

# Assignment-4.3

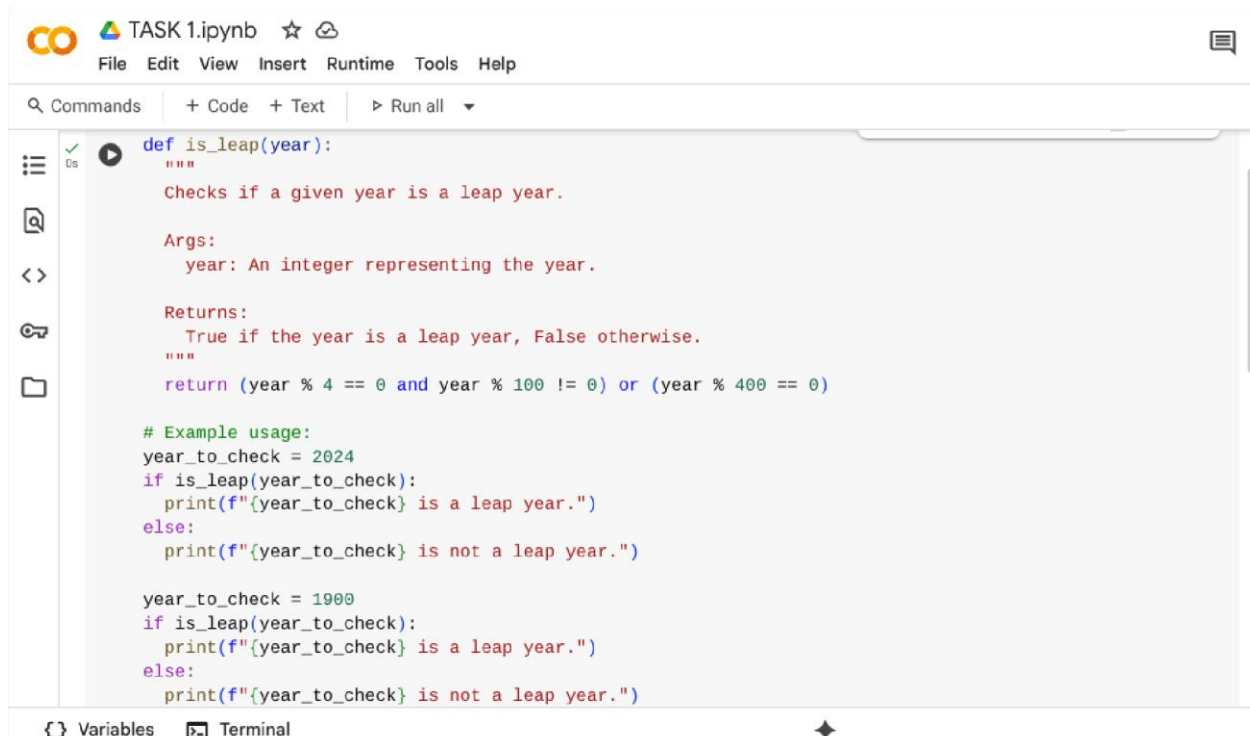
Name: N. Bhanuchander

Roll No: 2403A52085

## Task-1:

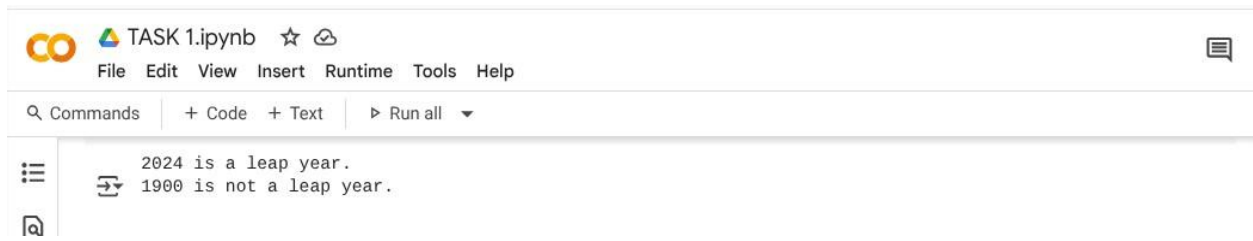
Write a python program that checks the given year is leap year using functions

## Code:



```
def is_leap(year):  
    """  
    Checks if a given year is a leap year.  
  
    Args:  
        year: An integer representing the year.  
  
    Returns:  
        True if the year is a leap year, False otherwise.  
    """  
    return (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0)  
  
# Example usage:  
year_to_check = 2024  
if is_leap(year_to_check):  
    print(f"{year_to_check} is a leap year.")  
else:  
    print(f"{year_to_check} is not a leap year.")  
  
year_to_check = 1900  
if is_leap(year_to_check):  
    print(f"{year_to_check} is a leap year.")  
else:  
    print(f"{year_to_check} is not a leap year.")
```


