

ASSIGNMENT -4.3

2303A51343

B-10

TASK-1

Prompt: generate a python code to check whether a given year is a leap year or not with user input

without input and output examples

code:

```
def is_leap_year(year):
```

```
    if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
```

```
        return True
```

```
    else:
```

```
        return False
```

```
# Example usage
```

```
year = int(input("Enter a year to check if it's a leap year: "))
```

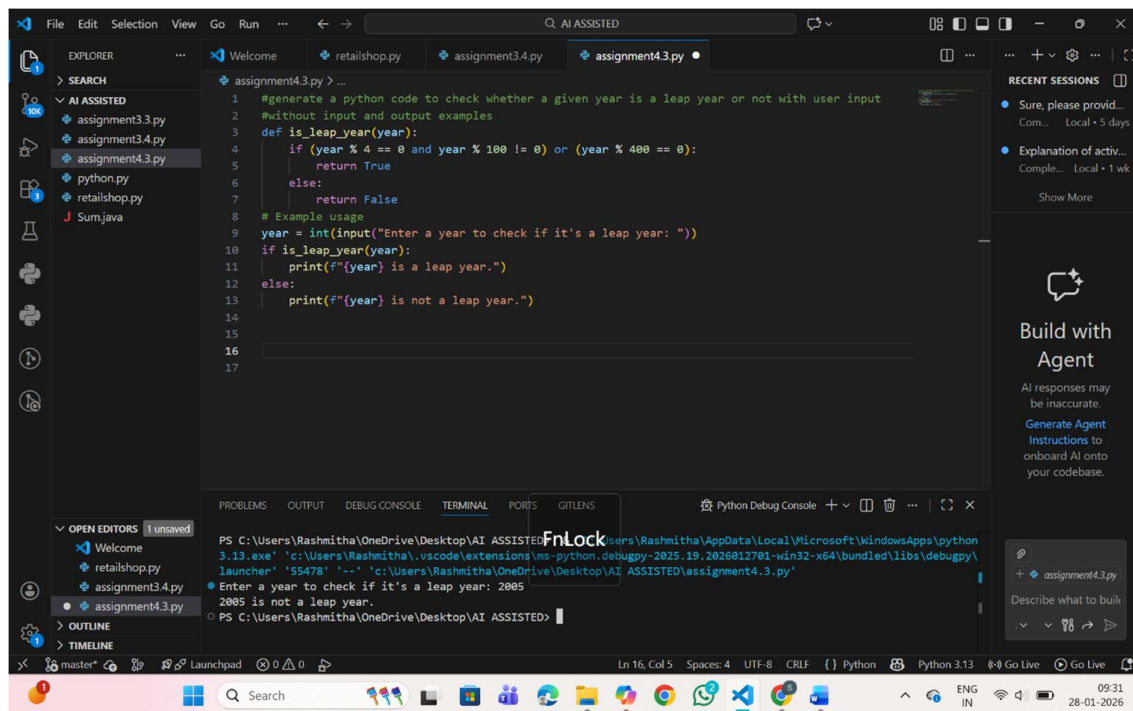
```
if is_leap_year(year):
```

```
    print(f"{year} is a leap year.")
```

```
else:
```

```
    print(f"{year} is not a leap year.")
```

output:



Analysis: The code defines a function that applies the correct leap year rules, accepts user input, and displays whether the year is a leap year. It is straightforward, easy to understand, and logically sound.

TASK-2

Prompt:

#generate a Python code that:

Converts centimeters to inches using the correct formula (1 cm = 0.3937 inches)

#Example: Input: 10 cm → Output: 3.94 inches

Code:

```
def cm_to_inches(cm):
```

```
    inches = cm * 0.3937
```

```
    return round(inches, 2)
```

```
# Example usage
```

```
cm_input = float(input("Enter length in centimeters: "))

inches_output = cm_to_inches(cm_input)

print(f"{cm_input} cm is equal to {inches_output} inches.")

output:
```

```
1 #generate a Python code that:
2 # Converts centimeters to inches using the correct formula (1 cm = 0.3937 inches)
3 #Example: Input: 10 cm → Output: 3.94 inches
4 def cm_to_inches(cm):
5     inches = cm * 0.3937
6     return round(inches, 2)
7 # Example usage
8 cm_input = float(input("Enter length in centimeters: "))
9 inches_output = cm_to_inches(cm_input)
10 print(f"{cm_input} cm is equal to {inches_output} inches.")
11
12
13
14
15
16
```

PS C:\Users\Rashmitha\OneDrive\Desktop\AI ASSISTED> python .\assignment4.3.py

Enter length in centimeters: 1

1.0 cm is equal to 0.39 inches.

