

SR UNIVERSITY

AI ASSIST CODING

Lab-2.4

ROLL NO:2503A51L13

NAME: Begala Hasini

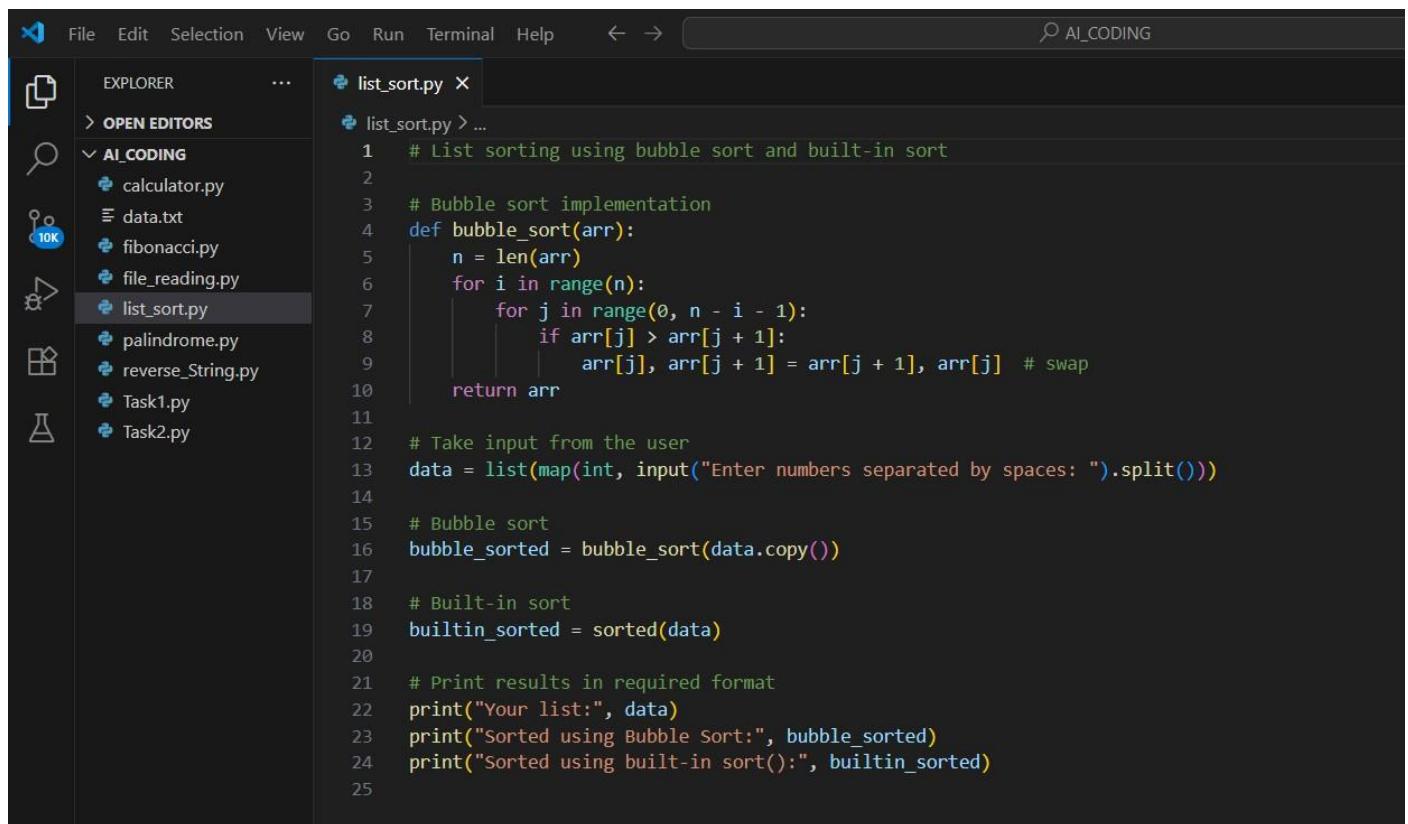
BATCH:19

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:



