Assignment 2.4
2303A510F2 - Anitha
Batch - 14

Lab 2: Exploring Additional AI Coding Tools beyond Copilot – Gemini (Colab) and Cursor AI

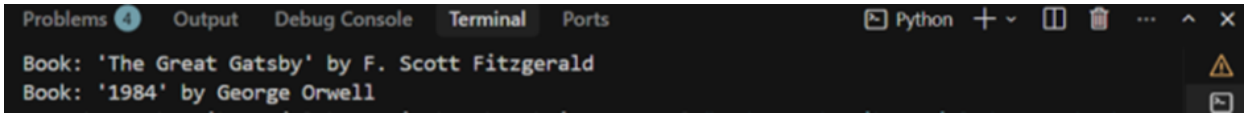
Task 1: Book Class Generation

Prompt:

Generate a Python class named Book with attributes title and author. Include a method summary() that returns a formatted string displaying the book title and author. Write clean, beginner-friendly code.

Code :

Output:



Explanation:

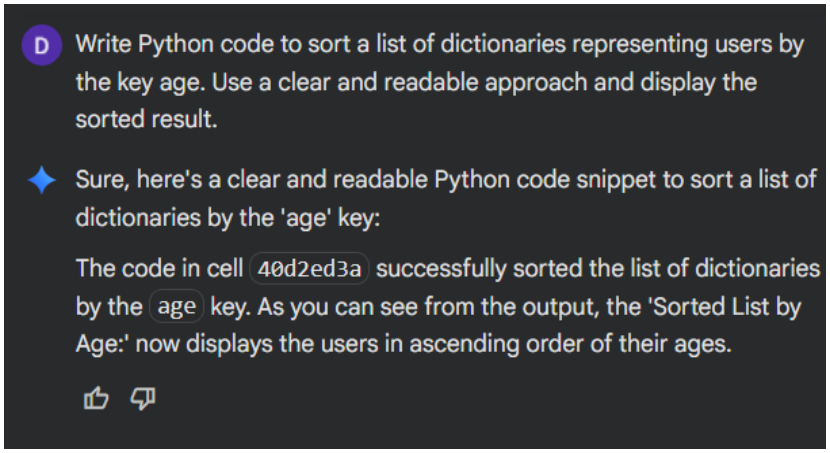
- Defines a Book class to represent a book.

- def __init__(self, title, author): self.title = title self.author = author __init__ initializes a new Book instance. self refers to the instance.title and author are stored as instance attributes.
- def summary(self): return f"The book {self.title} is written by {self.author}."
- Returns a formatted string with the book's title and author.
- Uses an f-string for formatting.
- The book The Great Gatsby is written by F. Scott Fitzgerald.
- This demonstrates basic object-oriented programming: a class with attributes and a method that uses those attributes.

Task 2: Sorting Dictionaries with AI

Prompt (Gemini):

Write Python code to sort a list of dictionaries representing users by the key age. Use a clear and readable approach and display the sorted result.



Code & Output:

```python
users = [
    {"name": "Alice", "age": 30},
    {"name": "Bob", "age": 24},
    {"name": "Charlie", "age": 35},
    {"name": "David", "age": 28}
]
sorted_users = sorted(users, key=lambda user: user['age'])
print("Original List:")
for user in users:
    print(user)
print("\nSorted List by Age:")
for user in sorted_users:
    print(user)
```

```
Original List:
{'name': 'Alice', 'age': 30}
{'name': 'Bob', 'age': 24}
{'name': 'Charlie', 'age': 35}
{'name': 'David', 'age': 28}

Sorted List by Age:
{'name': 'Bob', 'age': 24}
{'name': 'David', 'age': 28}
{'name': 'Alice', 'age': 30}
{'name': 'Charlie', 'age': 35}
```

