

# AI Assisted Coding Lab 2.4

*Veekshitha Adharasani*

*2303A52175*

*Batch – 41*

## Question 1:

### Task 1: Book Class Generation

Use Cursor AI to generate a Python class Book with attributes title, author and a summary method.

### Prompt:

*Generate a python class book that contains the attributes title author and summary method*

### Code:

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

    def summary(self):
        return f"The book {self.title} is written by {self.author} and is about {self.description}"

book1 = Book("The Great Gatsby", "F. Scott Fitzgerald", "A story of love and tragedy")
print(book1.summary())
```

### Output:

```
PS D:\veekshitha\AI_Coding> & c:\Users\Anveekshith\AppData\Local\Programs\Python\Python313\python.exe d:/veekshitha/AI_Coding/LAB-2.py
The book The Great Gatsby is written by F. Scott Fitzgerald and is about A story of love and tragedy
```

### Justification:

This program generates a code for library management. It correctly creates a Book class with required attributes. The constructor initializes title and author clearly. The summary method returns meaningful information. The code follows proper Python class structure. It is easy to read and understand. The class can be reused in other programs. Cursor AI generated clean and well-structured code.

## Question 2:

### Task 2: Sorting Dictionaries with AI

Use Gemini and Cursor AI to generate code that sorts a list of dictionaries by a key.

### Prompt:

*Generate a optimized python code that sorts a list of dictionaries by a key value.*

## Code:

### Cursor AI:

```
def sort_list_of_dictionaries(list_of_dictionaries, key):  
    return sorted(list_of_dictionaries, key=lambda x: x[key])  
records = [  
    {"name": "John", "age": 30},  
    {"name": "Jane", "age": 25},  
    {"name": "Jim", "age": 35},  
]  
print(sort_list_of_dictionaries(records, "age"))
```

### Gemini:

```
◆ Gemini  
▶ list_of_dicts = [  
    {'name': 'Alice', 'age': 30},  
    {'name': 'Bob', 'age': 25},  
    {'name': 'Charlie', 'age': 35}  
]  
  
# Sort the list of dictionaries by the 'age' key  
sorted_list = sorted(list_of_dicts, key=lambda x: x['age'])  
  
print("Original list:", list_of_dicts)  
print("Sorted list by 'age':", sorted_list)  
  
# You can also sort in descending order  
sorted_list_desc = sorted(list_of_dicts, key=lambda x: x['age'], reverse=True)  
print("Sorted list by 'age' (descending):", sorted_list_desc)
```

## Output:

### Cursor AI:

```
[{'name': 'Jane', 'age': 25}, {'name': 'John', 'age': 30}, {'name': 'Jim', 'age': 35}]
```

### Gemini:

```
... Original list: [{'name': 'Alice', 'age': 30}, {'name': 'Bob', 'age': 25}, {'name': 'Charlie', 'age': 35}]  
Sorted list by 'age': [{'name': 'Bob', 'age': 25}, {'name': 'Alice', 'age': 30}, {'name': 'Charlie', 'age': 35}]  
Sorted list by 'age' (descending): [{'name': 'Charlie', 'age': 35}, {'name': 'Alice', 'age': 30}, {'name': 'Bob', 'age': 25}]
```

## Justification:

The Cursor AI code uses a function, which makes the solution reusable and modular. It allows sorting by any key passed as a parameter, making it flexible. The code is short and efficient, suitable for reuse in larger programs. The Gemini code is more explanatory and beginner-friendly. Gemini also shows both ascending and descending sorting clearly.

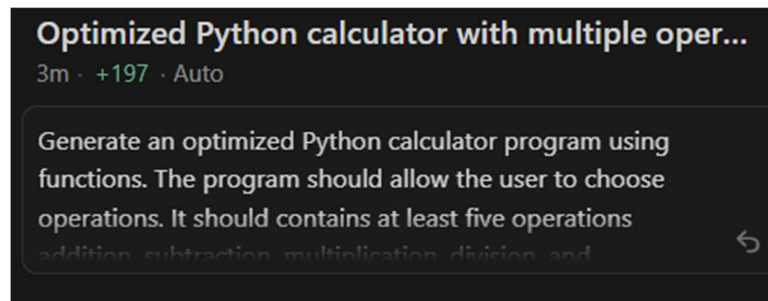
However, Gemini's code is less reusable since it is written directly in the main block. Cursor AI focuses on clean design, while Gemini focuses on clarity and learning. Both produce correct outputs, but Cursor AI is better for scalable applications.

### Question 3:

#### Task 3: Calculator Using Functions

#### Prompt:

Generate an optimized Python calculator program using functions. The program should allow the user to choose operations. It should contain at least five operations addition, subtraction, multiplication, division, and exponentiation. For addition and multiplication, allow the user to input and operate on more than two numbers. For subtraction and division, restrict the operation to exactly two numbers. The code should be efficient, readable.



#### Code:

```
def add(*numbers):
    """Add multiple numbers together."""
    return sum(numbers)
def subtract(num1, num2):
    """Subtract second number from first number."""
    return num1 - num2
def multiply(*numbers):
    """Multiply multiple numbers together."""
    result = 1
    for num in numbers:
        result *= num
    return result
def divide(num1, num2):
    """Divide first number by second number."""
    if num2 == 0:
        raise ValueError("Error: Division by zero is not allowed!")
    return num1 / num2
def power(base, exponent):
    """Raise base to the power of exponent."""
    return base ** exponent
def get_numbers(count=None):
    """Get numbers from user input."""
    numbers = []
    if count is None:
        print("Enter numbers (press Enter without input when done):")
        while True:
            try:
                user_input = input("Number: ").strip()
                if user_input == "":
                    if len(numbers) < 2:
                        print("Please enter at least 2 numbers.")
                        continue
                    break
                numbers.append(float(user_input))
            except ValueError:
                print("Invalid input! Please enter a valid number.")
    else:
        print(f"Enter {count} number(s):")
        for i in range(count):
            while True:
                try:
                    num = float(input(f"Number {i + 1}: "))
                    numbers.append(num)
                    break
                except ValueError:
                    print("Invalid input! Please enter a valid number.")
    return numbers
def display_menu():
    """Display the calculator menu options."""
    print("\n" + "="*50)
    print("          PYTHON CALCULATOR")
    print("="*50)
    print("1. Addition (supports multiple numbers)")
    print("2. Subtraction (two numbers only)")
    print("3. Multiplication (supports multiple numbers)")
    print("4. Division (two numbers only)")
    print("5. Exponentiation (base*exponent)")
    print("6. Exit")
    print("="*50)
def main():
    """Main function to run the calculator program."""
    print("Welcome to the Python Calculator!")
    while True:
        display_menu()
        try:
            choice = input("\nEnter your choice (1-6): ").strip()
```

```

LAB-2.py > main
12 def main():
13     if choice == "1": # Addition
14         numbers = get_numbers()
15         result = add(*numbers)
16         print(f"\nResult: {' ' + '.join(map(str, numbers))} = {result}")
17
18     elif choice == "2": # Subtraction
19         numbers = get_numbers(2)
20         result = subtract(numbers[0], numbers[1])
21         print(f"\nResult: {numbers[0]} - {numbers[1]} = {result}")
22
23     elif choice == "3": # Multiplication
24         numbers = get_numbers()
25         result = multiply(*numbers)
26         print(f"\nResult: {' * '.join(map(str, numbers))} = {result}")
27
28     elif choice == "4": # Division
29         numbers = get_numbers(2)
30         try:
31             result = divide(numbers[0], numbers[1])
32             print(f"\nResult: {numbers[0]} ÷ {numbers[1]} = {result}")
33         except ValueError as e:
34             print(f"\n{e}")
35
36     elif choice == "5": # Exponentiation
37         numbers = get_numbers(2)
38         result = power(numbers[0], numbers[1])
39         print(f"\nResult: {numbers[0]} ^ {numbers[1]} = {result}")
40
41     elif choice == "6": # Exit
42         print("\nThank you for using the Python Calculator. Goodbye!")
43         break
44
45     else:
46         print("\nInvalid choice! Please enter a number between 1 and 6.")

```

