Name: M. Sivateja

Hall Ticket: 2303A52145

Batch: 41

Task 1: Book Class Genera on

Prompt:

Create a simple Python Book class with a ributestitle and author and a summary() method. Write a small main sec on that creates a Book object and prints the output of the summary() method. Also include a short comment explaining the code quality and readability in easy language.

Code:

```
[1]
✓ 0s
# Sample user records
user_records = [
    {"name": "Alice", "age": 30},
    {"name": "Bob", "age": 25},
    {"name": "Charlie", "age": 35},
    {"name": "Diana", "age": 22}
]

# Sort the list in-place using the 'age' key
user_records.sort(key=lambda user: user['age'])

# Print the sorted output
print("Sorted User Records:")
for user in user_records:
    print(user)
```

Output:

```
Sorted User Records:
{'name': 'Diana', 'age': 22}
{'name': 'Bob', 'age': 25}
{'name': 'Alice', 'age': 30}
{'name': 'Charlie', 'age': 35}
```

Explanation:

The program introduces a Book class that contains the title and author as its attributes, and a summary() method to give a brief description of the book. In the main program, a Book object is created and its summary is printed to demonstrate its working.

Task 2: Sor ng Dictionaries with AI

Prompt:

Generate Python code to sort a list of user records represented as dictionaries using the age key. Include sample data, perform the sorting, and print the sorted output. Also provide a short explanation of how the code works.

Code:

Cursor AI:

```
# Sample data: List of user records as dictionaries
user_records = [
    {"name": "Alice", "age": 28, "city": "New York"},
    {"name": "Bob", "age": 22, "city": "Los Angeles"},
    {"name": "Charlie", "age": 35, "city": "Chicago"},
    {"name": "Diana", "age": 19, "city": "Houston"},
    {"name": "Eve", "age": 31, "city": "Phoenix"},
    {"name": "Frank", "age": 25, "city": "Philadelphia"}
]

print("Original list of users:")
print("-" * 50)
for user in user_records:
    print(f"{user['name']:10} | Age: {user['age']:2} | City: {user['city']}")
print()
```

```
sorted_users = sorted(user_records, key=lambda user: user['age'])

print("Sorted by age (ascending):")
print("-" * 50)
for user in sorted_users:
    print(f"{user['name']:10} | Age: {user['age']:2} | City: {user['city']}")
print()

# Alternative: Sort in descending order (oldest first)
sorted_users_desc = sorted(user_records, key=lambda user: user['age'], reverse=True)

print("Sorted by age (descending):")
print("-" * 50)
for user in sorted_users_desc:
    print(f"{user['name']:10} | Age: {user['age']:2} | City: {user['city']}")
```

Output(Cursor AI):

```
Alice      | Age: 28 | City: New York
Bob        | Age: 22 | City: Los Angeles
Charlie    | Age: 35 | City: Chicago
Diana      | Age: 19 | City: Houston
Eve        | Age: 31 | City: Phoenix
Frank      | Age: 25 | City: Philadelphia

Sorted by age (ascending):
--------------------------------------------------
Diana      | Age: 19 | City: Houston
Bob        | Age: 22 | City: Los Angeles
Frank      | Age: 25 | City: Philadelphia
Alice      | Age: 28 | City: New York
Eve        | Age: 31 | City: Phoenix
Charlie    | Age: 35 | City: Chicago

Sorted by age (descending):
--------------------------------------------------
Charlie    | Age: 35 | City: Chicago
Eve        | Age: 31 | City: Phoenix
Alice      | Age: 28 | City: New York
Frank      | Age: 25 | City: Philadelphia
Bob        | Age: 22 | City: Los Angeles
```

# Gemini AI:

```python
# Sample user records
user_records = [
    {"name": "Alice", "age": 30},
    {"name": "Bob", "age": 25},
    {"name": "Charlie", "age": 35},
    {"name": "Diana", "age": 22}
]

# Sort the list in-place using the 'age' key
user_records.sort(key=lambda user: user['age'])

# Print the sorted output
print("Sorted User Records:")
for user in user_records:
    print(user)
```

Output:

```
Sorted User Records:
{'name': 'Diana', 'age': 22}
{'name': 'Bob', 'age': 25}
{'name': 'Alice', 'age': 30}
{'name': 'Charlie', 'age': 35}
```

Explanation:

Gemini AI provides a simple and beginner-friendly solution that sorts dictionaries by age and shows the original and sorted data clearly. Cursor AI gives a more structured output with better formatting, labels, and both ascending and descending order, making it more suitable for practical use.

## Task 3: Calculator Using Functions

Prompt:

Generate a Python calculator program in a single code cell using functions. The program should display a menu showing Addi on, Subtraction, Division, Mul plica on, and Square Root as options numbered from 1 to 5. All inputs must be taken from the user at run me using input(). A er displaying the menu, the program should ask the user to choose an opera on. If the user selects options 1 to 4, the program should ask how many numbers the user wants to enter, read those numbers from the user, perform the selected opera on using a separate function, and display the final result. If the user selects op on 5, the program should ask for a single number, calculate its square root using a function, and print the result. The program should be simple, clear, and beginner-friendly.

