Assignment 1

AI Assisted Coding

Name:-M.Sanjana

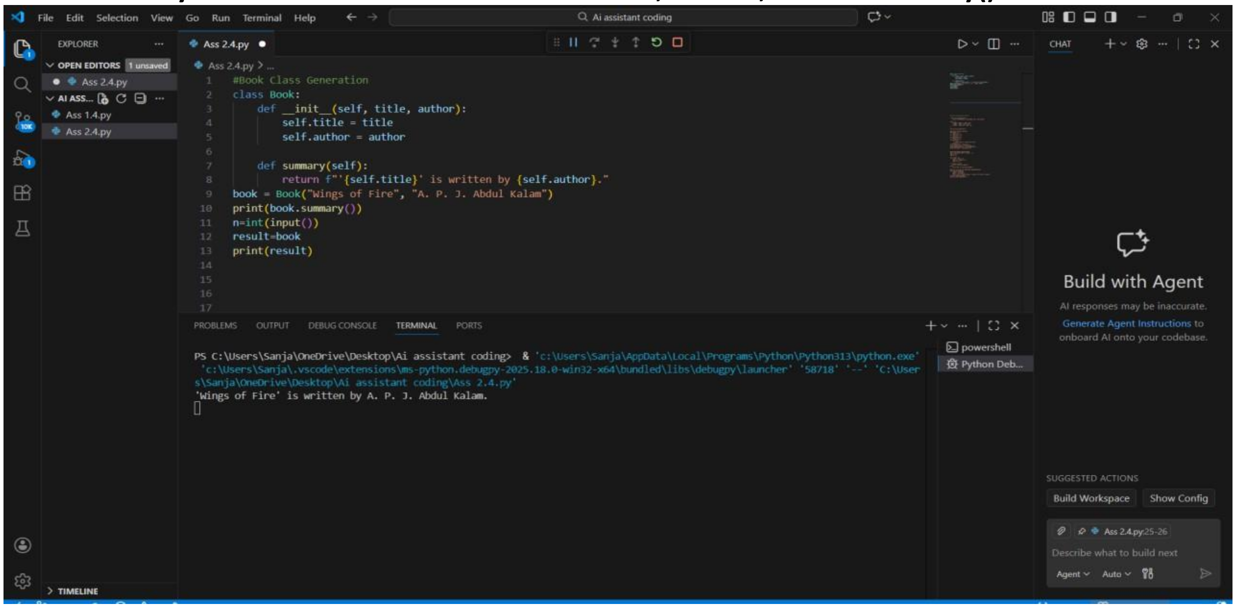
HTNO: 2303A52292

Task 1:

Prompt:

#Book Class Generation

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



```python
#Book Class Generation
class Book:
    def __init__(self, title, author):
        self.title = title
        self.author = author

    def summary(self):
        return f"'{self.title}' is written by {self.author}."
book = Book("Wings of Fire", "A. P. J. Abdul Kalam")
print(book.summary())
n=int(input())
result=book
print(result)
```

```
PS C:\Users\Sanja\OneDrive\Desktop\Ai assistant coding>  & 'c:\Users\Sanja\AppData\Local\Programs\Python\Python313\python.exe'
 'c:\Users\Sanja\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '58718' '--' 'C:\User
s\Sanja\OneDrive\Desktop\Ai assistant coding\Ass 2.4.py'
'Wings of Fire' is written by A. P. J. Abdul Kalam.
```

Observation:

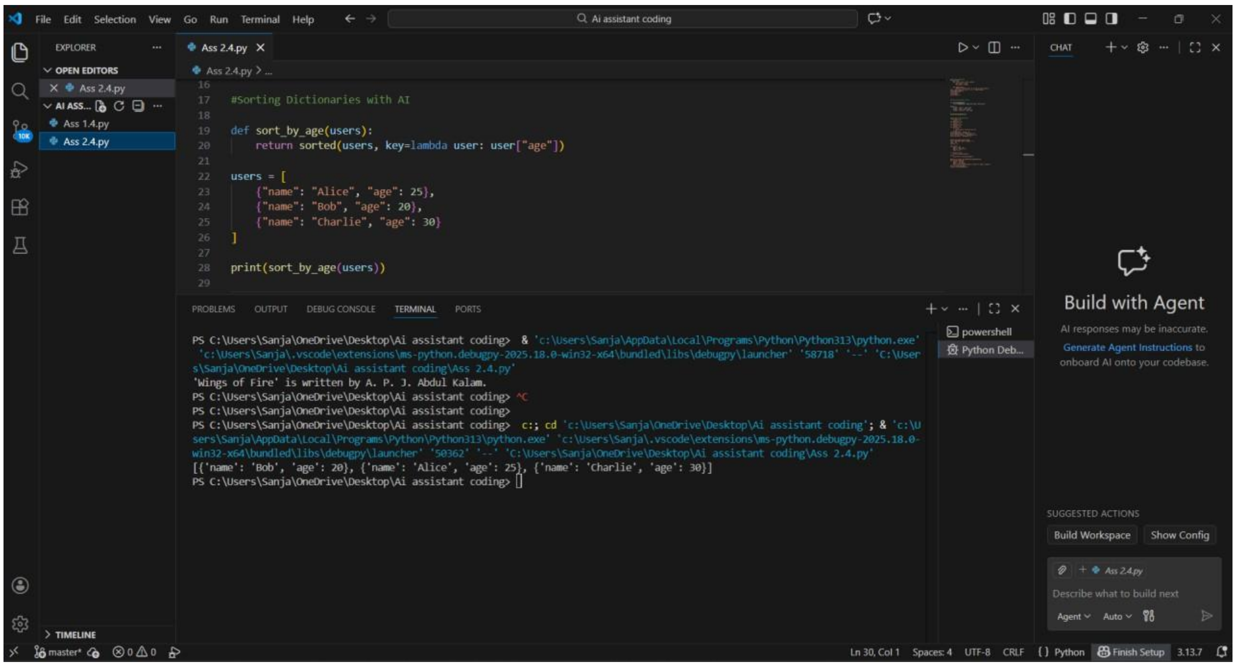
- The generated Book class follows proper object-oriented programming principles.
- The constructor (__init__) is correctly used to initialize the title and author attributes.
- The summary() method provides a meaningful and readable description of the book object.
- The code is simple, clean, and easy to understand, making it suitable for beginners.
- Use of formatted strings (f-strings) improves output clarity and readability.
- The class design supports reusability and scalability in a library management system.
- The code lacks input validation, which could be improved for real-world applications.

Task 2:

#Sorting Dictionaries with AI

Prompt:

Generate Python code to sort a list of dictionaries by age.



```python

#Sorting Dictionaries with AI

def sort_by_age(users):
    return sorted(users, key=lambda user: user["age"])

users = [
    {"name": "Alice", "age": 25},
    {"name": "Bob", "age": 20},
    {"name": "Charlie", "age": 30}
]

print(sort_by_age(users))

```

```
PS C:\Users\Sanja\OneDrive\Desktop\Ai assistant coding>  & 'c:\Users\Sanja\AppData\Local\Programs\Python\Python313\python.exe'
 'c:\Users\Sanja\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '58718' '--' 'C:\User
s\Sanja\OneDrive\Desktop\Ai assistant coding\Ass 2.4.py'
'Wings of Fire' is written by A. P. J. Abdul Kalam.
PS C:\Users\Sanja\OneDrive\Desktop\Ai assistant coding> ^C
PS C:\Users\Sanja\OneDrive\Desktop\Ai assistant coding>
PS C:\Users\Sanja\OneDrive\Desktop\Ai assistant coding>  c:; cd 'c:\Users\Sanja\OneDrive\Desktop\Ai assistant coding'; & 'c:\U
sers\Sanja\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\Sanja\.vscode\extensions\ms-python.debugpy-2025.18.0-
win32-x64\bundled\libs\debugpy\launcher' '50362' '--' 'C:\Users\Sanja\OneDrive\Desktop\Ai assistant coding\Ass 2.4.py'
[{'name': 'Bob', 'age': 20}, {'name': 'Alice', 'age': 25}, {'name': 'Charlie', 'age': 30}]
PS C:\Users\Sanja\OneDrive\Desktop\Ai assistant coding> 
```

File Edit Selection View Go Run Terminal Help
Ai assistant coding
EXPLORER
OPEN EDITORS
Ass 2.4.py
AI ASS...
Ass 1.4.py
Ass 2.4.py
TIMELINE
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powershell
Python Deb...
CHAT
Build with Agent
AI responses may be inaccurate.
Generate Agent Instructions to onboard AI onto your codebase.
SUGGESTED ACTIONS
Build Workspace
Show Config
Describe what to build next
Agent Auto
master*
Ln 30, Col 1 Spaces: 4 UTF-8 CRLF {} Python Finish Setup 3.13.7