# Output:



A screenshot of a Jupyter Notebook interface. The top bar shows the file name 'TASK 1.ipynb' and various menu options like File, Edit, View, Insert, Runtime, Tools, and Help. Below the menu is a toolbar with icons for running code, adding code cells, and adding text cells. The main area displays the output of a Python function: '2024 is a leap year.' and '1900 is not a leap year.'

```
2024 is a leap year.  
1900 is not a leap year.
```

# Explanation:



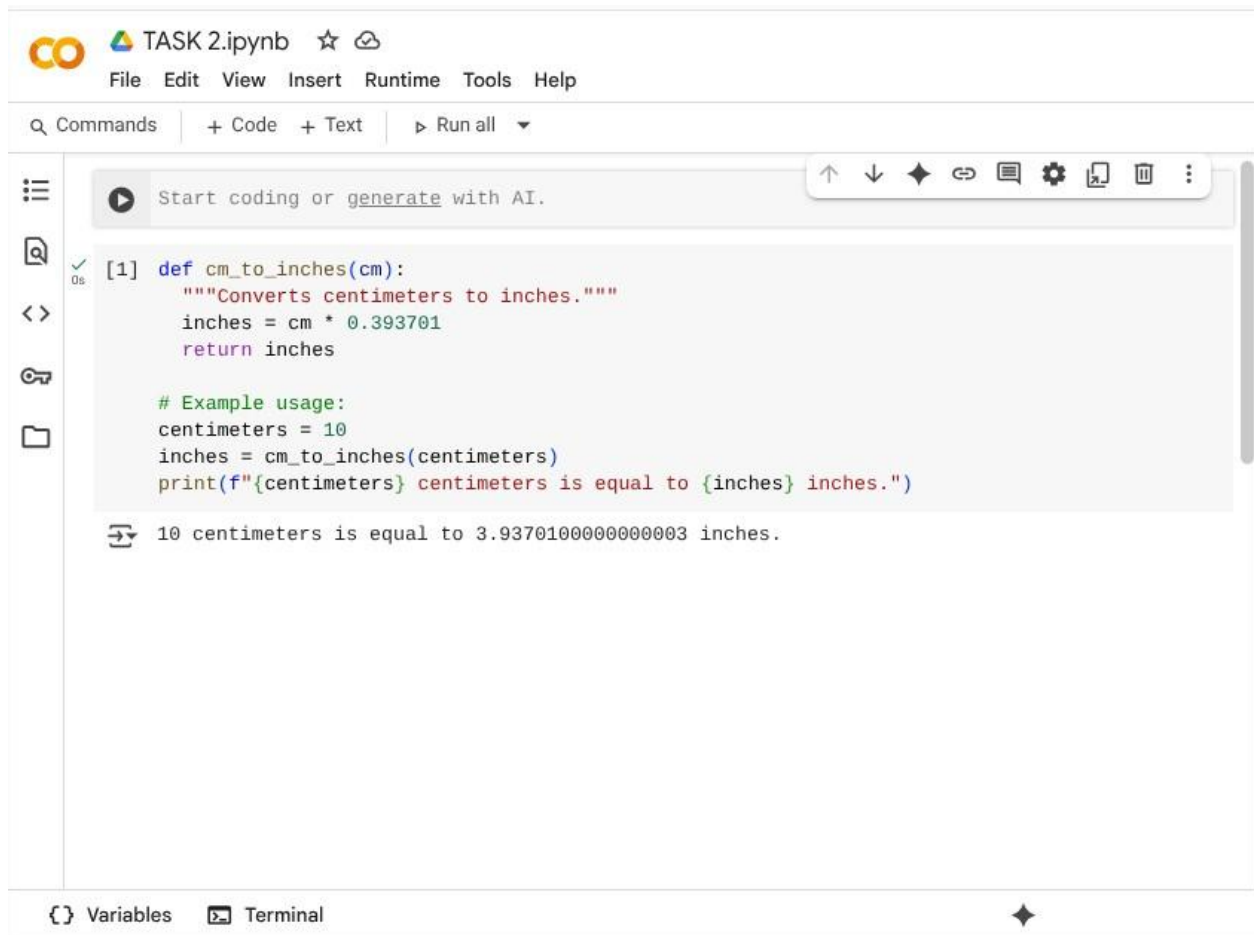
A screenshot of a Jupyter Notebook interface showing the explanation of the leap year function. The top bar is similar to the previous image, but it includes a 'Share' button and a 'Gemini' icon. The main area displays the explanation of the function, which is a docstring. The explanation is as follows:

```
def is_leap(year):  
    """  
    ...  
    """  
    return (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0)  
  
# Example usage:  
year_to_check = 2024  
if is_leap(year_to_check):  
    print(f"{year_to_check} is a leap year.")  
else:  
    print(f"{year_to_check} is not a leap year.")  
year_to_check = 1900  
if is_leap(year_to_check):  
    print(f"{year_to_check} is a leap year.")  
else:  
    print(f"{year_to_check} is not a leap year.")
```

## Task-2:

Write a program in python that converts centimeters into inches using one shot.

