

# SR UNIVERSITY

## AI ASSIST CODING

### Lab-2.4

ROLL NO:2503A51L17

NAME: Simra Tahseen

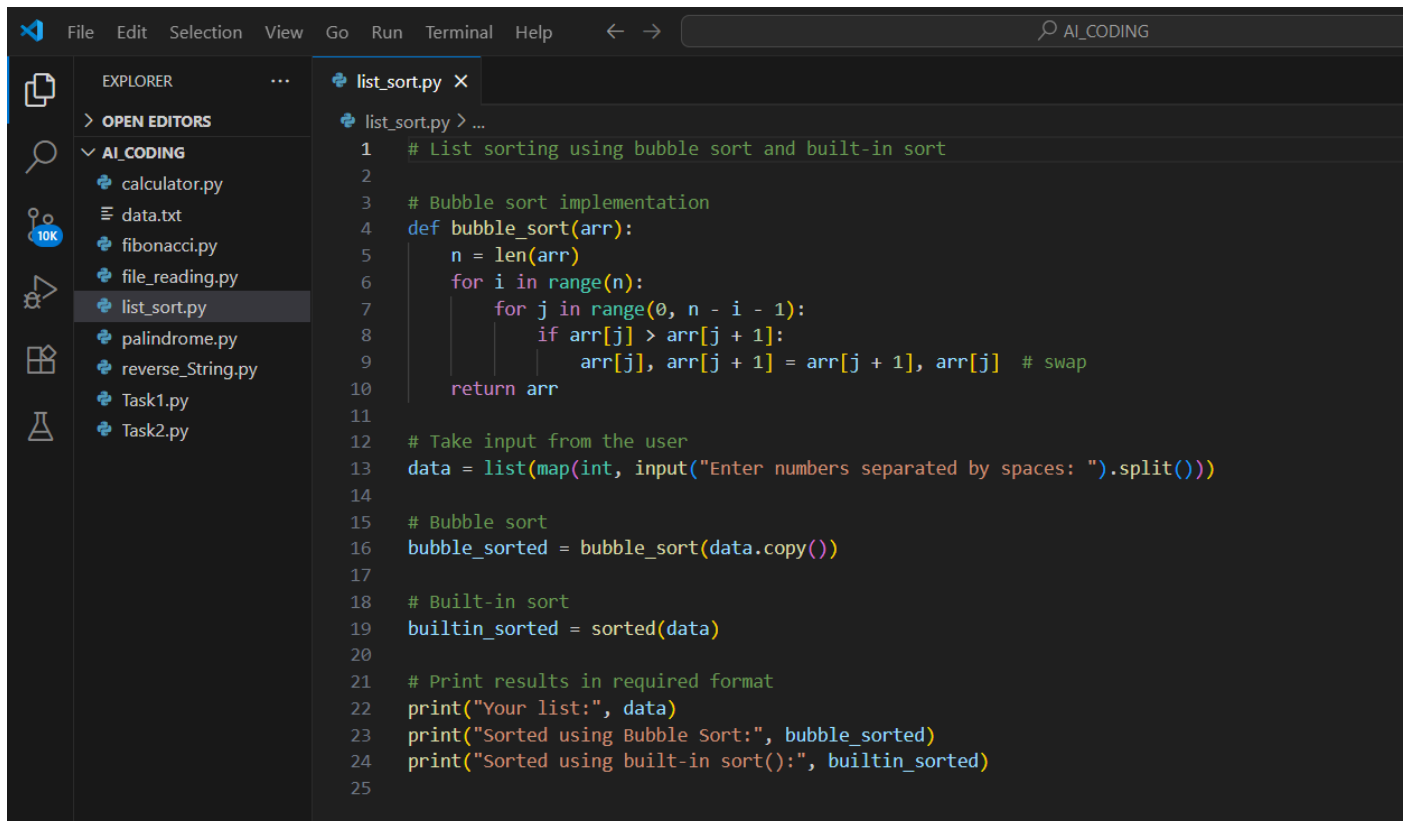
BATCH:19

### TASK #1:

### Prompt Used:

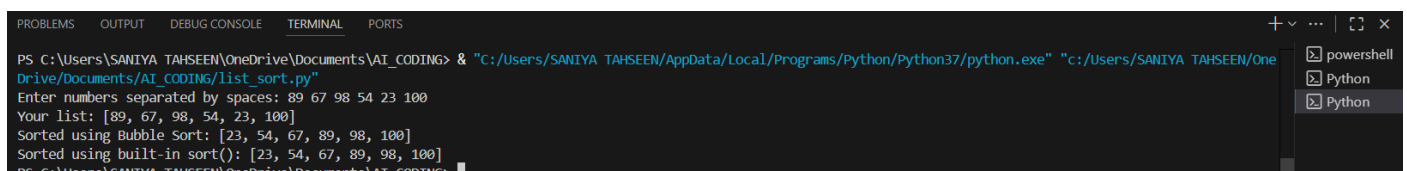
Open Google Colab and use Google Gemini to generate Python code that performs sorting of a list using both the bubble sort algorithm and Python's built-in sort () function. Compare the two implementations.

### Code Generated:



```
1 # List sorting using bubble sort and built-in sort
2
3 # Bubble sort implementation
4 def bubble_sort(arr):
5     n = len(arr)
6     for i in range(n):
7         for j in range(0, n - i - 1):
8             if arr[j] > arr[j + 1]:
9                 arr[j], arr[j + 1] = arr[j + 1], arr[j] # swap
10    return arr
11
12 # Take input from the user
13 data = list(map(int, input("Enter numbers separated by spaces: ").split()))
14
15 # Bubble sort
16 bubble_sorted = bubble_sort(data.copy())
17
18 # Built-in sort
19 builtin_sorted = sorted(data)
20
21 # Print results in required format
22 print("Your list:", data)
23 print("Sorted using Bubble Sort:", bubble_sorted)
24 print("Sorted using built-in sort():", builtin_sorted)
25
```

### Output After executing Code:



```
PS C:\Users\SANIYA TAHSEEN\OneDrive\Documents\AI_CODING> & "c:/Users/SANIYA TAHSEEN/AppData/Local/Programs/Python/Python37/python.exe" "c:/Users/SANIYA TAHSEEN/OneDrive/Documents/AI_CODING/list_sort.py"
Enter numbers separated by spaces: 89 67 98 54 23 100
Your list: [89, 67, 98, 54, 23, 100]
Sorted using Bubble Sort: [23, 54, 67, 89, 98, 100]
Sorted using built-in sort(): [23, 54, 67, 89, 98, 100]
PS C:\Users\SANIYA TAHSEEN\OneDrive\Documents\AI_CODING>
```

### Observations:

- The program sorts a user-provided list using both Bubble Sort and Python's built-in sort() for comparison.