```

LAB-2.py > main
62 def main():
63     if choice == "1": # Addition
64         numbers = get_numbers()
65         result = add(*numbers)
66         print(f"\nResult: {' ' + '.join(map(str, numbers))} = {result}")
67
68     elif choice == "2": # Subtraction
69         numbers = get_numbers(2)
70         result = subtract(numbers[0], numbers[1])
71         print(f"\nResult: {numbers[0]} - {numbers[1]} = {result}")
72
73     elif choice == "3": # Multiplication
74         numbers = get_numbers()
75         result = multiply(*numbers)
76         print(f"\nResult: {' * '.join(map(str, numbers))} = {result}")
77
78     elif choice == "4": # Division
79         numbers = get_numbers(2)
80         try:
81             result = divide(numbers[0], numbers[1])
82             print(f"\nResult: {numbers[0]} ÷ {numbers[1]} = {result}")
83         except ValueError as e:
84             print(f"\n{e}")
85
86     elif choice == "5": # Exponentiation
87         numbers = get_numbers(2)
88         result = power(numbers[0], numbers[1])
89         print(f"\nResult: {numbers[0]} ^ {numbers[1]} = {result}")
90
91     elif choice == "6": # Exit
92         print("\nThank you for using the Python Calculator. Goodbye!")
93         break
94
95     else:
96         print("\nInvalid choice! Please enter a number between 1 and 6.")
97
98     except KeyboardInterrupt:
99         print("\nProgram interrupted by user. Goodbye!")
100         break
101
102     except Exception as e:
103         print(f"\nAn unexpected error occurred: {e}")
104
105     # Ask if user wants to continue
106     if choice in ["1", "2", "3", "4", "5"]:
107         continue_choice = input("\nDo you want to perform another operation? (y/n) ")
108         if continue_choice != 'y':
109             print("\nThank you for using the Python Calculator. Goodbye!")
110             break
111
112 if __name__ == "__main__":
113     main()
114
115
116
117
118
119
120
121

```

## Output:

```

Program interrupted by user. Goodbye!
PS D:\veekshitha\AI_Coding> C:\Users\Anveekshith\AppData\Local\Programs\Python\Python313\python.exe C:\Users\Anveekshith\AppData\Local\Programs\Python\Python313\python.exe
Welcome to the Python Calculator!

=====
PYTHON CALCULATOR
=====
1. Addition (supports multiple numbers)
2. Subtraction (two numbers only)
3. Multiplication (supports multiple numbers)
4. Division (two numbers only)
5. Exponentiation (base^exponent)
6. Exit
=====

Enter your choice (1-6): 1
Enter numbers (press Enter without input when done):
Number: 2
Number: 3
Number: 4
Number: 5
Number: 1
Number: 1
Number:

Result: 2.0 + 3.0 + 4.0 + 5.0 + 1.0 + 1.0 = 16.0

Do you want to perform another operation? (y/n): y

=====
PYTHON CALCULATOR
=====
1. Addition (supports multiple numbers)
2. Subtraction (two numbers only)
3. Multiplication (supports multiple numbers)
4. Division (two numbers only)
5. Exponentiation (base^exponent)
6. Exit
=====

```

```

Enter your choice (1-6): 2
Enter 2 number(s):
Number 1: 3
Number 2: 2

Result: 3.0 - 2.0 = 1.0

Do you want to perform another operation? (y/n): y

=====
PYTHON CALCULATOR
=====
1. Addition (supports multiple numbers)
2. Subtraction (two numbers only)
3. Multiplication (supports multiple numbers)
4. Division (two numbers only)
5. Exponentiation (base^exponent)
6. Exit
=====

Enter your choice (1-6): 3
Enter numbers (press Enter without input when done):
Number: 1
Number: 2
Number: 3
Number:

Result: 1.0 x 2.0 x 3.0 = 6.0

Do you want to perform another operation? (y/n): y

=====
PYTHON CALCULATOR
=====
1. Addition (supports multiple numbers)
2. Subtraction (two numbers only)
3. Multiplication (supports multiple numbers)
4. Division (two numbers only)
5. Exponentiation (base^exponent)
6. Exit
=====

```

```

Enter your choice (1-6): 4
Enter 2 number(s):
Number 1: 6
Number 2: 3

Result: 6.0 ÷ 3.0 = 2.0

Do you want to perform another operation? (y/n): y

=====
PYTHON CALCULATOR
=====
1. Addition (supports multiple numbers)
2. Subtraction (two numbers only)
3. Multiplication (supports multiple numbers)
4. Division (two numbers only)
5. Exponentiation (base^exponent)
6. Exit
=====