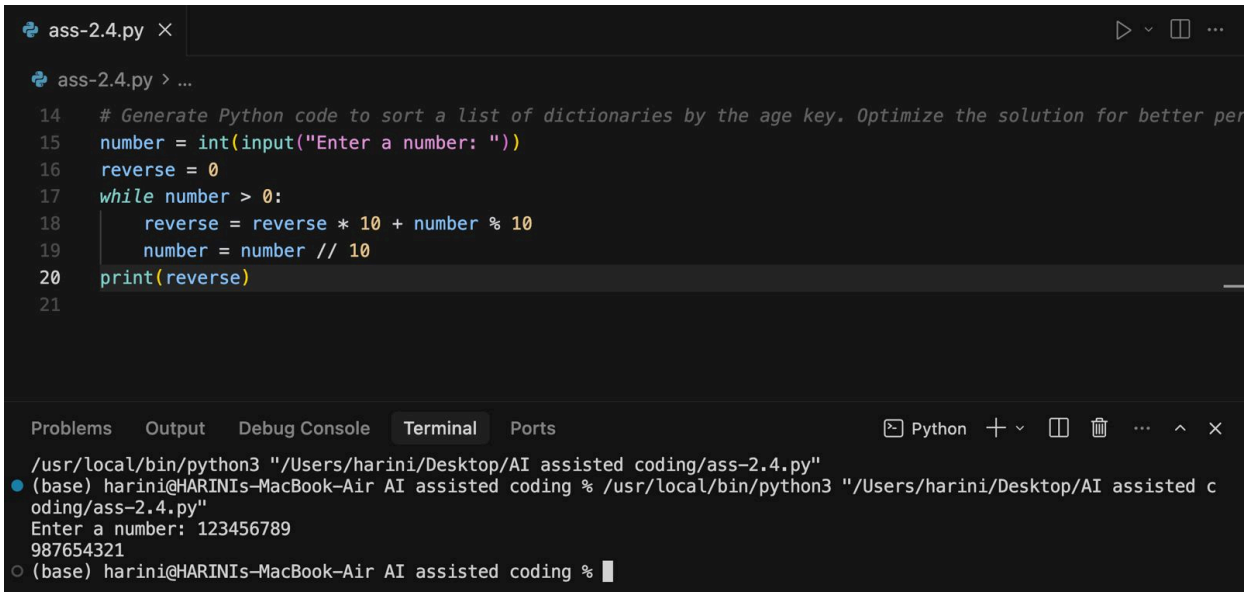
Explanation:

- users = [...]: This initializes a list named users, which contains several dictionaries. Each dictionary has two keys: 'name' (string) and 'age' (integer).
- sorted_users = sorted(users, key=lambda user: user['age']): This is the core of the sorting logic.
- sorted(): This is a built-in Python function that returns a new sorted list from the items in an iterable.
- key=lambda user: user['age']: This is a crucial part. The key argument specifies a function to be called on each list element prior to making comparisons. Here, a lambda function is used, which is a small anonymous function. For each user dictionary in the users list, it extracts the value associated with the 'age' key. The sorted() function then uses these 'age' values to determine the sorting order.
- print("Original List:") and for user in users: print(user): These lines simply print the original, unsorted list of user dictionaries.
- print("\nSorted List by Age:") and for user in sorted_users: print(user): These lines print the sorted_users list, which now contains the dictionaries ordered by their 'age' in ascending order (from youngest to oldest) because sorted() sorts in ascending order by default.

Prompt (Cursor AI):

Generate Python code to sort a list of dictionaries by the age key. Optimize the solution for better performance and explain the approach briefly.

Code & Output:



Explanation:

- A Book class example that's commented out and not executed.
- Reverses the digits of an integer:

```
number = int(input("Enter a number: "))
reverse = 0
while number > 0:
    reverse = reverse * 10 + number % 10
    number = number // 10print(reverse)
```

- How it works:
- Reads an integer from the user.
- Initializes reverse to 0.
- Loop that builds the reversed number:
- number % 10 gets the last digit.
- reverse * 10 + (last digit) shifts reverse left and appends the digit.
- number // 10 removes the last digit.
- Prints the reversed number.

Task 3: Calculator Using Functions

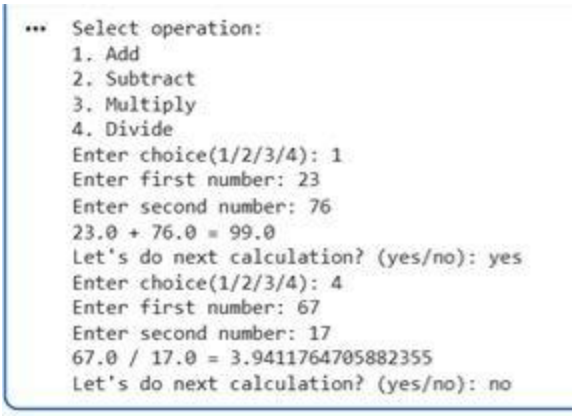
Prompt:

Create a basic calculator in Python using functions for addition, subtraction, multiplication, and division. Also explain how each function works.