# Code and output:



The screenshot shows a Jupyter Notebook window titled "TASK 2.ipynb". The interface includes a menu bar (File, Edit, View, Insert, Runtime, Tools, Help), a command bar (Commands, + Code, + Text, Run all), and a left sidebar with icons for file explorer, search, and other functions. The main area displays a code cell with the following Python code:

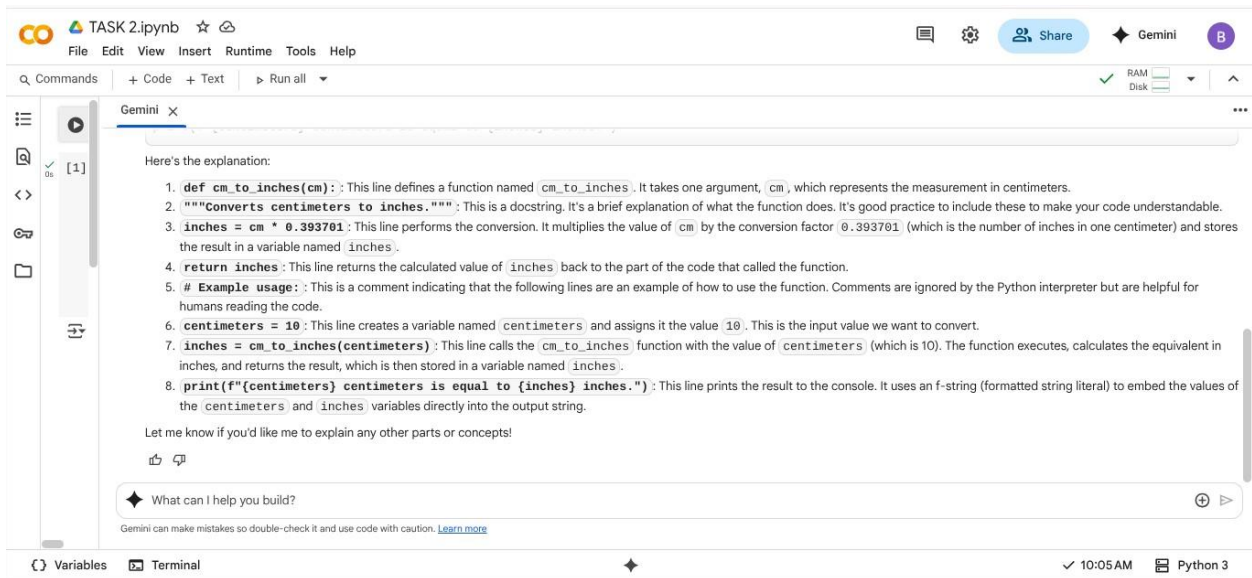
```
[1] def cm_to_inches(cm):  
    """Converts centimeters to inches."""  
    inches = cm * 0.393701  
    return inches  
  
# Example usage:  
centimeters = 10  
inches = cm_to_inches(centimeters)  
print(f"{centimeters} centimeters is equal to {inches} inches.")
```

Below the code cell, the output is displayed:

```
10 centimeters is equal to 3.9370100000000003 inches.
```

The bottom of the window shows tabs for "Variables" and "Terminal".

