Assesment=2.4

Abhishek
2303a52469
batch - 43

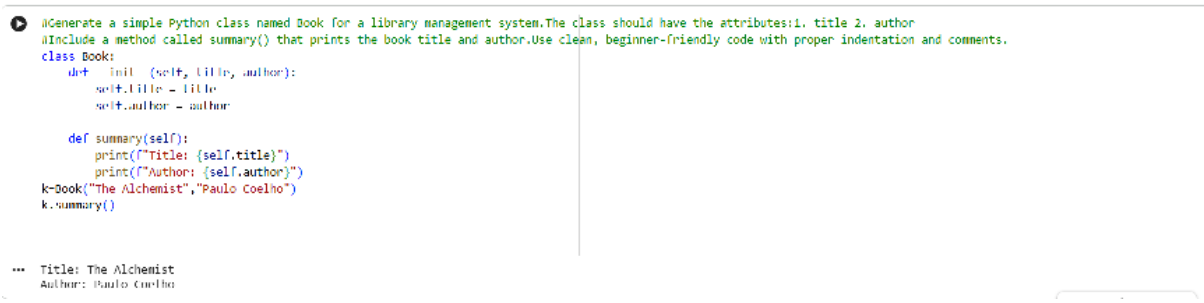
Task 1:

Prompt: #Generate a Python class named Book for a simple library management system.

The class should have attributes title and author.

Include a method summary that prints the book details.

Use clean, readable, beginner-friendly code.



```
#Generate a simple Python class named Book for a library management system.The class should have the attributes:1. title 2. author
#Include a method called summary() that prints the book title and author.Use clean, beginner-friendly code with proper indentation and comments.
class Book:
    def __init__(self, title, author):
        self.title = title
        self.author = author

    def summary(self):
        print(f"Title: {self.title}")
        print(f"Author: {self.author}")
k=Book("The Alchemist","Paulo Coelho")
k.summary()
```

```
Title: The Alchemist
Author: Paulo Coelho
```

Observation:

The Python class Book was successfully implemented using object-oriented programming concepts. The class contains appropriate attributes for storing

the book title and author. The constructor correctly initializes

the object with given values.

The summary method effectively displays the book details in a readable format. The code is simple, well-structured, and easy to understand for beginners. Proper indentation and meaningful variable names improve code

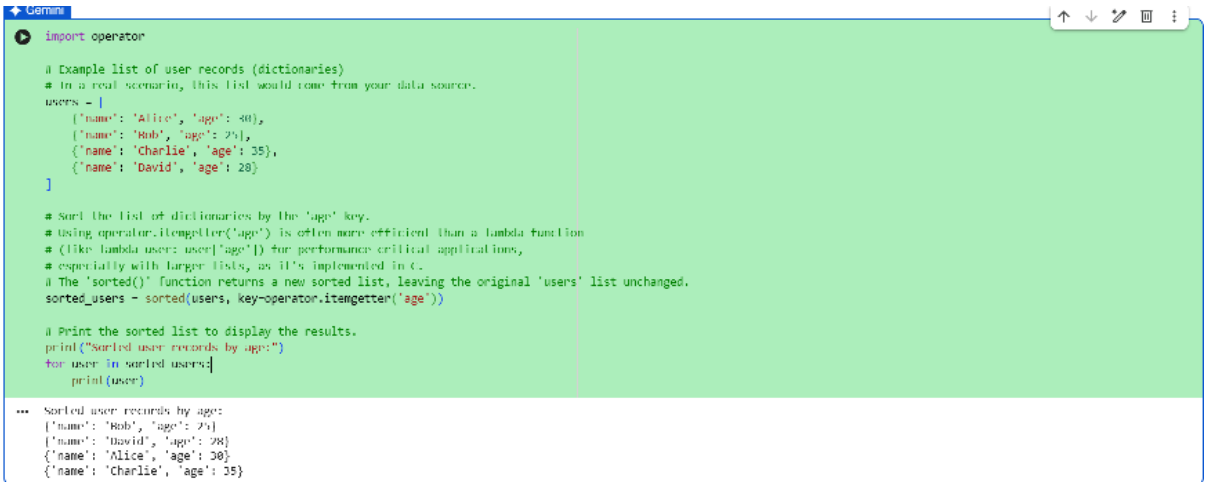
readability and maintainability.