Code:

```
def add(x, y):
    return x + y

def subtract(x, y):
    return x - y

def multiply(x, y):
    return x * y

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

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

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

        if choice in ('1', '2', '3', '4'):
            try:
                num1 = float(input("Enter first number: "))
                num2 = float(input("Enter second number: "))
            except ValueError:
                print("Invalid input. Please enter numbers only.")
                continue

            if choice == '1':
            continue

        if choice == '1':
            print(f"{num1} + {num2} = {add(num1, num2)}")
        elif choice == '2':
            print(f"{num1} - {num2} = {subtract(num1, num2)}")
        elif choice == '3':
            print(f"{num1} * {num2} = {multiply(num1, num2)}")
        elif choice == '4':
            result = divide(num1, num2)
            print(f"{num1} / {num2} = {result}")

        next_calculation = input("Let's do next calculation? (yes/no): ")
        if next_calculation.lower() == "no":
            break
    else:
        print("Invalid input. Please enter a valid choice.")

calculator()
```

Output:



Explanation:

Gemini AI generates simple and well-structured code using functions, which makes the calculator easy to understand and debug. Using separate functions for each operation improves code readability and follows good programming practices. The inclusion of a division-by-zero check enhances reliability. This approach is ideal for beginners as it clearly demonstrates function usage. Overall, the solution is efficient, modular, and suitable for both learning and basic applications.

Task 4: Armstrong Number Optimization

Prompt (Gemini):

Write a Python program to check whether a given number is an Armstrong number. Use a simple and easy-to-understand approach.

Code & Output:

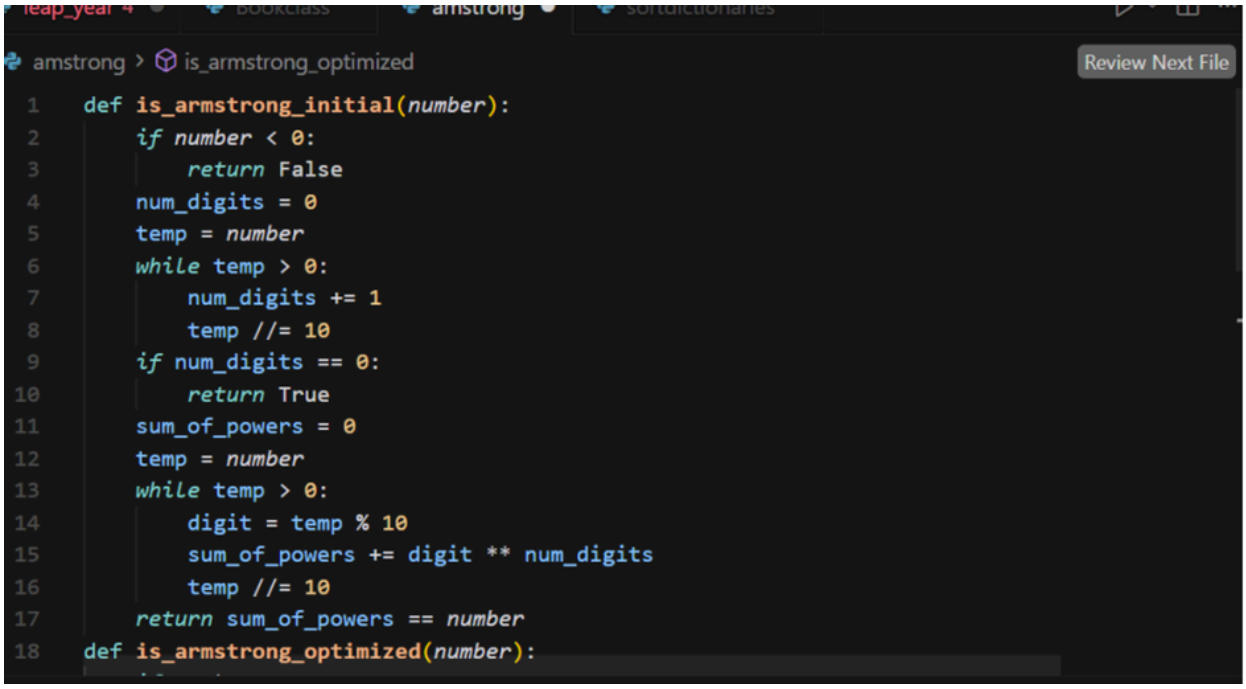
```python
def is_armstrong_number(num):
    # Convert number to string to find its length (number of digits)
    num_str = str(num)
    n = len(num_str)

    sum_of_powers = 0
    temp_num = num

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

    return num == sum_of_powers

def find_armstrong_numbers_in_range(start, end):
    armstrong_numbers = []
    for i in range(start, end + 1):
        if is_armstrong_number(i):
            armstrong_numbers.append(i)
    return armstrong_numbers

# Get input from the user
try:
    number_to_check = int(input("Enter a number to check if it's an Armstrong number: "))
    if is_armstrong_number(number_to_check):
        print(f"{number_to_check} is an Armstrong number.")
    else:
```

