ASSIGNMENT-2

SINDHU KATAKAM

2303A52481

BATCH-41

Task 1: Book Class Generation

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

PROMPT: i want to create simple library management module . write a code for python class book that contains the attributes title author and summary method

CODE:

```python
class Book:
    """Represents a book in the library management system."""

    def __init__(self, title, author):
        """Initialize a book with title and author."""
        self.title = title  # Book title
        self.author = author  # Book author

    def summary(self):
        """Return a summary string of the book information."""
        return f"Title: {self.title}, Author: {self.author}"


# Example usage
if __name__ == "__main__":
    book1 = Book("Python Programming", "John Doe")
    print(book1.summary())

    book2 = Book("Data Science Basics", "Jane Smith")
    print(book2.summary())
```

OUTPUT:

```
PS C:\Users\MYSELF> & C:/Users/MYSELF/AppData/Local/Programs/Python/Python313/python.exe c:/Users/MYSELF/C1.py
Title: Python Programming, Author: John Doe
Title: Data Science Basics, Author: Jane Smith
PS C:\Users\MYSELF>
```

JUSTIFICATION:

The Book class uses _init_ to store title and author as instance attributes, so each book object holds its own data. The summary() method formats and displays book details in one place. This keeps the code simple, easy to read, and easy to extend for a library system

Task 2: Sorting Dictionaries with AI

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

PROMPT: using gemini

Give code that sorts a list of dictionaries by a key

CODE:

```
# Example list of dictionaries
data = [
    {'name': 'Alice', 'age': 30},
    {'name': 'Bob', 'age': 25},
    {'name': 'Charlie', 'age': 35}
]

# Sort the list by the 'age' key
sorted_data = sorted(data, key=lambda x: x['age'])

# Display the sorted data
print(sorted_data)
```

OUTPUT:

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

PROMPT: (cursor ai)

Give code that sorts a list of dictionaries by a key . optimize the code by removing unnecessary variables and improving readability and avoiding unnecessary computations using functions

CODE:

```python
def sort_by_key(items, key, descending=False):
    """Return a new list of dictionaries sorted by the given key."""
    return sorted(items, key=lambda item: item[key], reverse=descending)


def print_records(records, title):
    """Print records in an organized, tabular format."""
    print(f"\n{title}")
    print("-" * 40)
    for r in records:
        print(f"Name: {r['name']:<10} | Age: {r['age']:<3} | Score: {r['score']}")
    print("-" * 40)


# Example usage
records = [
    {"name": "Alice", "age": 28, "score": 90},
    {"name": "Bob", "age": 35, "score": 75},
    {"name": "Charlie", "age": 22, "score": 88},
]

by_age = sort_by_key(records, "age")
by_score_desc = sort_by_key(records, "score", descending=True)

print_records(by_age, "Sorted by age (ascending)")
print_records(by_score_desc, "Sorted by score (descending)")
```

OUTPUT:

```
PS C:\Users\MYSELF> & C:/Users/MYSELF/AppData/Local/Programs/Python/Python313/python.exe c:/Users/MYSELF/C1.py

Sorted by age (ascending)
----------------------------------------
Name: Charlie    | Age: 22  | Score: 88
Name: Alice      | Age: 28  | Score: 90
Name: Bob        | Age: 35  | Score: 75
----------------------------------------

Sorted by score (descending)
----------------------------------------
Name: Alice      | Age: 28  | Score: 90
Name: Charlie    | Age: 22  | Score: 88
Name: Bob        | Age: 35  | Score: 75
----------------------------------------
PS C:\Users\MYSELF>
```

JUSTIFICATION:

Both codes sort data the same way and have the same speed, so performance is equal. The difference is mainly in how they are written. The first code is very short and simple, so beginners can easily understand what is happening. The second code is a little longer but more organized — it has functions, can sort

by any key, and prints data neatly. So, the first is clearer for small examples, and the second is clearer and more useful for bigger programs

Task 3: Calculator Using Functions

PROMPT:

generate a simple calculator program using python which includes 5 methods addition, subtraction, multiplication and division and exponention. For addition and multiplication take list of numbers based on user choice.