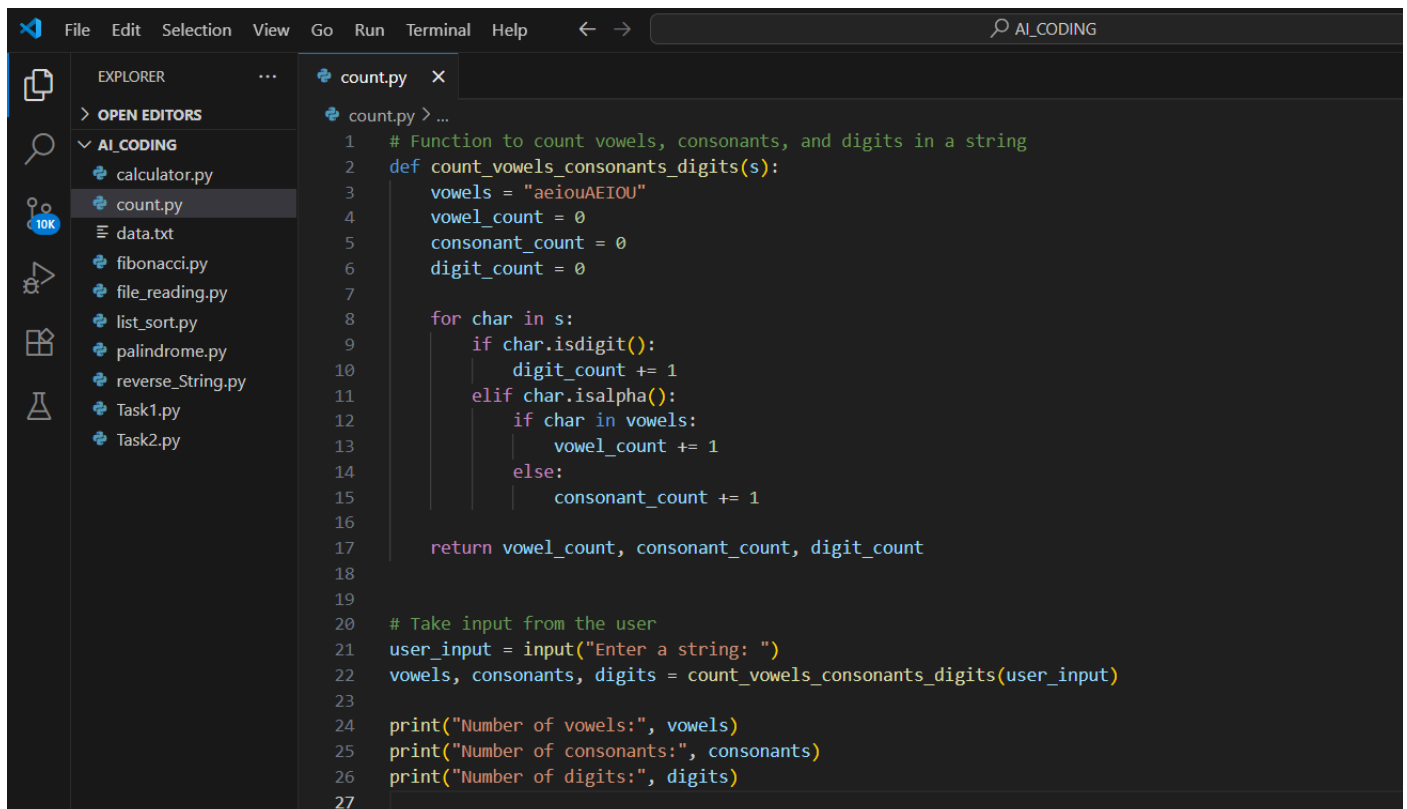
- Bubble Sort works by repeatedly swapping adjacent elements, making it easy to understand but inefficient for large lists.
- Python's built-in sort() is highly optimized and much faster than Bubble Sort.
- Both methods yield the same sorted result, but Python's built-in sort is significantly more efficient.

## **TASK #2:**

### **Prompt Used:**

In Colab, use Google Gemini to generate a Python function that takes a string and returns: The number of vowels, The number of consonants, The number of digits in the string.

### **Code Generated:**

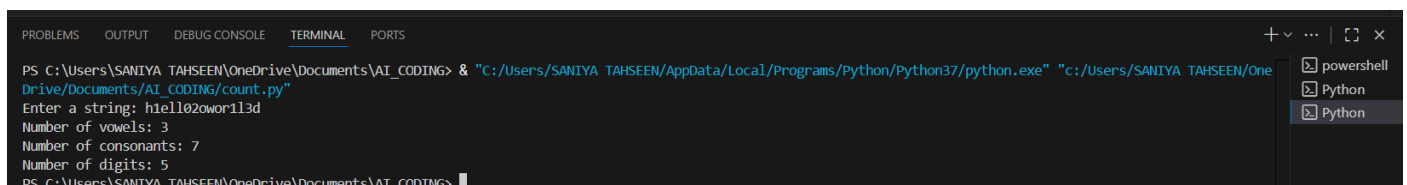


```

1  # Function to count vowels, consonants, and digits in a string
2  def count_vowels_consonants_digits(s):
3      vowels = "aeiouAEIOU"
4      vowel_count = 0
5      consonant_count = 0
6      digit_count = 0
7
8      for char in s:
9          if char.isdigit():
10             digit_count += 1
11          elif char.isalpha():
12              if char in vowels:
13                  vowel_count += 1
14              else:
15                  consonant_count += 1
16
17      return vowel_count, consonant_count, digit_count
18
19
20 # Take input from the user
21 user_input = input("Enter a string: ")
22 vowels, consonants, digits = count_vowels_consonants_digits(user_input)
23
24 print("Number of vowels:", vowels)
25 print("Number of consonants:", consonants)
26 print("Number of digits:", digits)
27