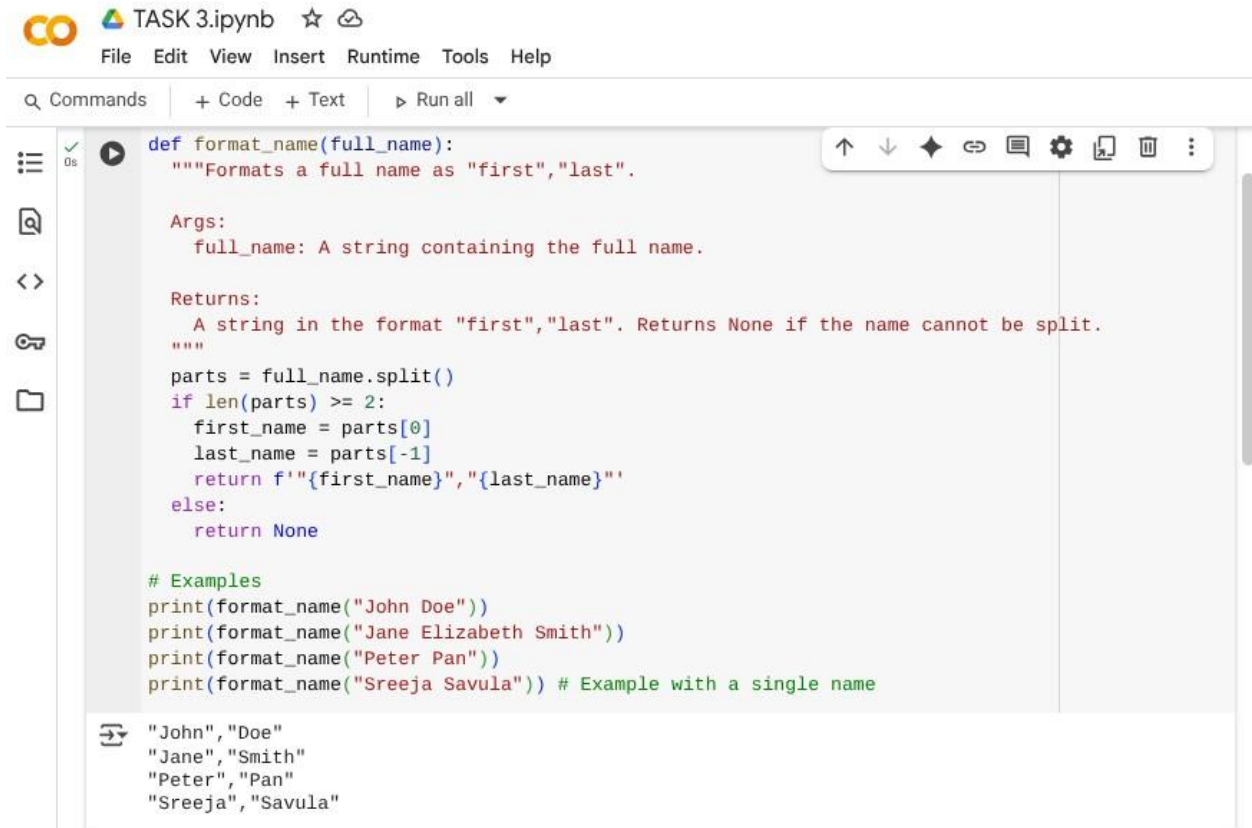
# Explanation:



## Task-3:

Write a python program that formats full names into “first” and “last” using few shot.

## Code and Output:



The image shows a Jupyter Notebook interface with a file named 'TASK 3.ipynb'. The notebook contains a Python function 'format\_name' that takes a 'full\_name' as input and returns a list of first and last names. The function uses 'split()' to separate the names. Examples are provided for 'John Doe', 'Jane Elizabeth Smith', 'Peter Pan', and 'Sreeja Savula'. The output of the function calls is displayed below the code.

```
def format_name(full_name):  
    """Formats a full name as "first","last".  
  
    Args:  
        full_name: A string containing the full name.  
  
    Returns:  
        A string in the format "first","last". Returns None if the name cannot be split.  
    """  
    parts = full_name.split()  
    if len(parts) >= 2:  
        first_name = parts[0]  
        last_name = parts[-1]  
        return f"{first_name}", "{last_name}"  
    else:  
        return None  
  
# Examples  
print(format_name("John Doe"))  
print(format_name("Jane Elizabeth Smith"))  
print(format_name("Peter Pan"))  
print(format_name("Sreeja Savula")) # Example with a single name
```

Output:

```
"John", "Doe"  
"Jane", "Smith"  
"Peter", "Pan"  
"Sreeja", "Savula"
```

# Explanation:



TASK 3.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

RAM Disk

Gemini X

```
print(format_name("Single Name")) # Example with a single name
```

1. `def format_name(full_name):` : This line defines a function named `format_name` that takes one argument, `full_name`.
2. `"""Formats a full name as "first", "last". ... Returns None if the name cannot be split."""` : This is a docstring, which explains what the function does, its arguments (`Args`), and what it returns (`Returns`). It's good practice to include docstrings to make your code understandable.
3. `parts = full_name.split()` : This line splits the input `full_name` string into a list of words using spaces as delimiters. For example, if `full_name` is "John Doe", `parts` will be `['John', 'Doe']`.
4. `if len(parts) >= 2:` : This line checks if the `parts` list has two or more elements. This is to ensure that there is at least a first and a last name.
5. `first_name = parts[0]` : If the condition in the `if` statement is true, this line assigns the first element of the `parts` list (which is the first name) to the variable `first_name`.
6. `last_name = parts[-1]` : This line assigns the last element of the `parts` list (which is the last name) to the variable `last_name`. Using `[-1]` is a convenient way to access the last element of a list in Python.
7. `return f"{first_name}", "{last_name}"` : This line constructs the output string in the format `"first", "last"` using an f-string. The `f` before the opening quote indicates an f-string, which allows you to embed the values of variables directly within the string using curly braces `{}`. The function then returns this formatted string.
8. `else:` : This indicates what to do if the `if` condition (`len(parts) >= 2`) is false, meaning the name has less than two parts.
9. `return None` : If the name cannot be split into at least two parts, the function returns `None`.
10. `# Examples:` : This is a comment indicating the start of the example usage of the function.
11. `print(format_name("John Doe"))` : This line calls the `format_name` function with the input "John Doe" and prints the returned value.
12. `print(format_name("Jane Elizabeth Smith"))` : This line calls the `format_name` function with the input "Jane Elizabeth Smith" and prints the returned value.
13. `print(format_name("Peter Pan"))` : This line calls the `format_name` function with the input "Peter Pan" and prints the returned value.
14. `print(format_name("Single Name"))` : This line calls the `format_name` function with the input "Single Name" and prints the returned value. The comment explains that this is an example with a single name.

