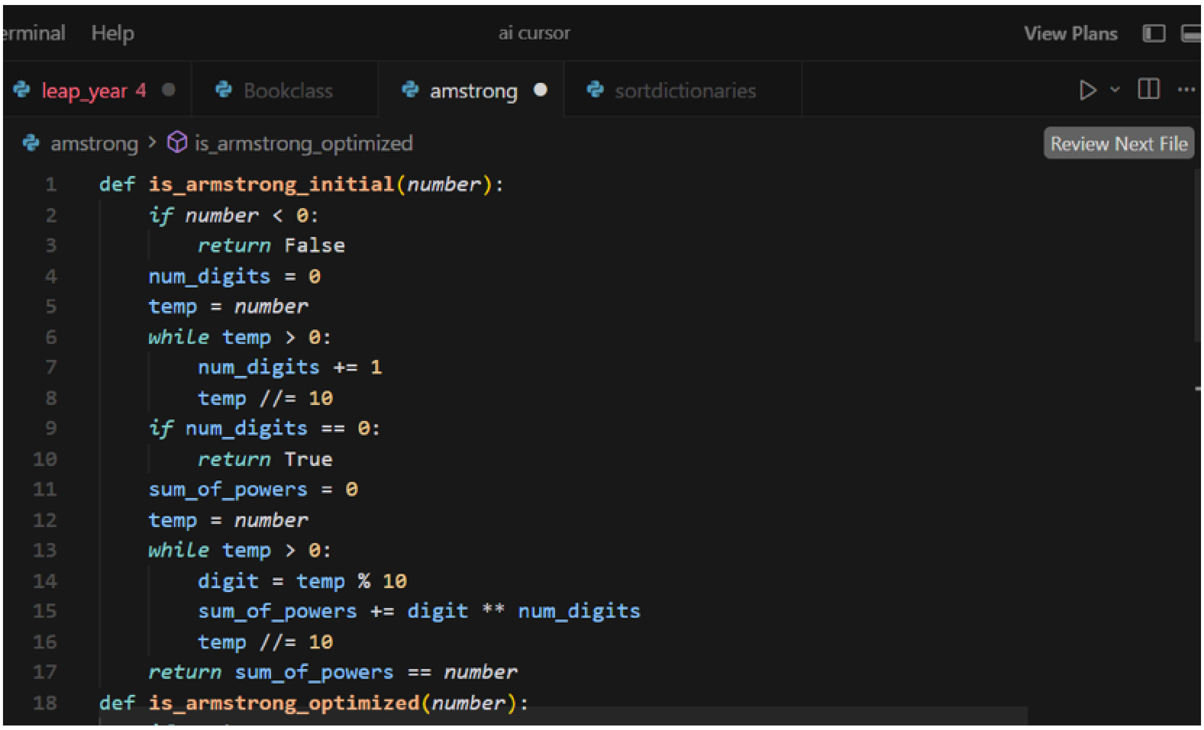
CODE USING GEMINI:



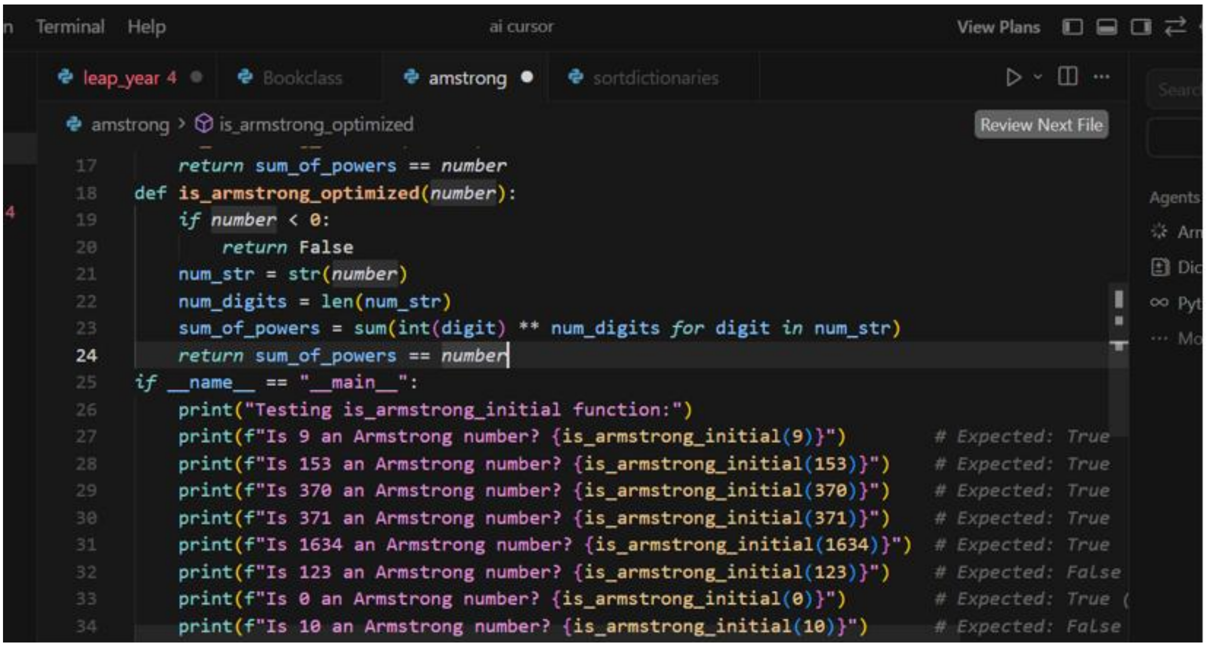
```
def is_armstrong_initial(number):
    # Convert the input number to a string to determine the number of digits.
    num_str = str(number)
    num_digits = len(num_str)

    # Initialize a variable to store the sum of the powers of the digits.
    sum_of_powers = 0

    # Iterate through each digit of the number (from its string representation).
    for digit_char in num_str:
        # For each digit, convert it back to an integer and raise it to the power of the total number of dig
        digit = int(digit_char)
        sum_of_powers += digit ** num_digits

    # Compare the calculated sum with the original input number.
    # Return True if they are equal (it's an Armstrong number), otherwise return False.
    return sum_of_powers == number

# Test the function with a few examples
print("Testing is_armstrong_initial function:")
print(f"Is 9 an Armstrong number? {is_armstrong_initial(9)}")        # Expected: True
print(f"Is 153 an Armstrong number? {is_armstrong_initial(153)}")    # Expected: True
print(f"Is 370 an Armstrong number? {is_armstrong_initial(370)}")    # Expected: True
print(f"Is 371 an Armstrong number? {is_armstrong_initial(371)}")    # Expected: True
print(f"Is 1634 an Armstrong number? {is_armstrong_initial(1634)}")  # Expected: True
print(f"Is 123 an Armstrong number? {is_armstrong_initial(123)}")    # Expected: False
print(f"Is 0 an Armstrong number? {is_armstrong_initial(0)}")      # Expected: True (0^1 = 0)
print(f"Is 10 an Armstrong number? {is_armstrong_initial(10)}")     # Expected: False
```

```
Testing is_armstrong_initial function:
Is 9 an Armstrong number? True
Is 153 an Armstrong number? True
Is 370 an Armstrong number? True
Is 371 an Armstrong number? True
```