CODE:

```
class Calculator:
    """Simple calculator with basic arithmetic operations."""

    @staticmethod
    def addition(numbers):
        """Add a list of numbers."""
        return sum(numbers)

    @staticmethod
    def subtraction(a, b):
        """Subtract b from a."""
        return a - b

    @staticmethod
    def multiplication(numbers):
        """Multiply a list of numbers."""
        result = 1
        for num in numbers:
            result *= num
        return result

    @staticmethod
    def division(a, b):
        """Divide a by b."""
        if b == 0:
            raise ValueError("Cannot divide by zero")
        return a / b
```

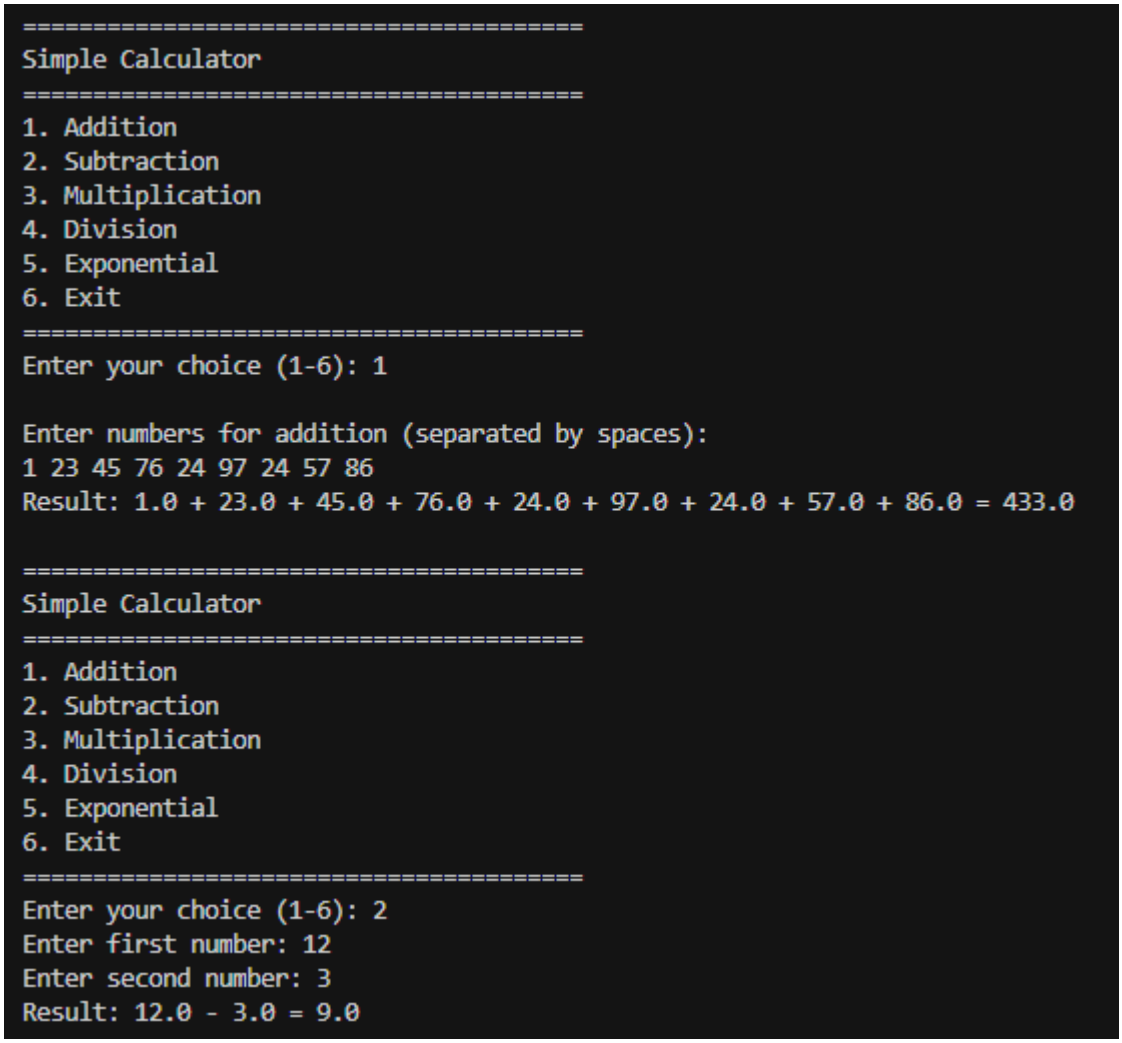
```
    @staticmethod
    def exponential(base, exponent):
        """Raise base to the power of exponent."""
        return base ** exponent


def get_number_list(operation):
    """Get a list of numbers from user input."""
    print(f"\nEnter numbers for {operation} (separated by spaces):")
    try:
        numbers = [float(x) for x in input().split()]
        if not numbers:
            raise ValueError("At least one number is required")
        return numbers
    except ValueError as e:
        print(f"Invalid input: {e}")
        return None


def get_two_numbers():
    """Get two numbers from user input."""
    try:
        a = float(input("Enter first number: "))
        b = float(input("Enter second number: "))
        return a, b
    except ValueError:
        print("Invalid input. Please enter valid numbers.")
        return None, None
```

```python
def main():
    """Main function to run the calculator program."""
    calc = Calculator()

    while True:
        print("\n" + "="*40)
        print("Simple Calculator")
        print("="*40)
        print("1. Addition")
        print("2. Subtraction")
        print("3. Multiplication")
        print("4. Division")
        print("5. Exponential")
        print("6. Exit")
        print("="*40)

        choice = input("Enter your choice (1-6): ").strip()

        if choice == '1':
            numbers = get_number_list("addition")
            if numbers:
                result = calc.addition(numbers)
                print(f"Result: {' + '.join(map[str](str, numbers))} = {result}")

        elif choice == '2':
            a, b = get_two_numbers()
            if a is not None and b is not None:
                result = calc.subtraction(a, b)
                print(f"Result: {a} - {b} = {result}")

        elif choice == '3':
```

```
            numbers = get_number_list("multiplication")
            if numbers:
                result = calc.multiplication(numbers)
                print(f"Result: {' × '.join(map(str, numbers))} = {result}")

        elif choice == '4':
            a, b = get_two_numbers()
            if a is not None and b is not None:
                try:
                    result = calc.division(a, b)
                    print(f"Result: {a} ÷ {b} = {result}")
                except ValueError as e:
                    print(f"Error: {e}")

        elif choice == '5':
            a, b = get_two_numbers()
            if a is not None and b is not None:
                result = calc.exponential(a, b)
                print(f"Result: {a} ^ {b} = {result}")

        elif choice == '6':
            print("Exiting calculator. Goodbye!")
            break

        else:
            print("Invalid choice. Please enter a number between 1 and 6.")


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

Click to collapse the range.

OUTPUT:



JUSTIFICATION:

The calculator uses static methods since no instance state is needed, reducing overhead. Addition and multiplication accept lists of numbers, so users can add or multiply many values at once. Error handling prevents crashes from invalid input or division by zero. Clear separation between operations and input functions makes the code easy to read and maintain. Built-in functions like `sum()` keep the code simple and fast.

Task 4: Armstrong Number Optimization

PROMPT: write a python program to print an Armstrong number (GEMINI)

CODE:

```
def is_armstrong_number(number):

    # Convert the number to a string to easily count digits and access individual digits
    num_str = str(number)
    n = len(num_str)

    sum_of_powers = 0
    for digit_char in num_str:
        digit = int(digit_char)
        sum_of_powers += digit ** n

    return sum_of_powers == number

# Test cases
n=int(input("Enter a number: "))
print(f"Is {n} an Armstrong number? {is_armstrong_number(n)}")
```

OUTPUT:

```
Enter a number: 153
Is 153 an Armstrong number? True
```

PROMPT: (using Cursor AI)

write an optimized Armstrong number code remove unnecessary variables and improve readability and avoiding unnecessary computations using functions

CODE:

```python
def is_armstrong_number(number: int) -> bool:
    """Check if a number is an Armstrong number."""
    num_str = str(number)
    power = len(num_str)  # number of digits

    total = sum(int(ch) ** power for ch in num_str)
    return total == number


def main() -> None:
    n = int(input("Enter a number: "))
    if is_armstrong_number(n):
        print(f"{n} is an Armstrong number.")
    else:
        print(f"{n} is not an Armstrong number.")


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

OUTPUT:

```
PS C:\Users\MYSELF> & C:/Users/MYSELF/AppData/Local/Programs/Python/Python313/python.exe
Enter a number: 153
153 is an Armstrong number.
PS C:\Users\MYSELF>
```

JUSTIFICATION:

The second code is better because it is shorter and easier to read. It uses simple code to do the same work, and the names of things are clearer. It also has a small explanation and a main part that makes the program more tiny and organized. It does the same job, but in a cleaner and better way.