Task 2:

Prompt: # Write a Python program to sort a list of dictionaries by age.

Each dictionary should contain name and age.

Explain the code clearly.



Observation:

The program successfully sorts a list of user records stored as dictionaries based on the 'age' key. The use of improves code readability and provides better performance compared to a lambda function.

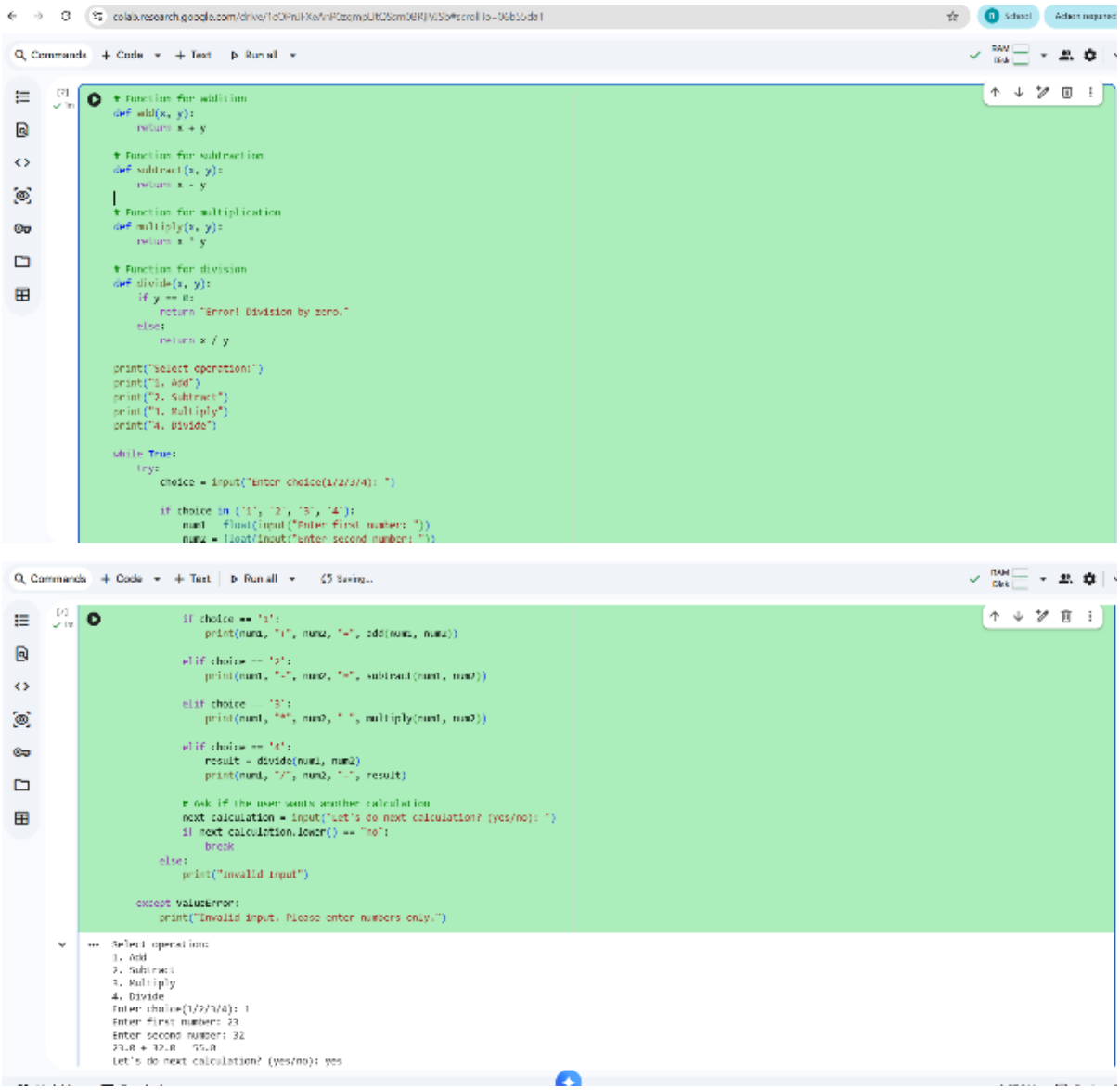
The sorted () function efficiently returns a new sorted list without modifying the original data. The output is displayed clearly, showing users arranged in ascending order of age. Overall, the code is well-structured, efficient, and suitable for handling larger datasets.