Let me know if you have any more questions or would like to explore other ways to format names!

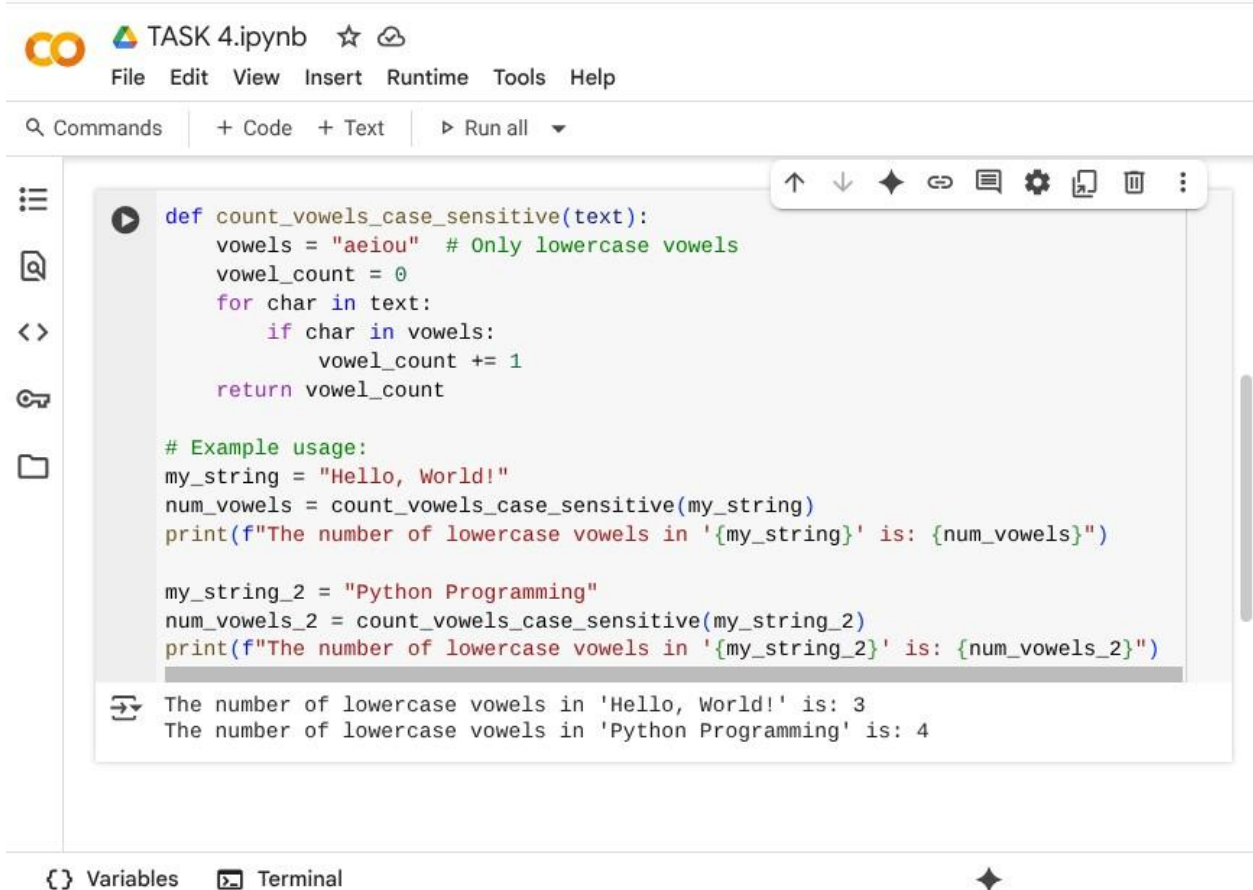
What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

## Task-4:

Write a python program that counts the number of vowels in a string.