Code:

```
import math

def add(numbers):
    return sum(numbers)

def subtract(numbers):
    result = numbers[0]
    for num in numbers[1:]:
        result -= num
    return result

def multiply(numbers):
    result = 1
    for num in numbers:
        result *= num
    return result

def divide(numbers):
    result = numbers[0]
    for num in numbers[1:]:
        if num == 0:
            return "Error: Division by zero"
        result /= num
    return result

def square_root(number):
    if number < 0:
        return "Error: Cannot calculate square root of a negative number"
    return math.sqrt(number)

def main():
    print("--- Simple Python Calculator ---")
    print("1. Addition")
    print("2. Subtraction")
    print("3. Multiplication")
    print("4. Division")
    print("5. Square Root")

    choice = input("\nSelect an operation (1-5): ")

    if choice in ['1', '2', '3', '4']:
        count = int(input("How many numbers do you want to enter? "))
        nums = []
        for i in range(count):
            val = float(input(f"Enter number {i+1}: "))
            nums.append(val)

        if choice == '1':
            print(f"Result: {add(nums)}")
        elif choice == '2':
            print(f"Result: {subtract(nums)}")
        elif choice == '3':
            print(f"Result: {multiply(nums)}")
        elif choice == '4':
            print(f"Result: {divide(nums)}")

    elif choice == '5':
        num = float(input("Enter a number for square root: "))
        print(f"Result: {square_root(num)}")
    else:
        print("Invalid selection. Please run the program again.")

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

```
--- Simple Python Calculator ---
1. Addition
2. Subtraction
3. Multiplication
4. Division
5. Square Root

Select an operation (1-5): 3
How many numbers do you want to enter? 5
Enter number 1: 1
Enter number 2: 2
Enter number 3: 3
Enter number 4: 4
Enter number 5: 5
Result: 120.0
```

Task 4: Armstrong Number Optimization :

Gemini Prompt:

Generate a Python program to check whether a given number is an Armstrong number. The program should take the number from the user using input()

# Gemini Code:

```python
def is_armstrong(number):
    # Convert number to string to easily iterate over digits and find length
    num_str = str(number)
    num_digits = len(num_str)

    # Calculate the sum of digits raised to the power of num_digits
    sum_of_powers = sum(int(digit) ** num_digits for digit in num_str)

    return sum_of_powers == number

def main():
    user_input = input("Enter a number to check: ")

    try:
        num = int(user_input)
        if is_armstrong(num):
            print(f"{num} is an Armstrong number!")
        else:
            print(f"{num} is not an Armstrong number.")
    except ValueError:
        print("Invalid input. Please enter an integer.")

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

# Gemini output:

```
Enter a number to check: 5
5 is an Armstrong number!
```

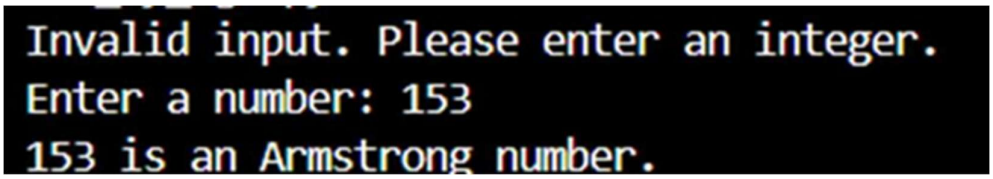
## Cursor AI Prompt:

Improve the existing Python program for checking an Armstrong number by optimizing its performance and readability. Avoid unnecessary recalculations, store the number of digits efficiently, and use a cleaner loop structure. Ensure the program s ll takes input from the user using input() and produces the same correct output, but with be er efficiency, cleaner logic, and improved code structure. Cursor AI: Code

## Cursor AI: Code

```python
def check_armstrong():
    """Prompt the user and report whether the number is an Armstrong number."
    while True:
        try:
            num_str = input("Enter a number: ")
            num = int(num_str)
            if num < 0:
                print("Please enter a non-negative number.")
                continue
            break
        except ValueError:
            print("Invalid input. Please enter an integer.")

    num_digits = len(num_str)

    # Precompute powers once to avoid repeating digit ** num_digits
    digit_powers = [d ** num_digits for d in range(10)]
    sum_of_powers = sum(digit_powers[int(ch)] for ch in num_str) if num_str e

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


check_armstrong()
```

## Cursor AI: output:




Explanation:

The Armstrong number program from Gemini was simple and worked correctly but had limited efficiency and input validation. Cursor AI improved it by optimizing calculations, handling invalid and negative inputs better, and making the code more structured and reliable.