```

### **Output After executing Code:**



```

PS C:\Users\SANIYA TAHSEEN\OneDrive\Documents\AI_CODING> & "C:/Users/SANIYA TAHSEEN/AppData/Local/Programs/Python/Python37/python.exe" "C:/Users/SANIYA TAHSEEN/OneDrive/Documents/AI_CODING/count.py"
Enter a string: hell0zowor1l3d
Number of vowels: 3
Number of consonants: 7
Number of digits: 5
PS C:\Users\SANIYA TAHSEEN\OneDrive\Documents\AI_CODING>

```

### **Observations:**

- The function counts vowels, consonants, and digits in a string provided by the user.
- Each character is checked: vowels are matched using a predefined set, if it is an alphabetic character but not a vowel, it is classified as a consonant, digits are detected with isdigit().
- The function uses .lower() to handle both uppercase and lowercase letters consistently.

- The results are returned as a tuple and unpacked into separate variables, making the output clear and structured.

### **TASK #3:**

#### **Prompt Used:**

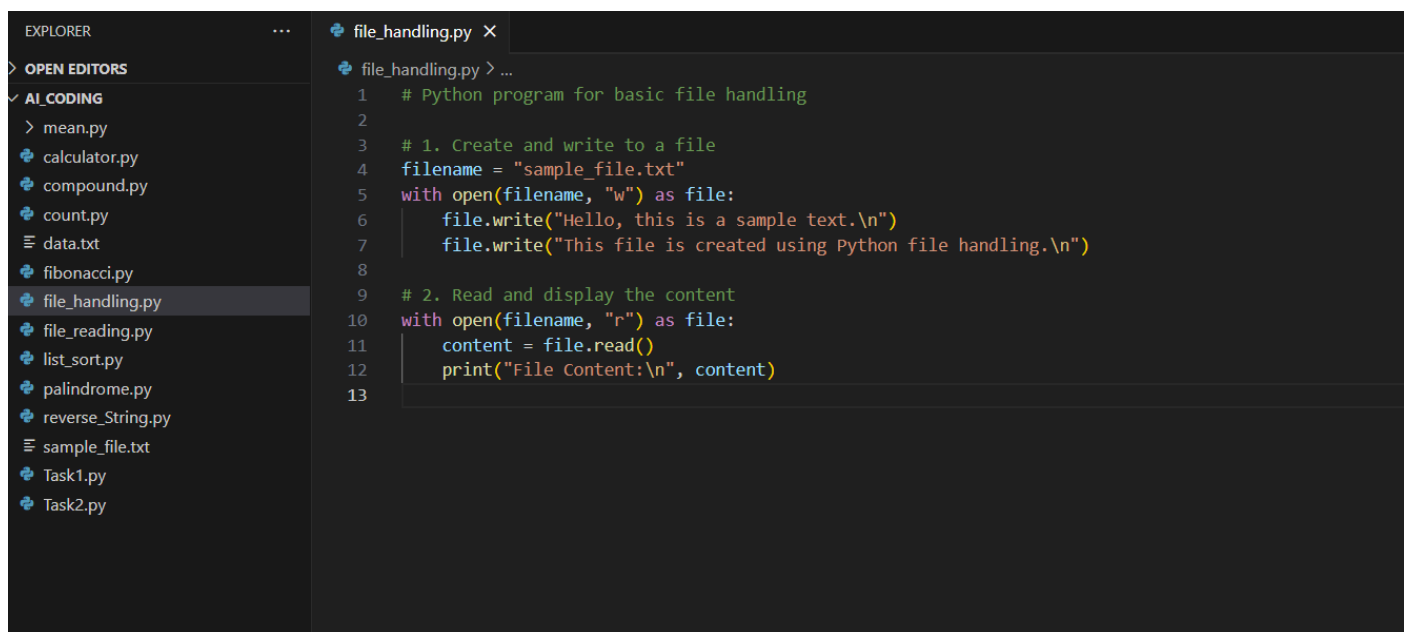
Install and set up Cursor AI. Use it to generate a Python program that performs file handling:

Create a text file

Write sample text

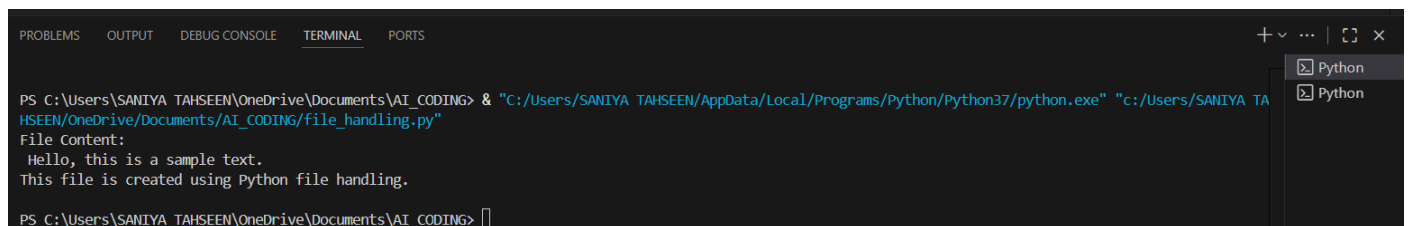
Read and display the content

#### **Code Generated:**



```
1 # Python program for basic file handling
2
3 # 1. Create and write to a file
4 filename = "sample_file.txt"
5 with open(filename, "w") as file:
6     file.write("Hello, this is a sample text.\n")
7     file.write("This file is created using Python file handling.\n")
8
9 # 2. Read and display the content
10 with open(filename, "r") as file:
11     content = file.read()
12     print("File Content:\n", content)
13
```

#### **Output After executing Code:**



```
PS C:\Users\SANIYA TAHSEEN\OneDrive\Documents\AI_CODING> & "C:/Users/SANIYA TAHSEEN/AppData/Local/Programs/Python/Python37/python.exe" "c:/Users/SANIYA TAHSEEN/OneDrive/Documents/AI_CODING/file_handling.py"
File Content:
Hello, this is a sample text.
This file is created using Python file handling.
PS C:\Users\SANIYA TAHSEEN\OneDrive\Documents\AI_CODING>
```

#### **Observations:**

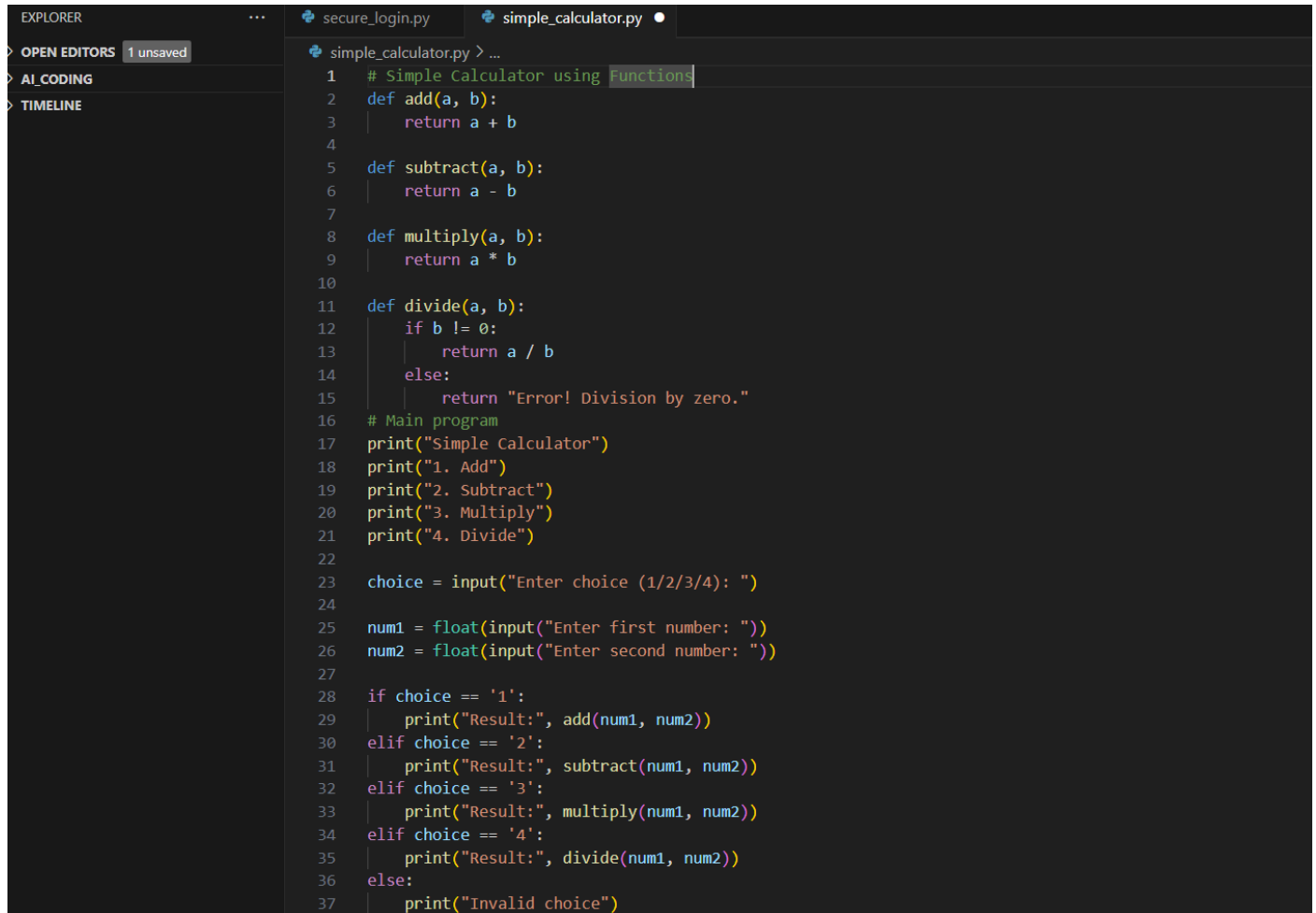
- A file named sample\_file.txt is created using write ("w") mode, which overwrites the file if it already exists.
- The with open() statement is used for automatic handling of closing the file.
- Two lines of text are written into the file using write().
- The file is then opened in read ("r") mode to fetch its content.
- read() reads the entire file content at once and prints it on the console.