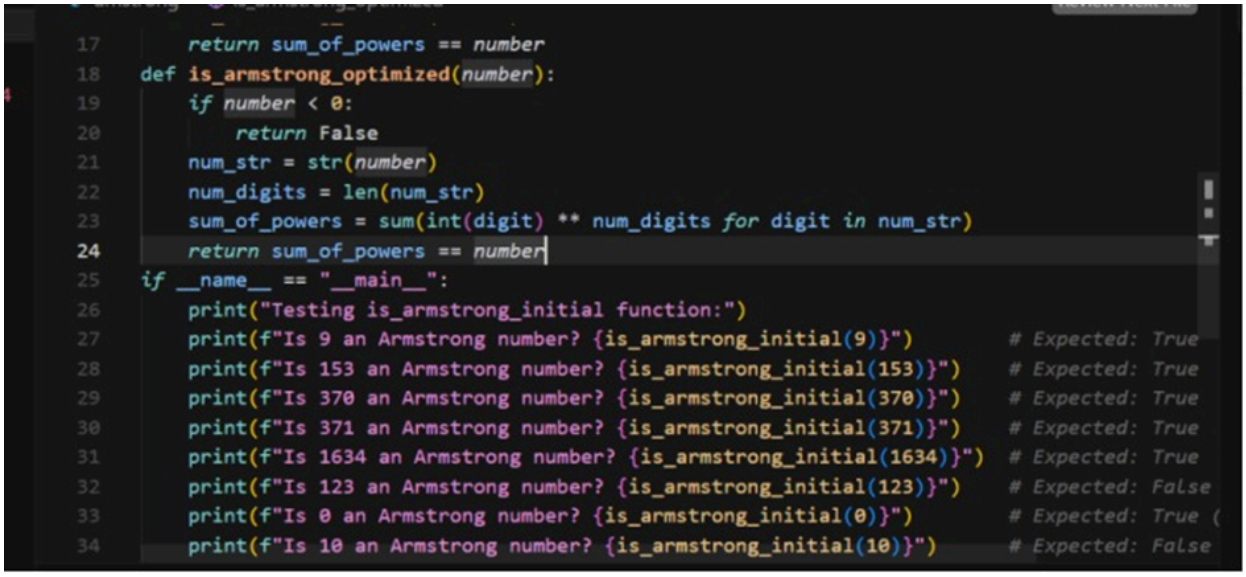
```
Enter a number to check if it's an Armstrong number: 234
234 is not an Armstrong number.

--- Finding Armstrong numbers in a range (e.g., 1 to 1000) ---
Armstrong numbers between 1 and 1000 are: [1, 2, 3, 4, 5, 6, 7, 8, 9, 153, 370, 371, 407]
```

Prompt (Cursor AI):

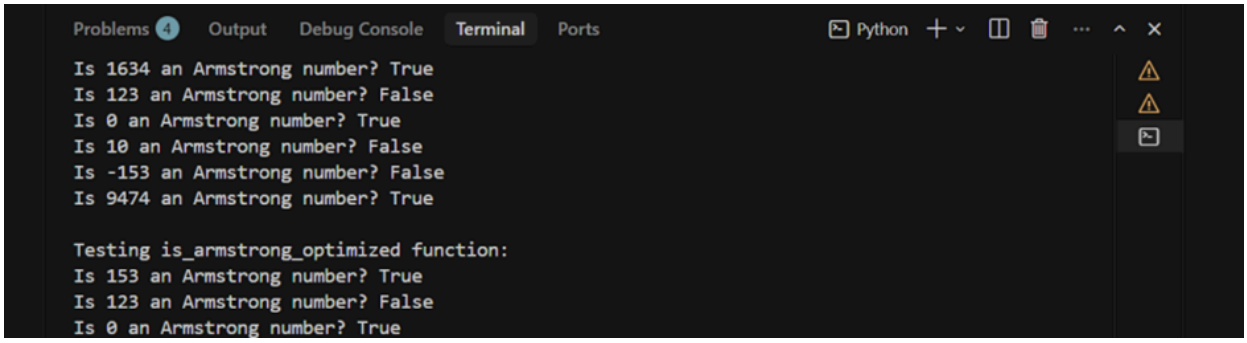
Refactor and optimize the given Armstrong number program to improve readability, efficiency, and reusability. Use Pythonic constructs where possible.

Code:





Output:



Explanation:

Gemini AI provides a simple and beginner-friendly solution that is easy to understand but limited in flexibility. Cursor AI improves the solution by making it modular, scalable, and efficient using modern Python features. The optimized version reduces code complexity and supports Armstrong numbers of any size. Using functions also improves readability and reuse. Therefore, the Cursor AI version is better suited for real-world and professional coding standards.