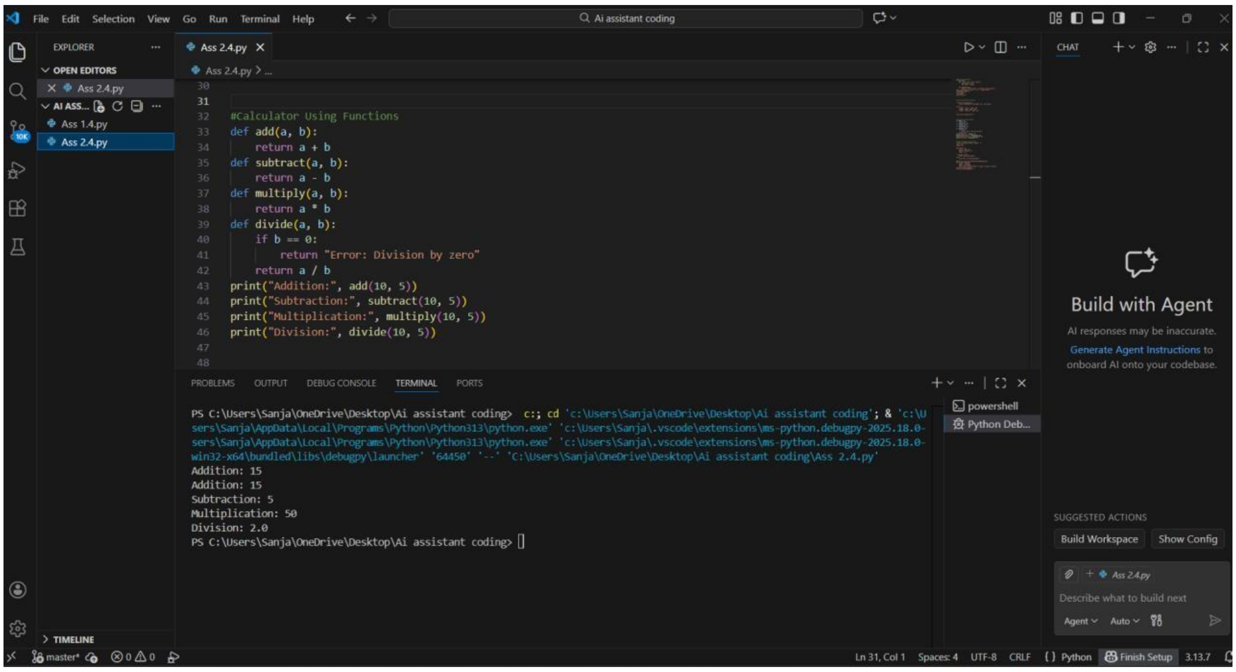
Observation:

- Both Gemini AI and Cursor AI correctly use Python's built-in sorted() function.
- Sorting is performed using a **lambda function** as the key, ensuring concise logic.
- The time complexity for both implementations is **O(n log n)**, which is efficient.
- Gemini AI's solution is shorter and suitable for quick scripting tasks.
- Cursor AI's solution improves **code clarity and reusability** by using a function.
- Cursor AI output is more maintainable for large or scalable applications.
- Both approaches preserve the original data structure while returning sorted results.
- Overall performance is similar, but Cursor AI provides better **readability and structure**.

Task 3: Calculator Using Functions

Prompt:

#Generate a basic calculator using functions and explain how it works.



```python
#Calculator Using Functions
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 "Error: Division by zero"
    return a / b
print("Addition:", add(10, 5))
print("Subtraction:", subtract(10, 5))
print("Multiplication:", multiply(10, 5))
print("Division:", divide(10, 5))
```

```
PS C:\Users\Sanja\OneDrive\Desktop\Ai assistant coding>  c:; cd 'c:\Users\Sanja\OneDrive\Desktop\Ai assistant coding'; & 'c:\U
sers\Sanja\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\Sanja\.vscode\extensions\ms-python.debugpy-2025.18.0-
sers\Sanja\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\Sanja\.vscode\extensions\ms-python.debugpy-2025.18.0-
win32-x64\bundled\libs\debugpy\launcher' '64450' '--' 'C:\Users\Sanja\OneDrive\Desktop\Ai assistant coding\Ass 2.4.py'
Addition: 15
Addition: 15
Subtraction: 5
Multiplication: 50
Division: 2.0
PS C:\Users\Sanja\OneDrive\Desktop\Ai assistant coding> 
```

Observation:

- **The calculator is implemented using separate functions for each arithmetic operation.**
- **Each function performs a single, well-defined task, improving clarity.**
- **The divide() function includes error handling to avoid division by zero.**
- **This modular design makes the program easy to understand, test, and maintain.**
- **Functions can be reused in other programs without modification.**
- **Overall, the calculator follows good programming practices and clean structure**