Zero Shot Code and Output:



The screenshot shows a Jupyter Notebook window titled "TASK 4.ipynb". The interface includes a top menu bar with "File", "Edit", "View", "Insert", "Runtime", "Tools", and "Help". Below the menu is a toolbar with "Commands", "+ Code", "+ Text", and "Run all". The main area contains a code cell with the following Python code:

```
def count_vowels_case_sensitive(text):  
    vowels = "aeiou" # Only lowercase vowels  
    vowel_count = 0  
    for char in text:  
        if char in vowels:  
            vowel_count += 1  
    return vowel_count  
  
# Example usage:  
my_string = "Hello, World!"  
num_vowels = count_vowels_case_sensitive(my_string)  
print(f"The number of lowercase vowels in '{my_string}' is: {num_vowels}")  
  
my_string_2 = "Python Programming"  
num_vowels_2 = count_vowels_case_sensitive(my_string_2)  
print(f"The number of lowercase vowels in '{my_string_2}' is: {num_vowels_2}")
```

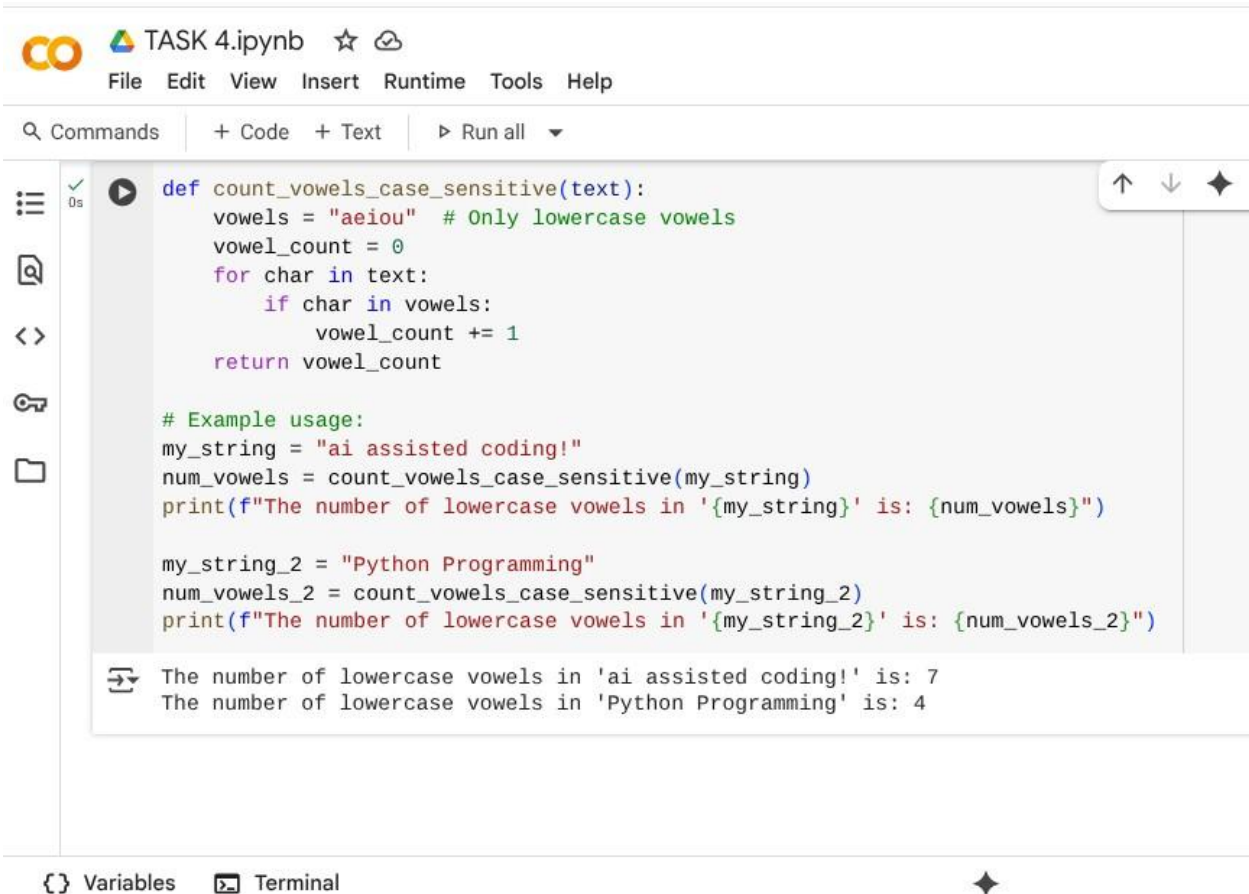
Below the code cell, the output is displayed:

```
The number of lowercase vowels in 'Hello, World!' is: 3  
The number of lowercase vowels in 'Python Programming' is: 4
```

At the bottom of the notebook, there are tabs for "Variables" and "Terminal".