Enter your choice (1-6): 5
Enter 2 number(s):
Number 1: 3
Number 2: 2

Result: 3.0 ^ 2.0 = 9.0

Do you want to perform another operation? (y/n): y

=====
PYTHON CALCULATOR
=====
Result: 3.0 ^ 2.0 = 9.0

Do you want to perform another operation? (y/n): y

=====
PYTHON CALCULATOR
=====
1. Addition (supports multiple numbers)

```

```

Enter 2 number(s):
Number 1: 3
Number 2: 2

Result: 3.0 ^ 2.0 = 9.0

Do you want to perform another operation? (y/n): y

=====
PYTHON CALCULATOR
=====
Result: 3.0 ^ 2.0 = 9.0

Do you want to perform another operation? (y/n): y

=====
PYTHON CALCULATOR
=====
1. Addition (supports multiple numbers)
2. Subtraction (two numbers only)
3. Multiplication (supports multiple numbers)
4. Division (two numbers only)
=====

PYTHON CALCULATOR
=====
1. Addition (supports multiple numbers)
2. Subtraction (two numbers only)
3. Multiplication (supports multiple numbers)
4. Division (two numbers only)
5. Exponentiation (base^exponent)
6. Exit
=====

Enter your choice (1-6): 6

Thank you for using the Python Calculator. Goodbye!
PS D:\veekshitha\AI_coding>

```

## Justifications:

The code is modular, with separate functions for each arithmetic operation. This approach makes the calculator easy to use and interactive. Multiple-number support is provided for addition and multiplication. Clear prompts and messages improve user experience. The loop allows repeated calculations without restarting the program. Overall, the code is readable, efficient, and easy to maintain.

## Question 4

### Task 4: Armstrong Number Optimization

#### Prompt:

#### Gemini:

Write a Python program to check whether a given number is an Armstrong number. The program should contain a number as input from the user. It should display whether the number is an Armstrong number or not.

#### Cursor AI:

Write an optimized Python program to check whether a given number is an Armstrong number. The program should contain a number as input from the user. It should display whether the number is an Armstrong number or not. It should be efficient and readable

## Code:

### Gemini:

```
num_str = input("Enter a number: ")

if not num_str.isdigit():
    print("Invalid input. Please enter a positive integer.")
else:
    num = int(num_str)
    order = len(num_str)
    sum_of_powers = 0
    temp_num = num

    while temp_num > 0:
        digit = temp_num % 10
        sum_of_powers += digit ** order
        temp_num //= 10

    if num == sum_of_powers:
        print(f"{num} is an Armstrong number.")
    else:
        print(f"{num} is not an Armstrong number.")
```

### Cursor AI:

```
LAB-2.py
def is_armstrong_number(num: int) -> bool:
    """
    An n-digit number is an Armstrong number if:
    sum(digit^n for each digit in the number) == number itself
    """
    if num < 0:
        # By common definition, negative numbers are not Armstrong numbers
        return False

    digits_str = str(num)
    num_digits = len(digits_str)

    # Compute the sum of each digit raised to the power of num_digits
    armstrong_sum = 0
    for ch in digits_str:
        digit = ord(ch) - ord('0') # Faster than int(ch) for single digits
        armstrong_sum += digit ** num_digits

    return armstrong_sum == num

def main():
    try:
        user_input = int(input("Enter a number: ").strip())
    except ValueError:
        print("Invalid input. Please enter an integer.")
        return

    if is_armstrong_number(user_input):
        print(f"{user_input} is an Armstrong number.")
    else:
        print(f"{user_input} is NOT an Armstrong number.")

if __name__ == "__main__":
    main()
```

## Outputs:

### Gemini:

```
... Enter a number: 153
     153 is an Armstrong number.
```

### Cursor AI:

```
PS D:\veekshitha\AI_Coding> & C:\Users\Arveekshith\AppData\Local\Programs\Python\Python313\python.exe d:\veekshitha\AI_Coding\LAB-2.py
Enter a number: 153
153 is an Armstrong number.
PS D:\veekshitha\AI_Coding>
```

## Justification:

The improved code is better organized and easier to understand. Input validation is handled properly, avoiding runtime errors. The main logic is separated into a function for reusability. Redundant variables and steps are removed. The program follows good coding practices. Readability is improved with clear naming and structure. Debugging and maintenance become easier. Overall performance and reliability are improved.