## TASK #4:

### Prompt Used:

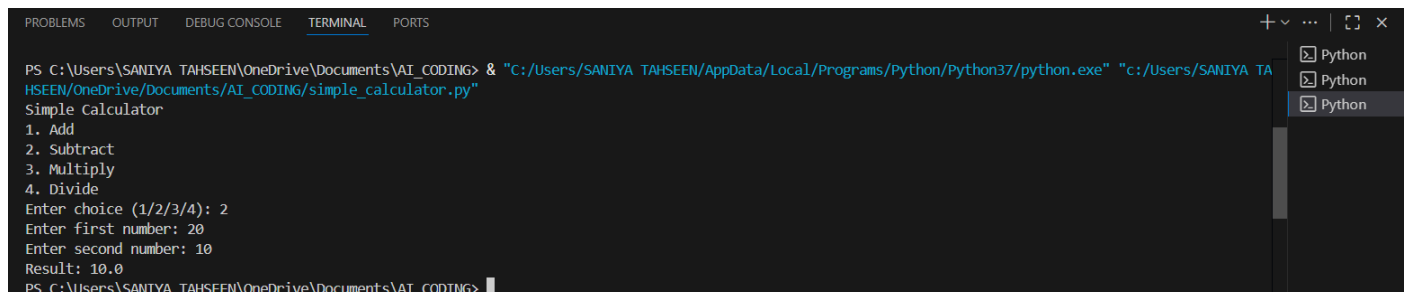
- Ask Google Gemini to generate a Python program that implements a simple calculator using functions (add, subtract, multiply, divide). Then, ask Gemini to explain how the code works.

### Code Generated:



```
1 # Simple Calculator using Functions
2 def add(a, b):
3     return a + b
4
5 def subtract(a, b):
6     return a - b
7
8 def multiply(a, b):
9     return a * b
10
11 def divide(a, b):
12     if b != 0:
13         return a / b
14     else:
15         return "Error! Division by zero."
16
17 # Main program
18 print("Simple Calculator")
19 print("1. Add")
20 print("2. Subtract")
21 print("3. Multiply")
22 print("4. Divide")
23
24 choice = input("Enter choice (1/2/3/4): ")
25
26 num1 = float(input("Enter first number: "))
27 num2 = float(input("Enter second number: "))
28
29 if choice == '1':
30     print("Result:", add(num1, num2))
31 elif choice == '2':
32     print("Result:", subtract(num1, num2))
33 elif choice == '3':
34     print("Result:", multiply(num1, num2))
35 elif choice == '4':
36     print("Result:", divide(num1, num2))
37 else:
38     print("Invalid choice")
```