The screenshot shows a code editor interface with a dark theme. On the left is an Explorer sidebar listing files in a directory named 'AI_CODING'. The file 'list_sort.py' is selected and shown in the main editor area. The code implements two sorting methods: bubble sort and Python's built-in sorted() function, and compares their results.

```
# List sorting using bubble sort and built-in sort
# Bubble sort implementation
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j] # swap
    return arr

# Take input from the user
data = list(map(int, input("Enter numbers separated by spaces: ").split()))

# Bubble sort
bubble_sorted = bubble_sort(data.copy())

# Built-in sort
builtin_sorted = sorted(data)

# Print results in required format
print("Your list:", data)
print("Sorted using Bubble Sort:", bubble_sorted)
print("Sorted using built-in sort():", builtin_sorted)
```

Output After executing Code:



The terminal window shows the execution of the 'list_sort.py' script. It prompts for user input, displays the original list, and then shows the sorted lists produced by both the bubble sort implementation and Python's built-in sorted() function.

```
Drive/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]
```

Observations:

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

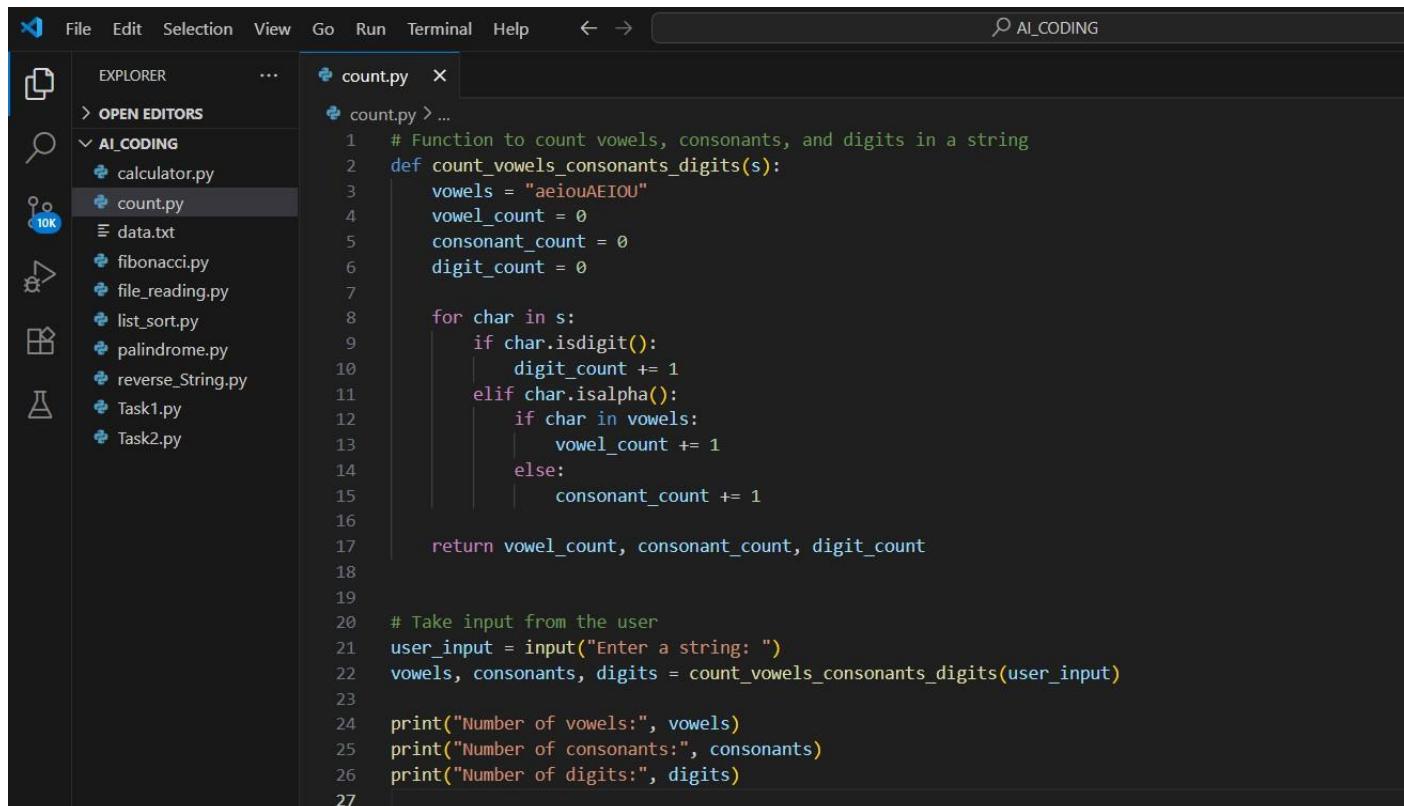
- Bubble Sort works by repeatedly swapping adjacent elements, making it easy to understand but inefficient for large lists.
- Python's built-in `sort()` is highly optimized and much faster than Bubble Sort.
- Both methods yield the same sorted result, but Python's built-in `sort` is significantly more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



```

File Edit Selection View Go Run Terminal Help ⌘ AI_CODING
EXPLORER OPEN EDITORS AI_CODING
calculator.py count.py data.txt fibonacci.py file_reading.py list_sort.py palindrome.py reverse_String.py Task1.py Task2.py
count.py > ...
# Function to count vowels, consonants, and digits in a string
def count_vowels_consonants_digits(s):
    vowels = "aeiouAEIOU"
    vowel_count = 0
    consonant_count = 0
    digit_count = 0

    for char in s:
        if char.isdigit():
            digit_count += 1
        elif char.isalpha():
            if char in vowels:
                vowel_count += 1
            else:
                consonant_count += 1

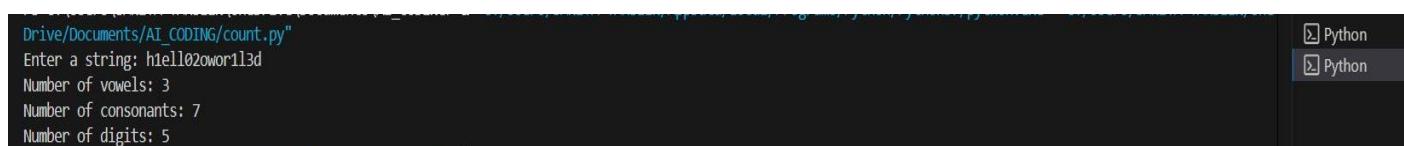
    return vowel_count, consonant_count, digit_count

# Take input from the user
user_input = input("Enter a string: ")
vowels, consonants, digits = count_vowels_consonants_digits(user_input)

print("Number of vowels:", vowels)
print("Number of consonants:", consonants)
print("Number of digits:", digits)

