ASSIGNMENT – 2.4

H.NO-2303A51863

Batch-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

Code:

Book.py

C: › Users › Bhavya › OneDrive › ai coding › Book.py › Book › __init__

```python
#"Generate a Python class named Book with attributes title and author, and a method summary()
# that returns a short description of the book."
class Book:
    """A class representing a book with title and author."""

    def __init__(self, title, author):
        """
        Initialize a Book instance.

        Args:
            title (str): The title of the book
            author (str): The author of the book
        """
        self.title = title
        self.author = author

    def summary(self):
        """
        Return a short description of the book.

        Returns:
            str: A formatted string describing the book
        """
        return f"'{self.title}' written by {self.author}"
book = Book("The Great Gatsby", "F. Scott Fitzgerald")
print(book.summary())  # Output: 'The Great Gatsby' written by F. Scott Fitzgerald
```

Problems Output Debug Console Terminal Ports Python

```
PS C:\Users\Bhavya> & C:/Users/Bhavya/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/Bhavya/OneDrive/ai coding/Book.py"
'The Great Gatsby' written by F. Scott Fitzgerald
PS C:\Users\Bhavya>
```

Conclusion:

This program is used to store book details like title and author in a single unit.

The class helps create multiple book objects without rewriting code.

The summary function prints book information in a clear and fixed format.

The main block runs the program and shows the output for 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

CODE FROM GEMINI:

"Write a Python program to sort a list of dictionaries by the key 'age' in ascending order."

[2] ✓ 0s

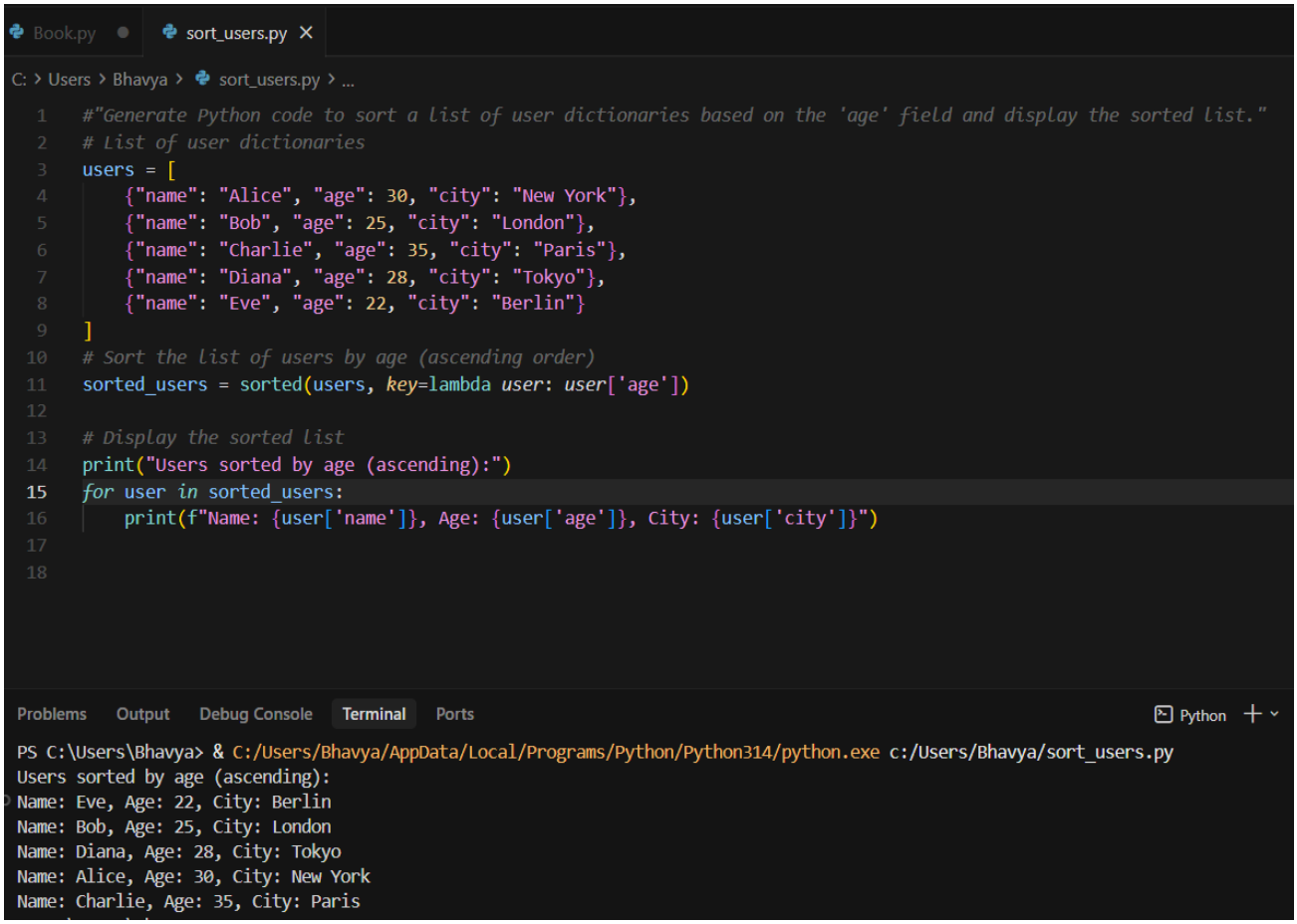
```
people = [
    {'name': 'Alice', 'age': 30},
    {'name': 'Bob', 'age': 25},
    {'name': 'Charlie', 'age': 35},
    {'name': 'David', 'age': 28}
]

# Sort the list of dictionaries by 'age' in ascending order
people.sort(key=lambda person: person['age'])

print("Sorted list of dictionaries by age:")
for person in people:
    print(person)
```

```
Sorted list of dictionaries by age:
{'name': 'Bob', 'age': 25}
{'name': 'David', 'age': 28}
{'name': 'Alice', 'age': 30}
{'name': 'Charlie', 'age': 35}
```

