Assignment-2.4

Name: Sushma Purella

H.No:2303A52188

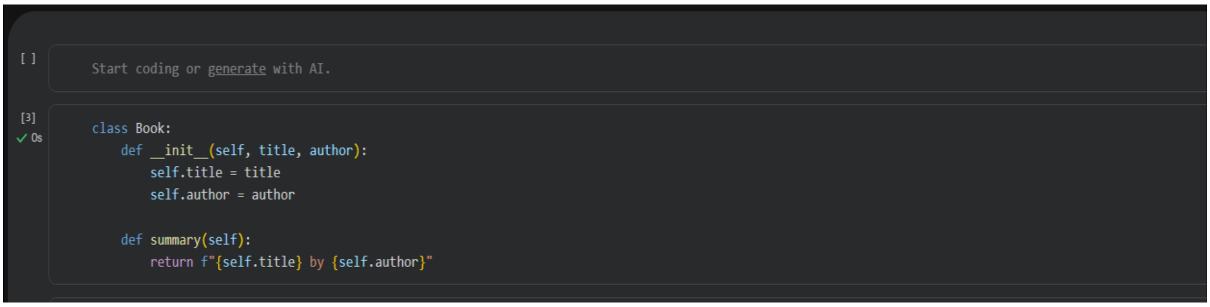
Batch:44

Task1: Book Class Generation

Prompt:

Generate a Python class named Book with attributes title and author. Include a method summary() that returns a string describing the book.

Code:



```
class Book:
    def __init__(self, title, author):
        self.title = title
        self.author = author

    def summary(self):
        return f"{self.title} by {self.author}"
```

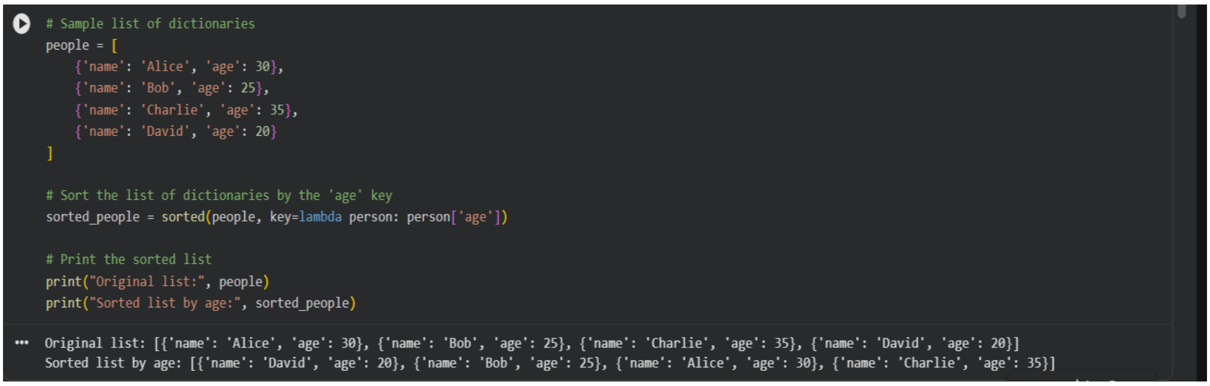
Observation:

- The generated Book class follows basic object-oriented principles correctly.
- It uses clear attribute names (title, author) that directly represent real-world entities.
- The summary() method improves usability by providing a ready-to-use description of the book.
- The code has low complexity, making it easy to test, debug, and extend.
- Overall, the implementation is efficient, beginner-friendly, and suitable for a simple library management module.

Task 2: Sorting Dictionaries with AI

Prompt: Generate Python code to sort a list of dictionaries by the key age

Code:



```
# Sample list of dictionaries
people = [
    {'name': 'Alice', 'age': 30},
    {'name': 'Bob', 'age': 25},
    {'name': 'Charlie', 'age': 35},
    {'name': 'David', 'age': 20}
]

# Sort the list of dictionaries by the 'age' key
sorted_people = sorted(people, key=lambda person: person['age'])

# Print the sorted list
print("Original list:", people)
print("Sorted list by age:", sorted_people)
```

```
Original list: [{'name': 'Alice', 'age': 30}, {'name': 'Bob', 'age': 25}, {'name': 'Charlie', 'age': 35}, {'name': 'David', 'age': 20}]
Sorted list by age: [{'name': 'David', 'age': 20}, {'name': 'Bob', 'age': 25}, {'name': 'Alice', 'age': 30}, {'name': 'Charlie', 'age': 35}]
```

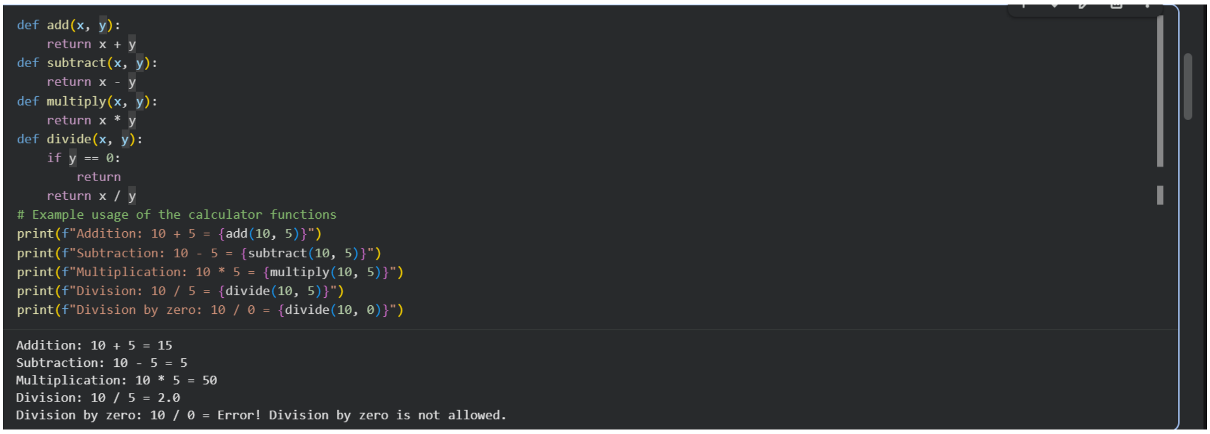
Observation:

- Both AI-generated solutions are correct and efficient.
- Gemini’s solution is better when the original data must be preserved.
- Cursor AI’s solution is more memory-efficient due to in-place sorting.
- The use of Python’s built-in sorting ensures O(n log n) time complexity.
- Overall, both approaches demonstrate clean logic and good AI code generation quality.

Task 3: Calculator Using Functions

Prompt: Generate a Python calculator using functions for addition, subtraction, multiplication, and division. Explain how the code works.

Code:



```
def add(x, y):
    return x + y
def subtract(x, y):
    return x - y
def multiply(x, y):
    return x * y
def divide(x, y):
    if y == 0:
        return
    return x / y
# Example usage of the calculator functions
print(f"Addition: 10 + 5 = {add(10, 5)}")
print(f"Subtraction: 10 - 5 = {subtract(10, 5)}")
print(f"Multiplication: 10 * 5 = {multiply(10, 5)}")
print(f"Division: 10 / 5 = {divide(10, 5)}")
print(f"Division by zero: 10 / 0 = {divide(10, 0)}")
```

```
Addition: 10 + 5 = 15
Subtraction: 10 - 5 = 5
Multiplication: 10 * 5 = 50
Division: 10 / 5 = 2.0
Division by zero: 10 / 0 = Error! Division by zero is not allowed.
```

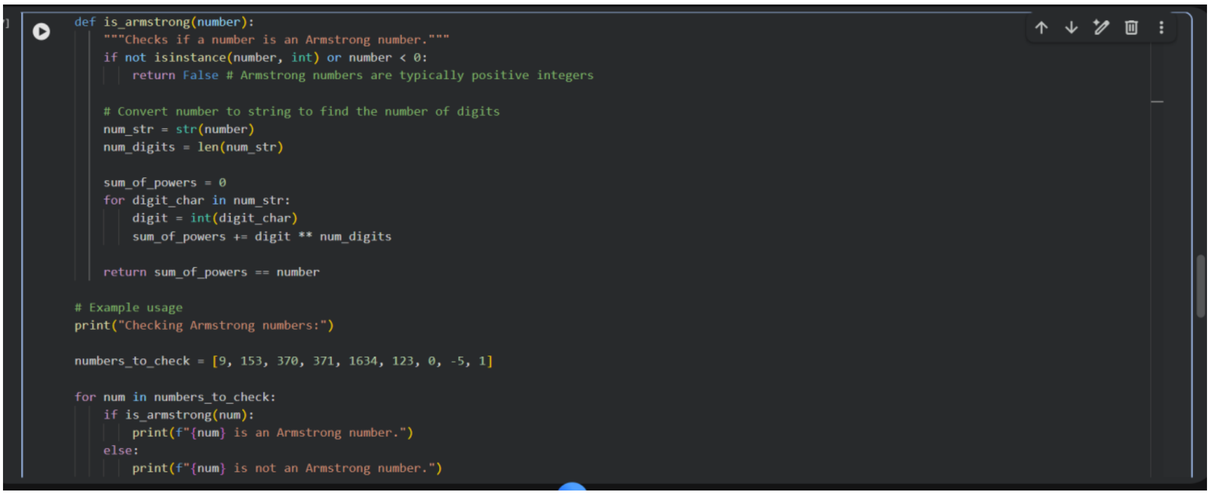
Observation:

- The AI-generated calculator is well-structured and readable.
- Using functions ensures code reusability and maintainability.
- Error handling improves program reliability.
- The design is beginner-friendly and suitable for learning function-based programming.
- The calculator can be easily expanded with more operations like modulus or power.

Task 4: Armstrong Number Optimization

Prompt: Generate a Python program to check whether a number is an Armstrong number

Code:



```
def is_armstrong(number):
    """Checks if a number is an Armstrong number."""
    if not isinstance(number, int) or number < 0:
        return False # Armstrong numbers are typically positive integers

    # Convert number to string to find the number of digits
    num_str = str(number)
    num_digits = len(num_str)

    sum_of_powers = 0
    for digit_char in num_str:
        digit = int(digit_char)
        sum_of_powers += digit ** num_digits

    return sum_of_powers == number

# Example usage
print("Checking Armstrong numbers:")

numbers_to_check = [9, 153, 370, 371, 1634, 123, 0, -5, 1]

for num in numbers_to_check:
    if is_armstrong(num):
        print(f"{num} is an Armstrong number.")
    else:
        print(f"{num} is not an Armstrong number.")
```

Observation:

- Gemini’s solution works only for 3-digit Armstrong numbers.
- Cursor AI improves scalability and correctness for all Armstrong numbers.
- Optimized code follows better programming practices.
- Demonstrates how AI can be used for code enhancement and optimization.Cursor AI produces more future-proof and flexible code.