PS C:\Users\Rashmitha\OneDrive\Desktop\AI ASSISTED>

Analysis: This program converts centimeters to inches using the formula $1 \text{ cm} = 0.3937 \text{ inches}$.

It defines a function that multiplies the input by 0.3937 and rounds to two decimals.

The user enters a centimeter value, which is passed to the function.

Finally, the result is printed clearly showing both input and output.

TASK-3

Prompt: #generate a Python code that:

Accepts a full name as input

Formats it as "Last, First"

Handles single-word names by returning them unchanged

Demonstrates with sample inputs and outputs

```

Code: def format_name(full_name):

    parts = full_name.split()

    if len(parts) == 1:

        return full_name

    else:

        first_name = parts[0]

        last_name = parts[-1]

        return f"{last_name}, {first_name}"

# Example usage

user_input = input("Enter your full name: ")

formatted_name = format_name(user_input)

print("Formatted Name:", formatted_name)

```

output:

The screenshot shows a Visual Studio Code editor window with a Python file named `assignment4.3.py` open. The code in the editor is as follows:

```

5 # Handles single-word names by returning them unchanged
6 # Demonstrates with sample inputs and outputs
7 def format_name(full_name):
8     parts = full_name.split()
9     if len(parts) == 1:
10         return full_name
11     else:
12         first_name = parts[0]
13         last_name = parts[-1]
14         return f"{last_name}, {first_name}"
15
16 # Example usage
17 user_input = input("Enter your full name: ")
18 formatted_name = format_name(user_input)
19 print("Formatted Name:", formatted_name)

```

The terminal at the bottom shows the execution of the script. The prompt `Enter your full name:` is followed by the input `rashmitha sundaragiri`. The output of the script is `Formatted Name: sundaragiri, rashmitha`.

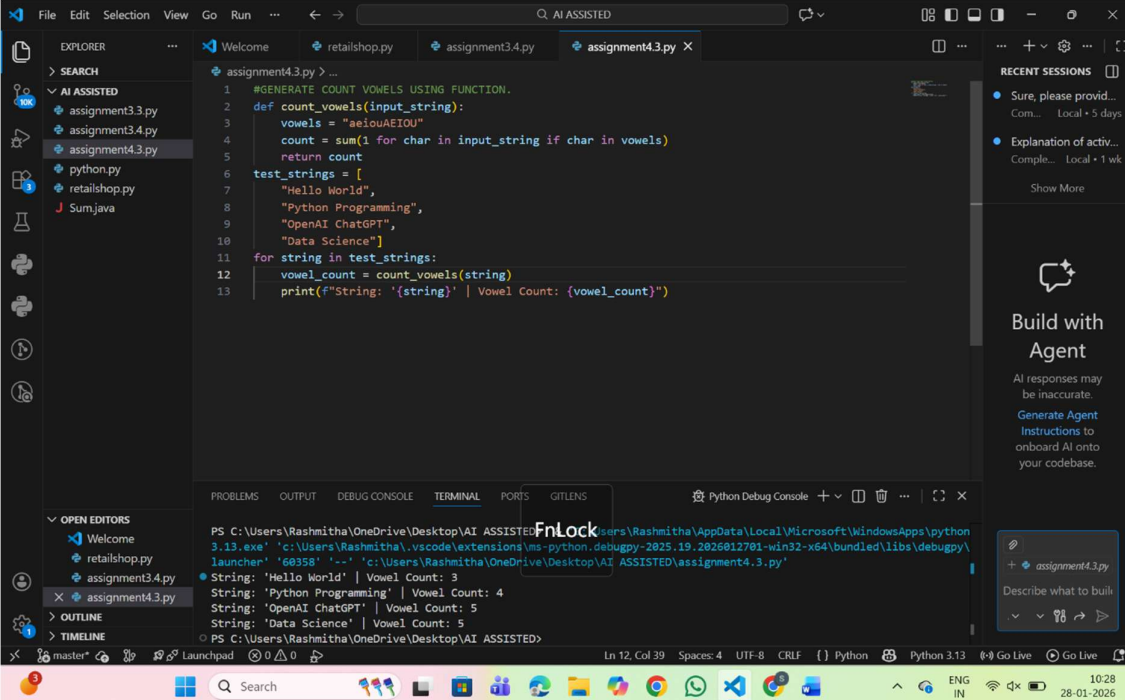
Analysis: The code correctly formats names as “Last, First” and returns single-word names unchanged. It is simple, efficient, and demonstrates usage clearly, though it does not handle middle or compound names. Overall, it fulfills the assignment requirements effectively.