Task 3:

Prompt: #Create a basic calculator in Python using functions.

Include addition, subtraction, multiplication, and division.

Explain how the program works step by step.



```
# Function for addition
def add(x, y):
    return x + y

# Function for subtraction
def subtract(x, y):
    return x - y

# Function for multiplication
def multiply(x, y):
    return x * y

# Function for division
def divide(x, y):
    if y == 0:
        return "Error! Division by zero."
    else:
        return x / y

print("Select operation:")
print("1. Add")
print("2. Subtract")
print("3. Multiply")
print("4. Divide")

while True:
    try:
        choice = input("Enter choice(1/2/3/4): ")

        if choice in ('1', '2', '3', '4'):
            num1 = float(input("Enter first number: "))
            num2 = float(input("Enter second number: "))

            if choice == '1':
                print(num1, "+", num2, "=", add(num1, num2))

            elif choice == '2':
                print(num1, "-", num2, "=", subtract(num1, num2))

            elif choice == '3':
                print(num1, "*", num2, " ", multiply(num1, num2))

            elif choice == '4':
                result = divide(num1, num2)
                print(num1, "/", num2, "=", result)

            # Ask if the user wants another calculation
            next_calculation = input("Let's do next calculation? (yes/no): ")
            if next_calculation.lower() == "no":
                break
        else:
            print("Invalid input")

    except ValueError:
        print("Invalid input. Please enter numbers only.")
```

```
Select operation:
1. Add
2. Subtract
3. Multiply
4. Divide
Enter choice(1/2/3/4): 1
Enter first number: 23
Enter second number: 32
23.0 + 32.0 = 55.0
Let's do next calculation? (yes/no): yes
```

Observation:

The program successfully implements a basic calculator using Python functions.

Each arithmetic operation such as addition, subtraction, multiplication, and

division is defined as a separate function, which improves modularity and code

reuse.

The program uses a menu-driven approach and accepts user input dynamically.

Error handling is implemented using try-except blocks to manage invalid numeric

inputs, and division by zero is handled gracefully.

The while loop allows multiple calculations without restarting the program.

Overall, the code is well-structured, user-friendly, and demonstrates effective

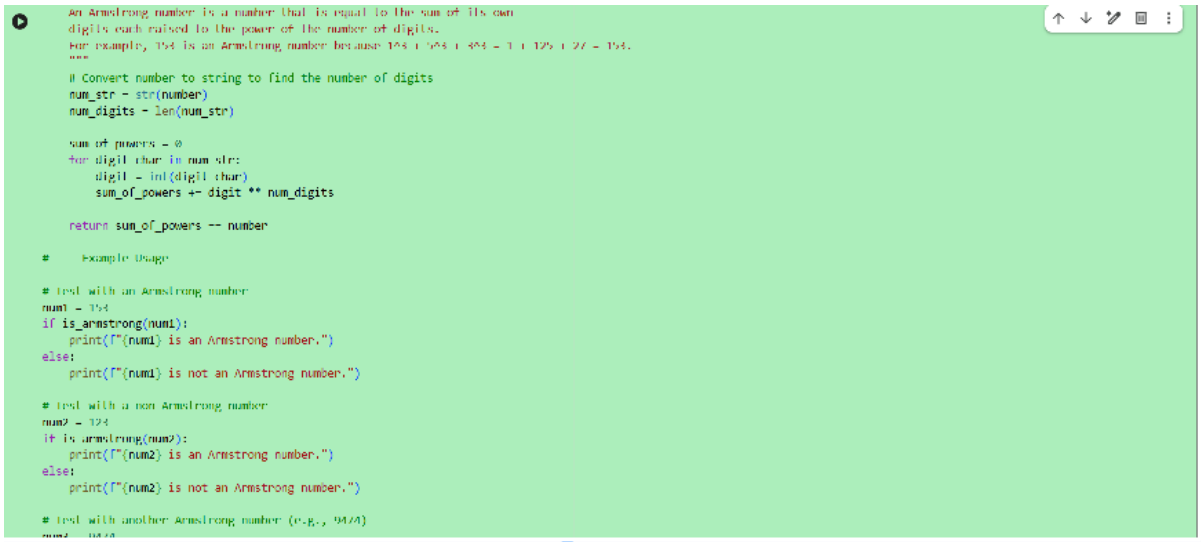
use of functions and control flow in Python.

