

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.

The screenshot shows the VS Code interface with the following details:

- EXPLORER:** Shows files Ass 2.4.py, Ass 2.4.py (unresolved), Ass 1.4.py, and AI ASS... (with a refresh icon).
- CODE EDITOR:** Displays the following Python code:

```
1 #Book Class Generation
2 class book:
3     def __init__(self, title, author):
4         self.title = title
5         self.author = author
6
7     def summary(self):
8         return f'{self.title} is written by {self.author}.'
9 book = book("Wings of Fire", "A. P. J. Abdul Kalam")
10 print(book.summary())
11 n=int(input())
12 result=book
13 print(result)
14
15
16
17
```
- TERMINAL:** Shows the command run in PowerShell: PS C:\Users\Sanja\OneDrive\Desktop\AI assistant coding & 'c:\Users\Sanja\AppData\Local\Programs\Python\Python311\python.exe' 'c:\Users\Sanja\.vscode\extensions\ms-python.python-2025.18.0-win32-x64\bundled\libs\debug\launcher' '58718' '--' 'C:\Users\Sanja\OneDrive\Desktop\AI assistant coding\Ass 2.4.py'. The output shows the generated code running correctly.
- CHAT:** A sidebar titled "Build with Agent" with a message: "AI responses may be inaccurate. Generate Agent Instructions to onboard AI onto your codebase."
- SUGGESTED ACTIONS:** A panel at the bottom right with a button "Describe what to build next".

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.

The screenshot shows a VS Code interface with the following details:

- File Explorer:** Shows files Ass 2.4.py, Ass 1.4.py, and AI ASS... (with a status icon).
- Code Editor:** Displays the following Python code:

```
16 #Sorting Dictionaries with AI
17
18 def sort_by_age(users):
19     return sorted(users, key=lambda user: user["age"])
20
21 users = [
22     {"name": "Alice", "age": 25},
23     {"name": "Bob", "age": 20},
24     {"name": "Charlie", "age": 30}
25 ]
26
27
28 print(sort_by_age(users))
```
- Terminal:** Shows a PowerShell session output:

```
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>
PS C:\Users\Sanja\OneDrive\Desktop\AI assistant coding>
PS C:\Users\Sanja\OneDrive\Desktop\AI assistant coding> <�� cd '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' '58962' '--' '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>
```
- Suggested Actions:** Includes "Build Workspace" and "Show Config".

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.

The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows files like "Ass 2.4.py", "Ass 1.4.py", and "AI ASS...".
- Terminal:** Displays Python code for a calculator and its execution results.
- Code Editor:** Shows the "Ass 2.4.py" file containing a function-based calculator.
- Bottom Status Bar:** Shows file path (C:\Users\Sanja\OneDrive\Desktop\AI assistant coding\), file name (Ass 2.4.py), and other status indicators.

AI Assistant Coding Panel:

- Build with Agent:** A button to generate AI responses.
- AI Response:** A message stating "AI responses may be inaccurate. Generate Agent Instructions to onboard AI onto your codebase."
- Suggested Actions:** Buttons for "Build Workspace" and "Show Config".
- Build Log:** A list of build steps and their results.

Code Editor Content:

```
30
31 #calculator using Functions
32 def add(a, b):
33     return a + b
34 def subtract(a, b):
35     return a - b
36 def multiply(a, b):
37     return a * b
38 def divide(a, b):
39     if b == 0:
40         return "Error: Division by zero"
41     return a / b
42 print("Addition:", add(10, 5))
43 print("Subtraction:", subtract(10, 5))
44 print("Multiplication:", multiply(10, 5))
45 print("Division:", divide(10, 5))
46
47
48
```