## Few Shot Code and Output:





The screenshot shows a Jupyter Notebook titled "TASK 4.ipynb". The interface includes a menu bar (File, Edit, View, Insert, Runtime, Tools, Help) and a toolbar with icons for commands, code, text, and running. The main area displays a Python function `count_vowels_case_sensitive` that counts only lowercase vowels. Below the function, example usage is shown with two strings: "ai assisted coding!" and "Python Programming". The output of the function calls is displayed below the code, showing 7 vowels for the first string and 4 for the second. At the bottom, there are tabs for "Variables" and "Terminal".

```
def count_vowels_case_sensitive(text):  
    vowels = "aeiou" # Only lowercase vowels  
    vowel_count = 0  
    for char in text:  
        if char in vowels:  
            vowel_count += 1  
    return vowel_count  
  
# Example usage:  
my_string = "ai assisted coding!"  
num_vowels = count_vowels_case_sensitive(my_string)  
print(f"The number of lowercase vowels in '{my_string}' is: {num_vowels}")  
  
my_string_2 = "Python Programming"  
num_vowels_2 = count_vowels_case_sensitive(my_string_2)  
print(f"The number of lowercase vowels in '{my_string_2}' is: {num_vowels_2}")
```

The number of lowercase vowels in 'ai assisted coding!' is: 7  
The number of lowercase vowels in 'Python Programming' is: 4

## Comparision between Zero shot and Few shot code Explanation:

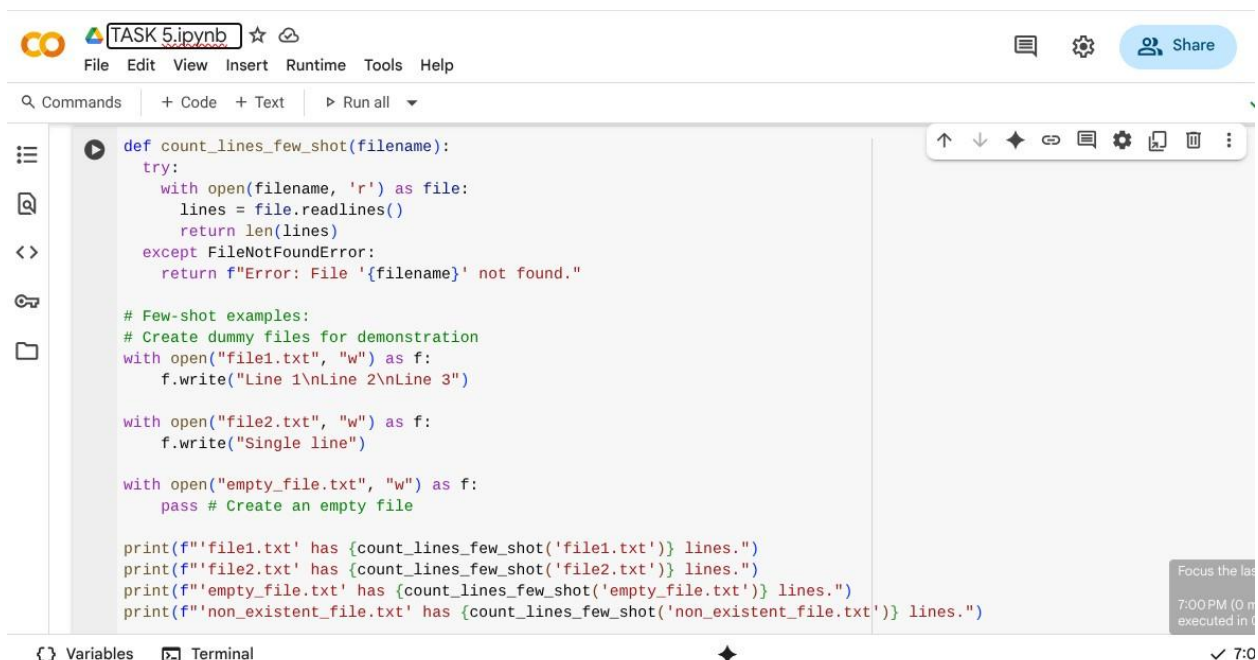
1. The first function `count_vowels_case_sensitive` checks only **lowercase vowels (a, e, i, o, u)**.
2. This means uppercase vowels (like A, E, I, O, U) are **not counted**.
3. For example, "Hello, World!" → counts only "o" → result = 1.
4. "Python Programming" → counts "o", "o", "a", "i" → result = 4.
5. The second function `count_vowels_few_shot` checks for **both lowercase and uppercase vowels**.
6. It defines vowels as "aeiouAEIOU", making the function **case-insensitive**.
7. For example, "Hello World" → counts "e", "o", "o" → result = 3.
8. "AEIOUaeiou" → counts all 10 vowels → result = 10.
9. The first code is useful when only lowercase vowels matter, while the second is more **general-purpose**.
10. Overall, the difference lies in **case sensitivity**: first = lowercase only, second = both cases.