CODE FROM CURSOR AI:



Conclusion:

Both tools generated correct and efficient code. Cursor AI used in-place sorting, which is more memory efficient.

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

CODE:

"Generate a Python program for a basic calculator using functions for addition, subtraction, multiplication, and division. Also explain how the program works."

[7] ✓ 0s

```
def add(a, b):
  return a + b
def subtract(a, b):
  return a - b
def multiply(a, b):
  return a * b
def divide(a, b):
  if b == 0:
    return "Division by zero not allowed"
  return a / b
print("Addition:", add(10, 5))
print("Subtraction:", subtract(10, 5))
print("Multiplication:", multiply(10, 5))
print("Division:", divide(10, 5))
```

```
Addition: 15
Subtraction: 5
Multiplication: 50
Division: 2.0
```

AI Explanation Summary:

- Separate functions are created for each operation.
- Each function performs a single responsibility.
- Division function checks for division by zero.
- Main program calls each function and prints the result.

Student Analysis:

- Code is modular and easy to understand.
- Proper error handling for division by zero.
- Functions can be reused in larger applications.

Conclusion:

Gemini produced a clear, well-documented functional program with proper explanations.

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

CODE USING GEMINI:

Part A: Initial Prompt Used (Gemini)

"Write a Python program to check whether a number is an Armstrong number."