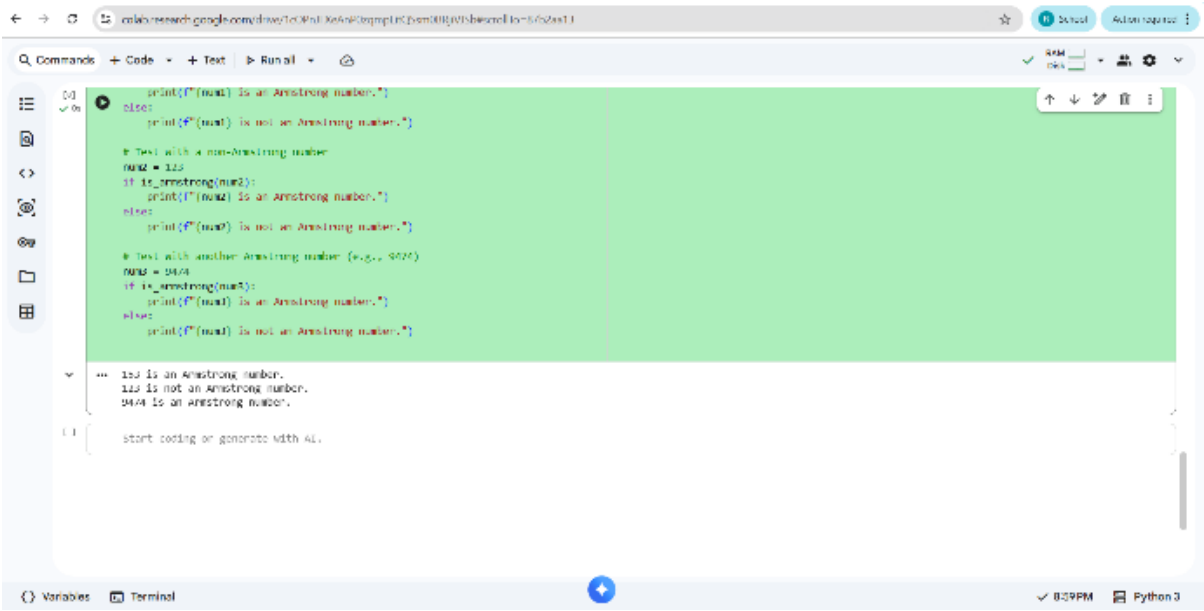
Task 4:

Prompt:

#Write a Python program to check whether a number is an Armstrong number.

Explain the logic.





Observation:

The program correctly checks whether a given number is an Armstrong number.

It converts the number into a string to determine the number of digits,

which simplifies the calculation process.

Each digit is raised to the power of the total number of digits and summed efficiently using a loop. The function returns a Boolean value, making the code reusable and easy to test with multiple inputs.

Clear documentation and comments improve code readability and understanding.

The program produces accurate results for both Armstrong and non-Armstrong

numbers, demonstrating correct logic and efficient implementation.