Terminal Output:

```
PS C:\Users\Sanja\OneDrive\Desktop\AI assistant coding> cd 'c:\Users\Sanja\OneDrive\Desktop\AI assistant coding'; & 'c:\Users\Sanja\AppData\Local\Programs\Python\Python311\python.exe' 'c:\Users\Sanja\vscode\extensions\ms-python.debugpy-2025.18.0\src\Sanja\appdata\local\programs\python\python311\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

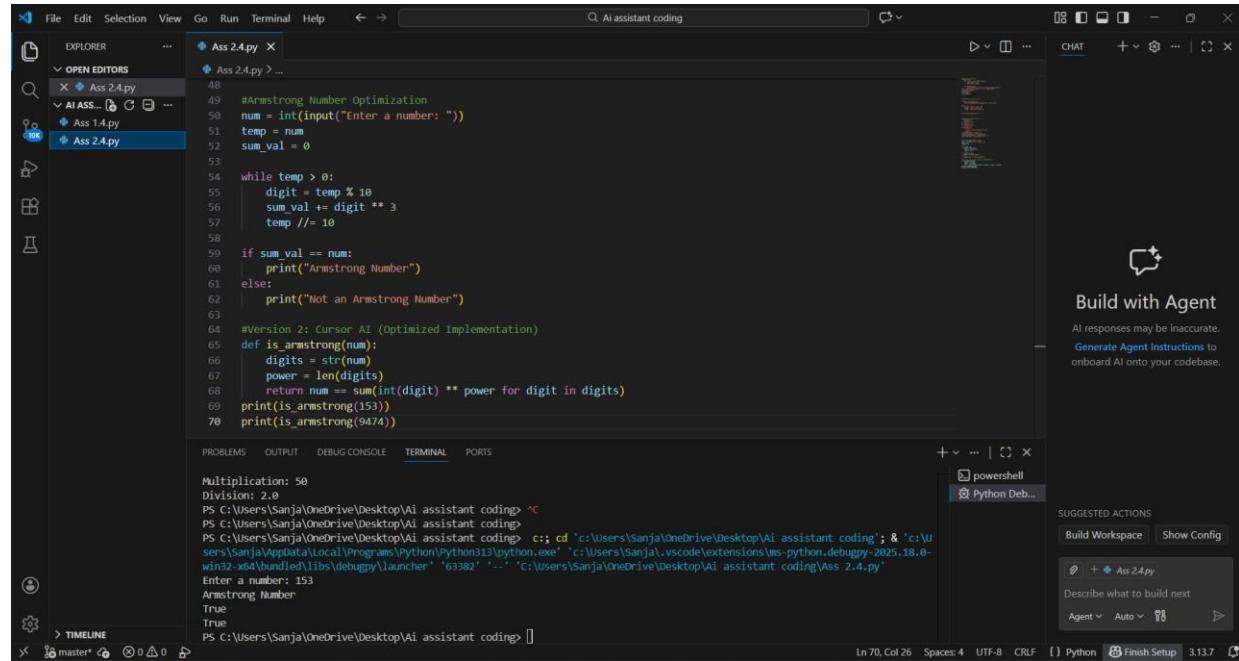
Task 4: Armstrong Number Optimization

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.



The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows two files: "Ass 2.4.py" and "Ass 1.4.py".
- Code Editor:** Displays the content of "Ass 2.4.py".

```
48
49 #Armstrong Number Optimization
50 num = int(input("Enter a number: "))
51 temp = num
52 sum_val = 0
53
54 while temp > 0:
55     digit = temp % 10
56     sum_val += digit ** 3
57     temp //= 10
58
59 if sum_val == num:
60     print("Armstrong Number")
61 else:
62     print("Not an Armstrong Number")
63
64 #Version 2: Cursor AI (Optimized Implementation)
65 def is_armstrong(num):
66     digits = str(num)
67     power = len(digits)
68     return num == sum(int(digit) ** power for digit in digits)
69 print(is_armstrong(153))
70 print(is_armstrong(9474))
```
- Terminal:** Shows the command-line output of running the script.

```
Multiplication: 58
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:\Users\Sanja\AppData\Local\Programs\Python\3.11\python.exe' 'c:\Users\Sanja\.vscode\extensions\ms-python.python.debuggy-2025.18.0-win32-x64\bundle\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> []
```
- Right Panel:** Includes a "Build with Agent" button, a message about AI responses being inaccurate, and a "SUGGESTED ACTIONS" section with "Build Workspace" and "Show Config" buttons.

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.