

# 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:

```
C: > Users > Bhavya > OneDrive > ai coding > Book.py > Book > __init__.py

1  # Generate a Python class named Book with attributes title and author, and a method summary()
2  # that returns a short description of the book.
3  class Book:
4      """A class representing a book with title and author."""
5
6  def __init__(self, title, author):
7      """
8          Initialize a Book instance.
9
10     Args:
11         title (str): The title of the book
12         author (str): The author of the book
13
14     """
15     self.title = title
16     self.author = author
17
18 def summary(self):
19     """
20         Return a short description of the book.
21
22     Returns:
23         str: A formatted string describing the book
24
25     return f'{self.title} written by {self.author}'
26 book = Book("The Great Gatsby", "F. Scott Fitzgerald")
27 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:

The screenshot shows a code editor with two tabs: 'Book.py' and 'sort\_users.py'. The 'sort\_users.py' tab is active, displaying the following Python code:

```
1  # Generate Python code to sort a list of user dictionaries based on the 'age' field and display the sorted list.
2  # List of user dictionaries
3  users = [
4      {"name": "Alice", "age": 30, "city": "New York"},
5      {"name": "Bob", "age": 25, "city": "London"},
6      {"name": "Charlie", "age": 35, "city": "Paris"},
7      {"name": "Diana", "age": 28, "city": "Tokyo"},
8      {"name": "Eve", "age": 22, "city": "Berlin"}
9  ]
10 # Sort the list of users by age (ascending order)
11 sorted_users = sorted(users, key=lambda user: user['age'])
12
13 # Display the sorted list
14 print("Users sorted by age (ascending):")
15 for user in sorted_users:
16     print(f"Name: {user['name']}, Age: {user['age']}, City: {user['city']}")
```

Below the code editor is a terminal window showing the execution of the script and its output:

```
PS C:\Users\Bhavya> & C:/Users/Bhavya/AppData/Local/Programs/Python/Python314/python.exe c:/Users/Bhavya/sort_users.py
Users sorted by age (ascending):
Name: Eve, Age: 22, City: Berlin
Name: Bob, Age: 25, City: London
Name: Diana, Age: 28, City: Tokyo
Name: Alice, Age: 30, City: New York
Name: Charlie, Age: 35, City: Paris
```

## 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:

```
C: > Users > Bhavya > armstrong_number.py > ...
1  #Refactor and optimize this Armstrong number program so that it works for numbers with
2  # any number of digits and handles invalid inputs.
3  def is_armstrong(n: int) -> bool:
4      '''Check if a number is an Armstrong number.'''
5      if n < 0:
6          return False
7      '''Get the digits of the number.'''
8      digits = [int(d) for d in str(n)]
9      power = len(digits)
10     total = 0
11     for d in digits:
12         total += d ** power
13
14     return total == n
15 n=int(input("Enter a number: "))
16 if is_armstrong(n):
17     print(f"{n} is an Armstrong number.")
18 else:
19     print(f"{n} is not an Armstrong number.)
```

Problems   Output   Debug Console   Terminal   Ports   Python

- True
- PS C:\Users\Bhavya> & C:/Users/Bhavya/AppData/Local/Programs/Python/Python314/python.exe c:/Users/Bhavya/armstrong\_number.py
- Enter a number: 120
- 120 is not an Armstrong number.
- PS C:\Users\Bhavya>

## 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.