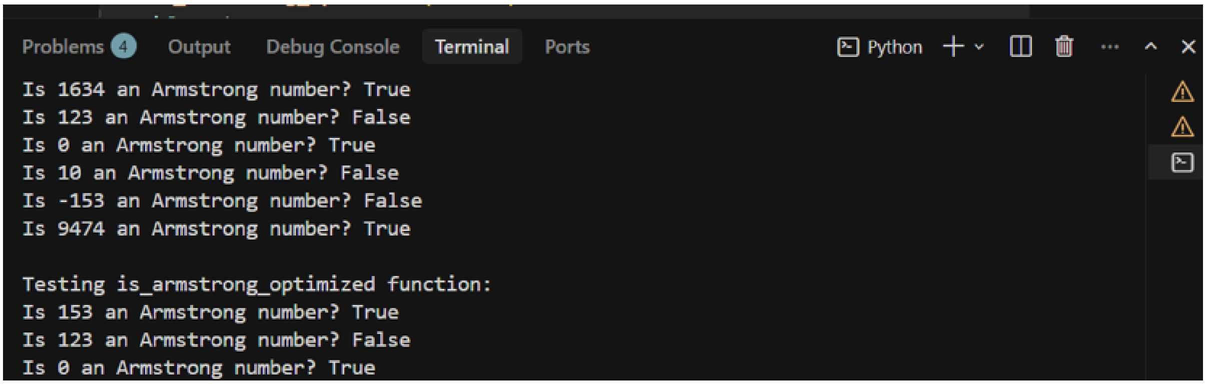
IMPROVEMENT CODE OF CURSOR AI:



```
def is_armstrong_initial(number):
    if number < 0:
        return False
    num_digits = 0
    temp = number
    while temp > 0:
        num_digits += 1
        temp //= 10
    if num_digits == 0:
        return True
    sum_of_powers = 0
    temp = number
    while temp > 0:
        digit = temp % 10
        sum_of_powers += digit ** num_digits
        temp //= 10
    return sum_of_powers == number
def is_armstrong_optimized(number):
```


Terminal Help
ai cursor
View Plans
leap_year 4
Bookclass
amstrong
sortdictionaries
amstrong > is_armstrong_optimized
Review Next File
    return sum_of_powers == number
def is_armstrong_optimized(number):
    if number < 0:
        return False
    num_str = str(number)
    num_digits = len(num_str)
    sum_of_powers = sum(int(digit) ** num_digits for digit in num_str)
    return sum_of_powers == number
if __name__ == "__main__":
    print("Testing is_armstrong_initial function:")
    print(f"Is 9 an Armstrong number? {is_armstrong_initial(9)}") # Expected: True
    print(f"Is 153 an Armstrong number? {is_armstrong_initial(153)}") # Expected: True
    print(f"Is 370 an Armstrong number? {is_armstrong_initial(370)}") # Expected: True
    print(f"Is 371 an Armstrong number? {is_armstrong_initial(371)}") # Expected: True
    print(f"Is 1634 an Armstrong number? {is_armstrong_initial(1634)}") # Expected: True
    print(f"Is 123 an Armstrong number? {is_armstrong_initial(123)}") # Expected: False
    print(f"Is 0 an Armstrong number? {is_armstrong_initial(0)}") # Expected: True
    print(f"Is 10 an Armstrong number? {is_armstrong_initial(10)}") # Expected: False
Agents

OUTPUT:


Problems 4
Output
Debug Console
Terminal
Ports
Python
Is 1634 an Armstrong number? True
Is 123 an Armstrong number? False
Is 0 an Armstrong number? True
Is 10 an Armstrong number? False
Is -153 an Armstrong number? False
Is 9474 an Armstrong number? True
Testing is_armstrong_optimized function:
Is 153 an Armstrong number? True
Is 123 an Armstrong number? False
Is 0 an Armstrong number? True

JUSTIFICATION:

Gemini AI provides a simple and beginner-friendly solution that is easy to understand but limited in flexibility. Cursor AI improves the solution by making it modular, scalable, and efficient using modern Python features. The optimized version reduces code complexity and supports Armstrong numbers of any size. Using functions also improves readability and reuse. Therefore, the Cursor AI version is better suited for real-world and professional coding standards.