TASK-4:

Prompt: GENERATE COUNT VOWELS USING FUNCTION.

```
Code: def count_vowels(input_string):  
    vowels = "aeiouAEIOU"  
    count = sum(1 for char in input_string if char in vowels)  
    return count  
  
test_strings = [  
    "Hello World",  
    "Python Programming",  
    "OpenAI ChatGPT",  
    "Data Science"]  
  
for string in test_strings:  
    vowel_count = count_vowels(string)  
    print(f"String: '{string}' | Vowel Count: {vowel_count}")
```

output:



The screenshot shows a VS Code editor with a Python file named `assignment4.3.py`. The code defines a function `count_vowels` that takes an input string and returns the count of vowels. It then tests the function with a list of strings: "Hello World", "Python Programming", "OpenAI ChatGPT", and "Data Science". The output in the terminal shows the vowel count for each string: 3, 4, 5, and 5 respectively.

```
1 #GENERATE COUNT VOWELS USING FUNCTION.
2 def count_vowels(input_string):
3     vowels = "aeiouAEIOU"
4     count = sum(1 for char in input_string if char in vowels)
5     return count
6
7 test_strings = [
8     "Hello World",
9     "Python Programming",
10    "OpenAI ChatGPT",
11    "Data Science"]
12
13 for string in test_strings:
14     vowel_count = count_vowels(string)
15     print(f"String: '{string}' | Vowel Count: {vowel_count}")
```

Terminal Output:

```
PS C:\Users\Rashmitha\OneDrive\Desktop\AI ASSISTED> python .\assignment4.3.py
String: 'Hello World' | Vowel Count: 3
String: 'Python Programming' | Vowel Count: 4
String: 'OpenAI ChatGPT' | Vowel Count: 5
String: 'Data Science' | Vowel Count: 5
PS C:\Users\Rashmitha\OneDrive\Desktop\AI ASSISTED>
```

analysis: The function `count_vowels` checks each character in a string and counts if it is a vowel. It uses a generator expression with `sum()` for concise and efficient calculation. The program then tests multiple strings and prints each with its corresponding vowel count.