### Output After executing Code:



```
PS C:\Users\SANIYA TAHSEEN\OneDrive\Documents\AI_CODING> & "c:/Users/SANIYA TAHSEEN/AppData/Local/Programs/Python/Python37/python.exe" "c:/Users/SANIYA TAHSEEN/OneDrive/Documents/AI_CODING/simple_calculator.py"
Simple Calculator
1. Add
2. Subtract
3. Multiply
4. Divide
Enter choice (1/2/3/4): 2
Enter first number: 20
Enter second number: 10
Result: 10.0
PS C:\Users\SANIYA TAHSEEN\OneDrive\Documents\AI_CODING>
```

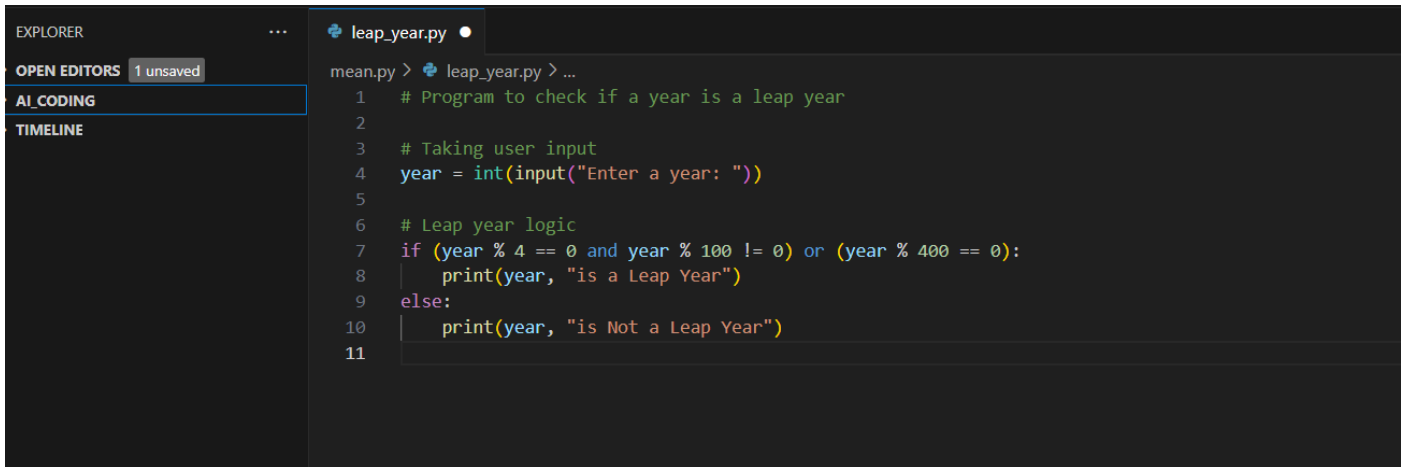
## TASK #5:

### Prompt Used:

Use Cursor AI to create a Python program that checks if a given year is a leap year or not. Try different prompt styles and see how Cursor modifies its code suggestions.

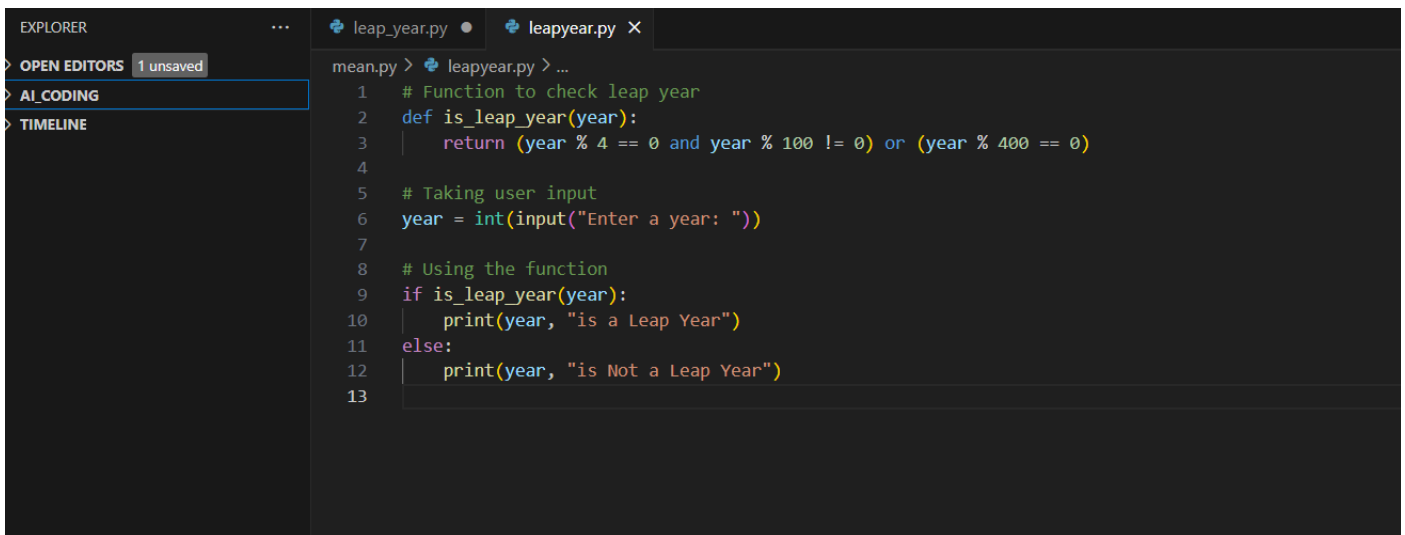
### Code Generated:

#### Prompt-1:



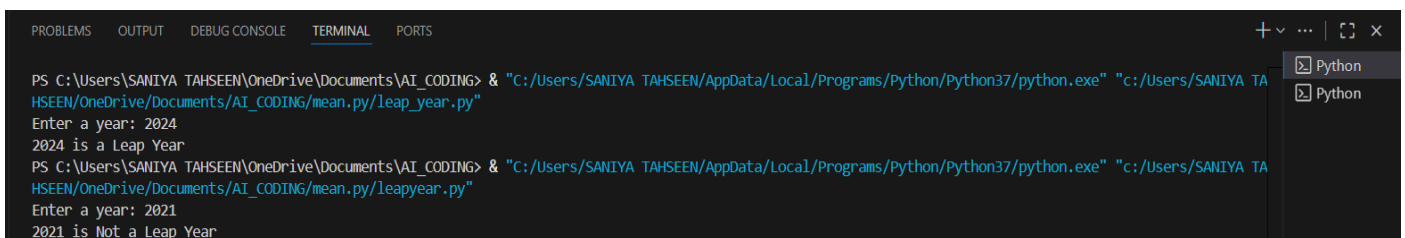
```
mean.py > leap_year.py > ...
1  # Program to check if a year is a leap year
2
3  # Taking user input
4  year = int(input("Enter a year: "))
5
6  # Leap year logic
7  if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
8      print(year, "is a Leap Year")
9  else:
10     print(year, "is Not a Leap Year")
11
```

#### Prompt-2:



```
mean.py > leapyear.py > ...
1  # Function to check leap year
2  def is_leap_year(year):
3      return (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0)
4
5  # Taking user input
6  year = int(input("Enter a year: "))
7
8  # Using the function
9  if is_leap_year(year):
10     print(year, "is a Leap Year")
11 else:
12     print(year, "is Not a Leap Year")
13
```

### Output After executing Code:



```
PS C:\Users\SANIYA TAHSEEN\OneDrive\Documents\AI_CODING> & "C:/Users/SANIYA TAHSEEN/AppData/Local/Programs/Python/Python37/python.exe" "c:/Users/SANIYA TAHSEEN/OneDrive/Documents/AI_CODING/mean.py/leap_year.py"
Enter a year: 2024
2024 is a Leap Year
PS C:\Users\SANIYA TAHSEEN\OneDrive\Documents\AI_CODING> & "C:/Users/SANIYA TAHSEEN/AppData/Local/Programs/Python/Python37/python.exe" "c:/Users/SANIYA TAHSEEN/OneDrive/Documents/AI_CODING/mean.py/leapyear.py"
Enter a year: 2021
2021 is Not a Leap Year
```

## **Observations:**

Prompt-1:

- Uses **inline if-else logic** to check leap year condition:  
(year % 4 == 0 and year % 100 != 0) or (year % 400 == 0)
- Drawback: The logic and input/output are mixed together, making it less reusable.

Prompt-2:

- The program defines a separate function `is_leap_year(year)` and returns True or False Depending on condition.
- Main code only handles input and output, while the logic is isolated inside the function.