# Task-5:

Write a python program that reads a .txt file and returns the number of lines using few short.

## Code:



The screenshot shows a Jupyter Notebook titled "TASK 5.ipynb". The code defines a function `count_lines_few_shot(filename)` that reads a file and returns the number of lines. It includes error handling for `FileNotFoundError`. Below the function, there are comments and code to create dummy files for demonstration: `file1.txt` (3 lines), `file2.txt` (1 line), and `empty_file.txt` (0 lines). The code also includes print statements to verify the function's output for these files and for a non-existent file.

```
def count_lines_few_shot(filename):
    try:
        with open(filename, 'r') as file:
            lines = file.readlines()
            return len(lines)
    except FileNotFoundError:
        return f"Error: File '{filename}' not found."

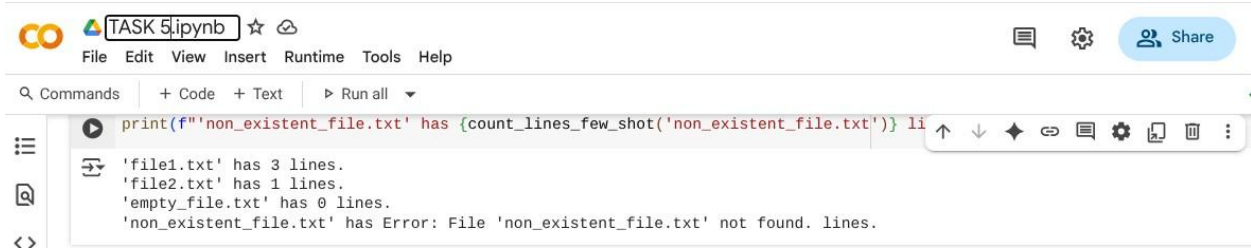
# Few-shot examples:
# Create dummy files for demonstration
with open("file1.txt", "w") as f:
    f.write("Line 1\nLine 2\nLine 3")

with open("file2.txt", "w") as f:
    f.write("Single line")

with open("empty_file.txt", "w") as f:
    pass # Create an empty file

print(f"'file1.txt' has {count_lines_few_shot('file1.txt')} lines.")
print(f"'file2.txt' has {count_lines_few_shot('file2.txt')} lines.")
print(f"'empty_file.txt' has {count_lines_few_shot('empty_file.txt')} lines.")
print(f"'non_existent_file.txt' has {count_lines_few_shot('non_existent_file.txt')} lines.")
```

## Output:



The screenshot shows a Jupyter Notebook window titled 'TASK 5.ipynb'. The menu bar includes 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. Below the menu is a toolbar with icons for commands, code, text, and running all cells. The code cell contains a function definition and several print statements. The output cell shows the results of these print statements, including an error for a non-existent file.

```
print(f"'non_existent_file.txt' has {count_lines_few_shot('non_existent_file.txt')} lines.")
```

```
'file1.txt' has 3 lines.  
'file2.txt' has 1 lines.  
'empty_file.txt' has 0 lines.  
'non_existent_file.txt' has Error: File 'non_existent_file.txt' not found. lines.
```

# Explanation:

1. **def count\_lines\_few\_shot(filename)::** Defines a function to count lines in a file.
2. **""" ... """:** A docstring explaining the function's purpose, arguments, and return value.
3. **try::** Starts a block to handle potential errors, like the file not being found.
4. **with open(filename, 'r') as file::** Opens the specified file for reading.
5. **lines = file.readlines():** Reads all lines into a list.
6. **return len(lines):** Returns the number of lines in the list.
7. **except FileNotFoundError::** Catches the error if the file doesn't exist.
8. **return f"Error: File '{filename}' not found.":** Returns an error message if the file is not found.
9. **# Few-shot examples::** Comments indicating the following lines are examples.
10. **# Create dummy files for demonstration:** Comment explaining dummy file creation.
11. **with open("file1.txt", "w") as f: f.write("Line 1\nLine 2\nLine 3"):** Creates "file1.txt" with three lines.
12. **with open("file2.txt", "w") as f: f.write("Single line"):** Creates "file2.txt" with one line.
13. **with open("empty\_file.txt", "w") as f: pass:** Creates an empty "empty\_file.txt".
14. **print(f"'file1.txt' has {count\_lines\_few\_shot('file1.txt')} lines."):**  Prints the line count for "file1.txt".
15. **print(f"'non\_existent\_file.txt' has {count\_lines\_few\_shot('non\_existent\_file.txt')} lines."):**  Prints the result for a non-existent file (showing the error handling).