TASK-5

Prompt: #generate a python program with function that reads the file and

#counts no.of lines in the file and return the line count

Code: `def count_lines_in_file(file_path):`

`try:`

`with open(file_path, 'r') as file:`

`lines = file.readlines()`

`return len(lines)`

`except FileNotFoundError:`

`print(f"Error: The file '{file_path}' was not found.")`

`return None`

Example usage

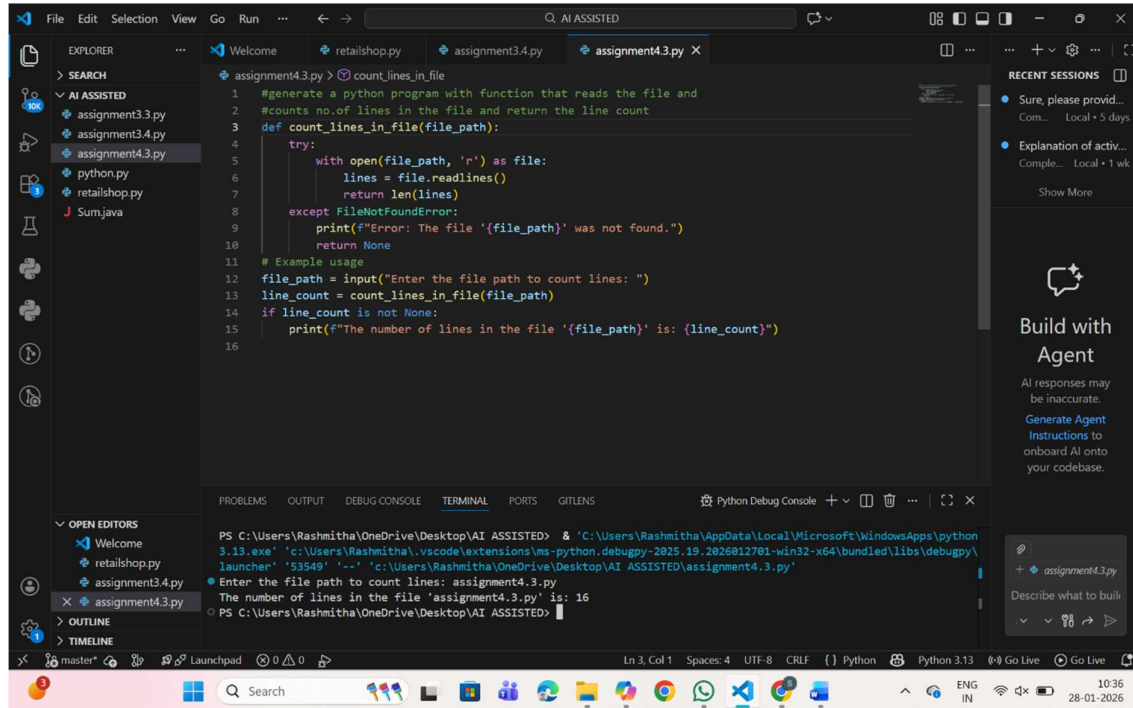
`file_path = input("Enter the file path to count lines: ")`

`line_count = count_lines_in_file(file_path)`

if line_count is not None:

```
print(f"The number of lines in the file '{file_path}' is: {line_count}")
```

output:



The screenshot shows the Visual Studio Code interface. The Explorer panel on the left lists files: 'Welcome', 'retailshop.py', 'assignment3.4.py', and 'assignment4.3.py'. The main editor window displays the code for 'assignment4.3.py'. The code defines a function 'count_lines_in_file' that takes a file path and returns the number of lines. It uses a try-except block to handle 'FileNotFoundError'. The script prompts the user for a file path and prints the line count. The Output panel at the bottom shows the execution results: 'Enter the file path to count lines: assignment4.3.py', 'The number of lines in the file 'assignment4.3.py' is: 16', and the command prompt prompt 'PS C:\Users\Rashmitha\OneDrive\Desktop\AI ASSISTED>'. The status bar at the bottom indicates the file is at line 3, column 1, using UTF-8 encoding with CRLF line endings, and the Python version is 3.13.

```
1 #generate a python program with function that reads the file and
2 #counts no. of lines in the file and return the line count
3 def count_lines_in_file(file_path):
4     try:
5         with open(file_path, 'r') as file:
6             lines = file.readlines()
7             return len(lines)
8     except FileNotFoundError:
9         print(f"Error: The file '{file_path}' was not found.")
10        return None
11
12 # Example usage
13 file_path = input("Enter the file path to count lines: ")
14 line_count = count_lines_in_file(file_path)
15 if line_count is not None:
16     print(f"The number of lines in the file '{file_path}' is: {line_count}")
```

PS C:\Users\Rashmitha\OneDrive\Desktop\AI ASSISTED> & 'C:\Users\Rashmitha\AppData\Local\Microsoft\WindowsApps\python3.13.exe' 'c:\Users\Rashmitha\.vscode\extensions\ms-python.debugpy-2025.19.2026012701-win32-x64\bundled\libs\debugpy\launcher' '53549' '--' 'c:\Users\Rashmitha\OneDrive\Desktop\AI ASSISTED\assignment4.3.py'

Enter the file path to count lines: assignment4.3.py

The number of lines in the file 'assignment4.3.py' is: 16

PS C:\Users\Rashmitha\OneDrive\Desktop\AI ASSISTED>

analysis:

This program defines a function `count_lines_in_file` that reads a file and returns its line count. It uses a try-except block to handle missing files gracefully by showing an error message. The function opens the file in read mode, counts lines with `len()`, and prints the result after the user enters a file path. Overall, it is a simple and reliable solution.