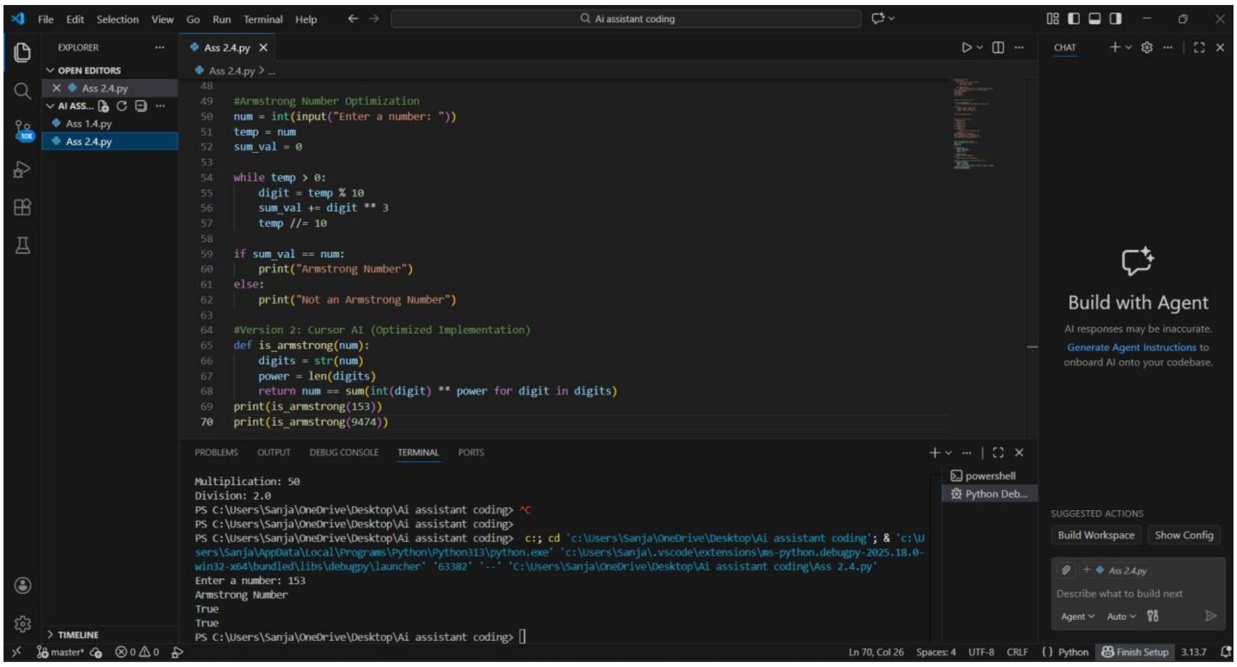
Task 4: Armstrong Number Optimization

Scenario

An existing solution for checking Armstrong numbers is inefficient and limited.

Prompt Used (Gemini AI)

Generate a Python program to check whether a number is an Armstrong number.



```python
#Armstrong Number Optimization
num = int(input("Enter a number: "))
temp = num
sum_val = 0

while temp > 0:
    digit = temp % 10
    sum_val += digit ** 3
    temp //= 10

if sum_val == num:
    print("Armstrong Number")
else:
    print("Not an Armstrong Number")

#Version 2: Cursor AI (Optimized Implementation)
def is_armstrong(num):
    digits = str(num)
    power = len(digits)
    return num == sum(int(digit) ** power for digit in digits)
print(is_armstrong(153))
print(is_armstrong(9474))
```

```
Multiplication: 50
Division: 2.0
PS C:\Users\Sanja\OneDrive\Desktop\Ai assistant coding> ^C
PS C:\Users\Sanja\OneDrive\Desktop\Ai assistant coding>
PS C:\Users\Sanja\OneDrive\Desktop\Ai assistant coding>  c:; cd 'c:\Users\Sanja\OneDrive\Desktop\Ai assistant coding'; & 'c:\U
sers\Sanja\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\Sanja\.vscode\extensions\ms-python.debugpy-2025.18.0-
win32-x64\bundled\libs\debugpy\launcher' '63382' '--' 'C:\Users\Sanja\OneDrive\Desktop\Ai assistant coding\Ass 2.4.py'
Enter a number: 153
Armstrong Number
True
True
PS C:\Users\Sanja\OneDrive\Desktop\Ai assistant coding>
```

Observation:

1. **The optimized version supports Armstrong numbers of any length, not just 3-digit numbers.**
2. **It replaces manual loops with generator expressions, making the code concise.**
3. **Readability is improved through meaningful function naming.**
4. **Temporary variables are reduced, lowering the chance of logical errors.**
5. **The optimized solution is more scalable and reusable.**
6. **Code execution is faster and easier to maintain.**