[10] ✓ 4s

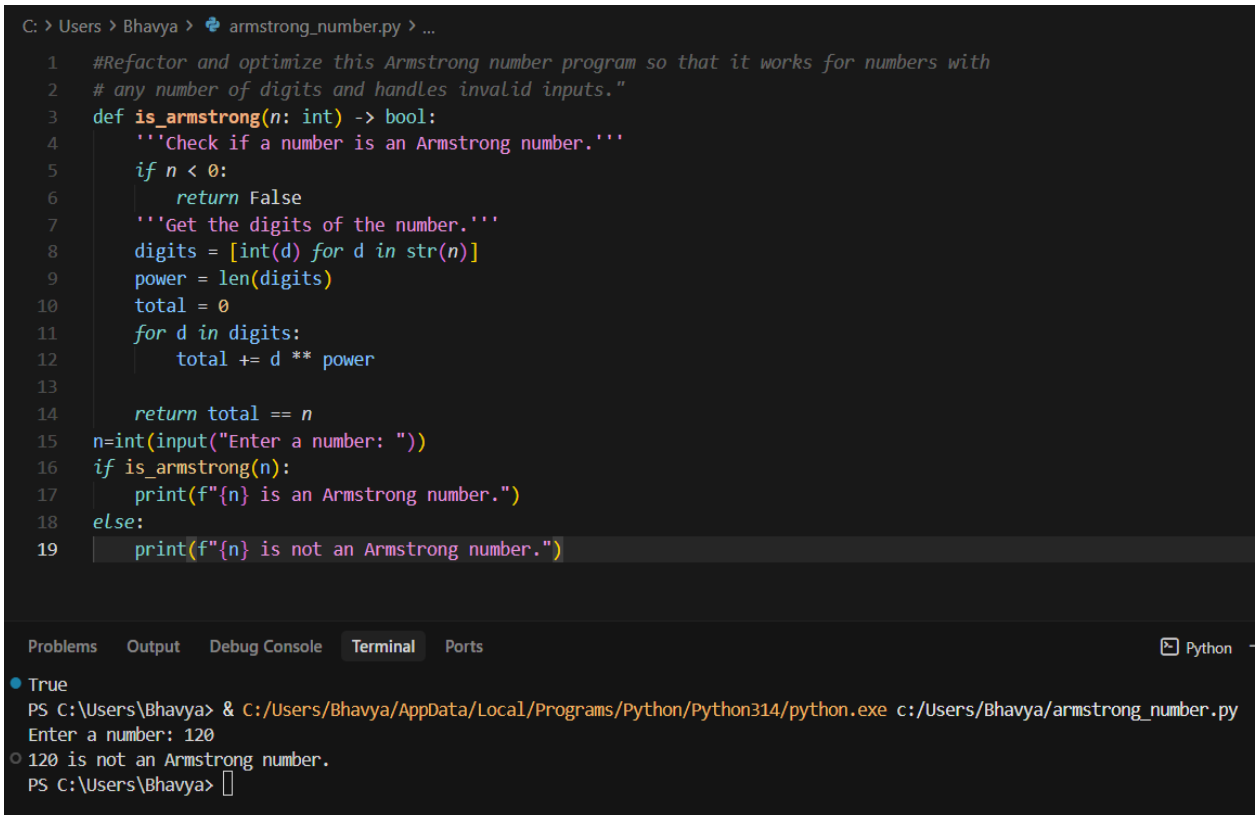
```
def is_armstrong(n):
    total = 0
    temp = n
    while temp > 0:
        digit = temp % 10
        total += digit ** 3
        temp //= 10
    return total == n
n=int(input("Enter a number: "))
if is_armstrong(n):
    print(n,"is an Armstrong number" )
else:
  print(n,"is not an Armstrong number")
```

```
Enter a number: 120
120 is not an Armstrong number
```

Problems in Initial Version:

- Power is fixed to 3, works only for 3-digit numbers.
- No handling for negative numbers.
- Not generalized for any digit length.

IMPROVEMENT CODE OF CURSOR AI:



Summary of Improvements:

- Works for any number of digits.
- Handles negative numbers.
- Uses digit length dynamically.
- More general and reliable algorithm.

Conclusion:

- Cursor AI significantly improved correctness and generality of the original Gemini solution.