```

Output After executing Code:



```

Drive/Documents/AI_CODING/count.py
Enter a string: helloworld
Number of vowels: 3
Number of consonants: 7
Number of digits: 5

```

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:

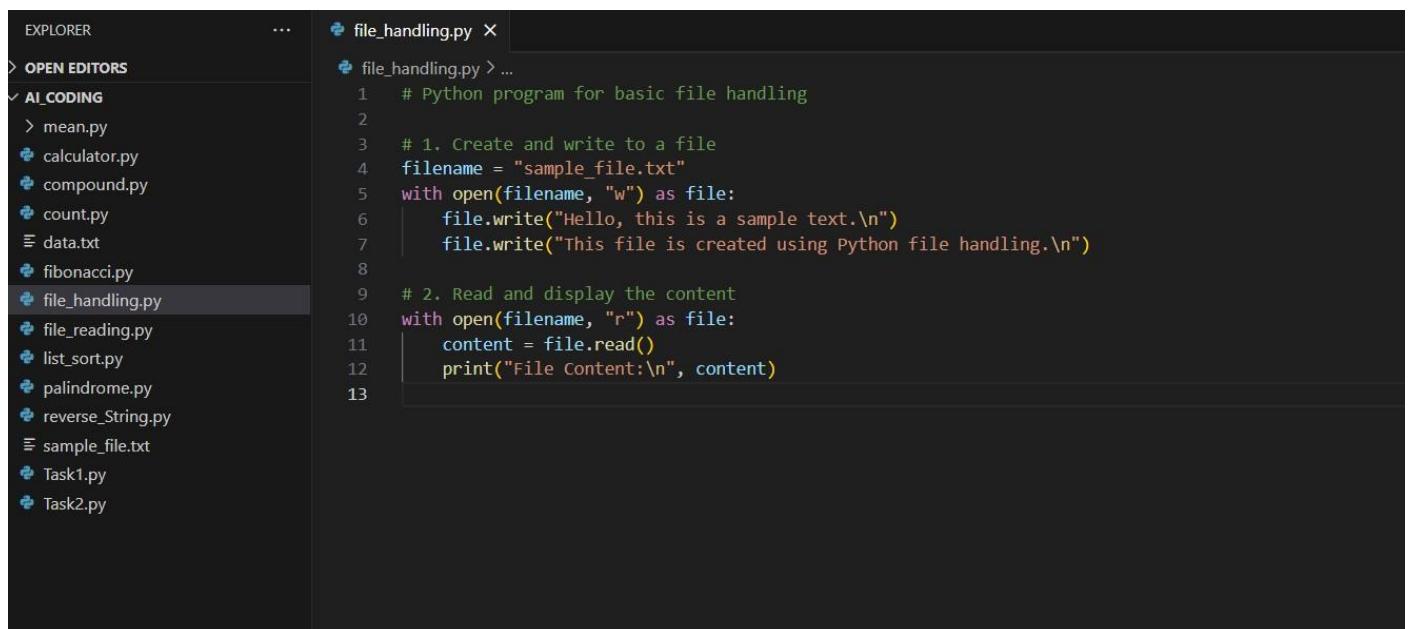
Install and set up Cursor AI. Use it to generate a Python program that performs file handling:

Create a text file

Write sample text

Read and display the content

Code Generated:



```

EXPLORER ... file_handling.py x
OPEN EDITORS ...
AI CODING ...
mean.py
calculator.py
compound.py
count.py
data.txt
fibonacci.py
file_handling.py
file_reading.py
list_sort.py
palindrome.py
reverse_String.py
sample_file.txt
Task1.py
Task2.py

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

```

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:

The screenshot shows the Visual Studio Code interface with the following details:

- EXPLORER**: Shows "OPEN EDITORS" (1 unsaved), "AI CODING", and "TIMELINE".
- simple_calculator.py** is the active editor, containing the following Python code:

```
# Simple Calculator using Functions
def add(a, b):
    return a + b

def subtract(a, b):
    return a - b

def multiply(a, b):
    return a * b

def divide(a, b):
    if b != 0:
        return a / b
    else:
        return "Error! Division by zero."
# Main program
print("Simple Calculator")
print("1. Add")
print("2. Subtract")
print("3. Multiply")
print("4. Divide")
choice = input("Enter choice (1/2/3/4): ")
num1 = float(input("Enter first number: "))
num2 = float(input("Enter second number: "))
if choice == '1':
    print("Result:", add(num1, num2))
elif choice == '2':
    print("Result:", subtract(num1, num2))
elif choice == '3':
    print("Result:", multiply(num1, num2))
elif choice == '4':
    print("Result:", divide(num1, num2))
else:
    print("Invalid choice")
```

Output After executing Code:

The screenshot shows a terminal window with the following output:

```
HSEEN/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
```

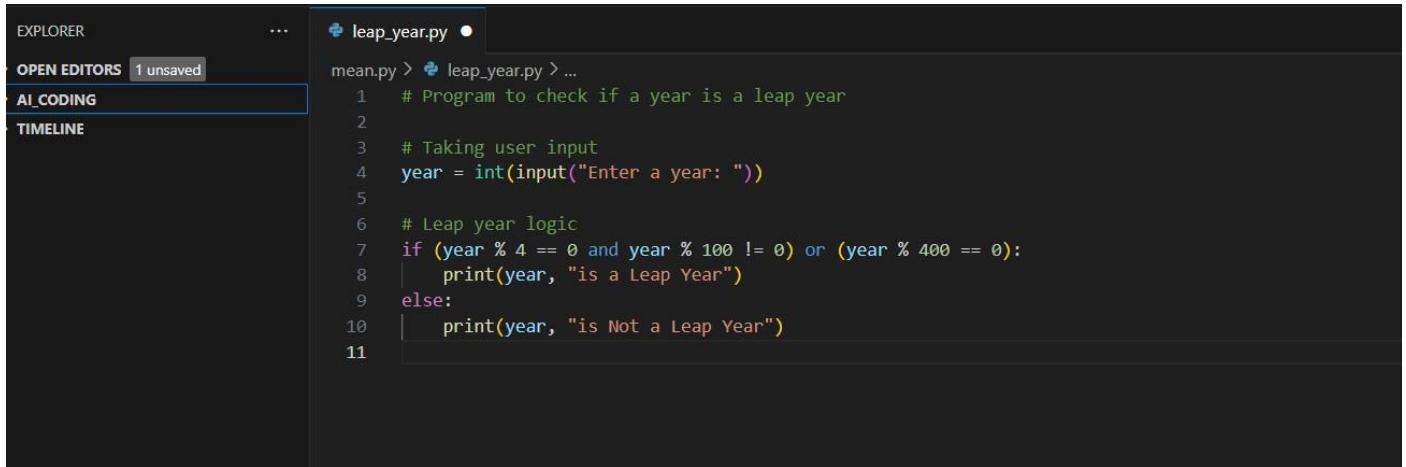
TASK #5:

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.

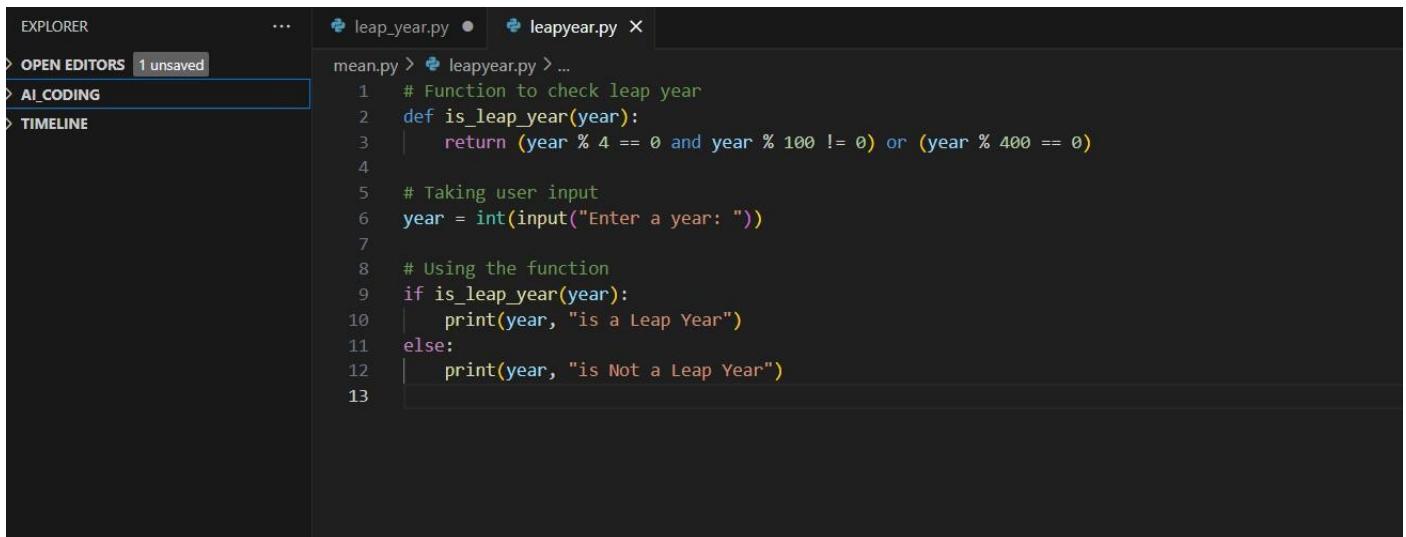
Code Generated:

Prompt-1:



```
EXPLORER ...  
OPEN EDITORS 1 unsaved  
AI_CODING  
TIMELINE  
  
leap_year.py ●  
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:



```
EXPLORER ...  
OPEN EDITORS 1 unsaved  
AI_CODING  
TIMELINE  
  
leap_year.py ● leapyear.py ✘ ...  
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\SANITA TASEEN\OneDrive\Documents\AI_CODING> C:/Users/SANITA TASEEN/AppData/Local/Programs/Python/Python37/python.exe C:/Users/SANITA TA  
SEEN/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.