

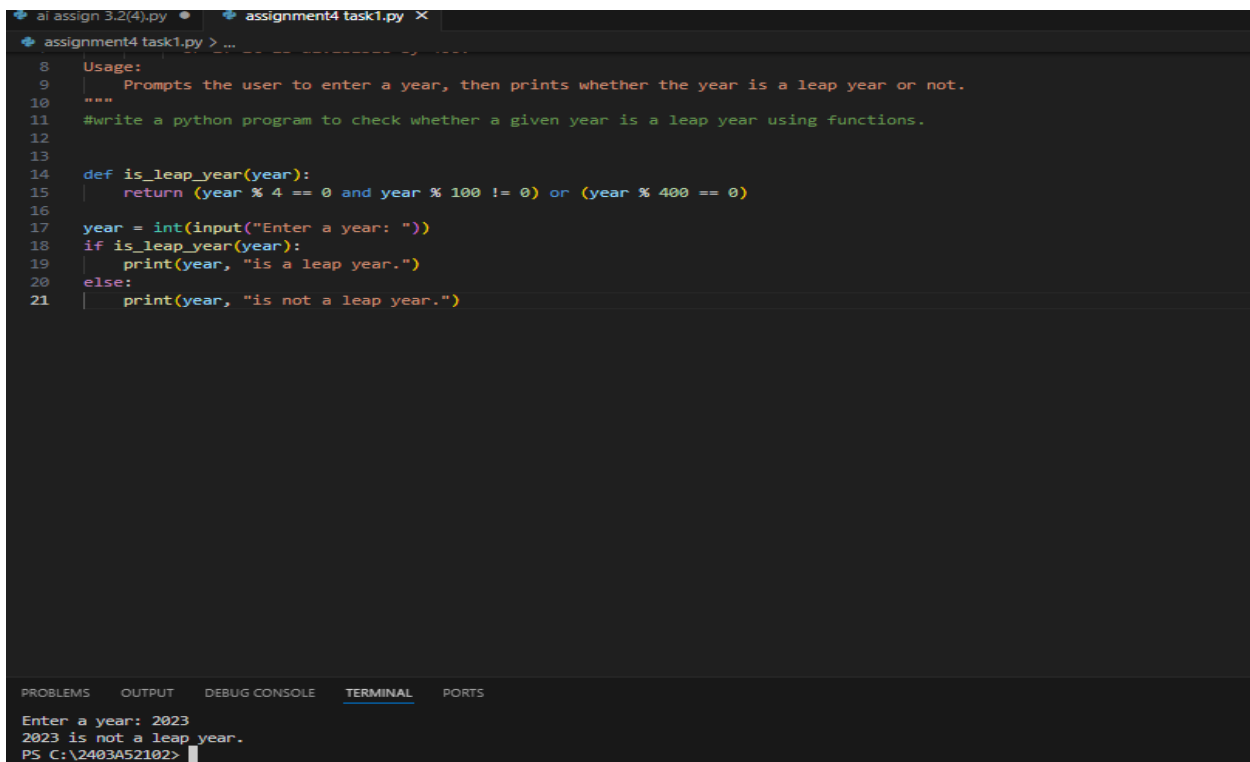
## ASSIGNMENT-4.3

### TASK-1:

#### Prompt:

#write a python program to check weather a given year is leap year using functions.

#### Code and Output:

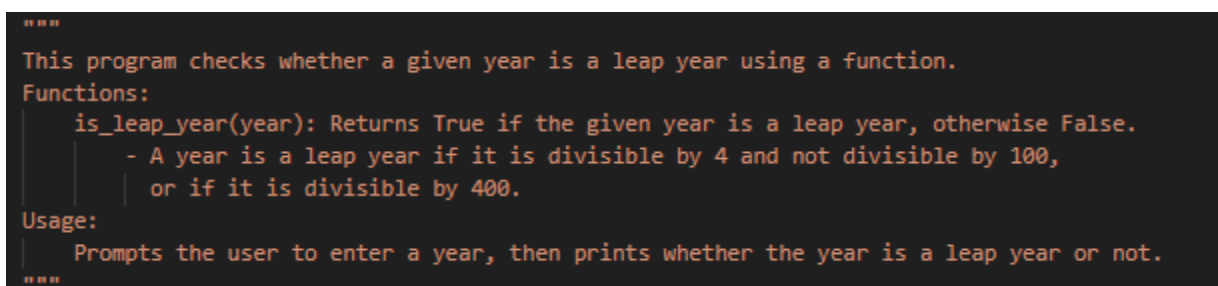


```
8  Usage:
9  | Prompts the user to enter a year, then prints whether the year is a leap year or not.
10 |
11 | """
12 | #write a python program to check whether a given year is a leap year using functions.
13 |
14 | def is_leap_year(year):
15 |     return (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0)
16 |
17 | year = int(input("Enter a year: "))
18 | if is_leap_year(year):
19 |     print(year, "is a leap year.")
20 | else:
21 |     print(year, "is not a leap year.")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Enter a year: 2023
2023 is not a leap year.
PS C:\2403A52102>
```

#### Explanation:



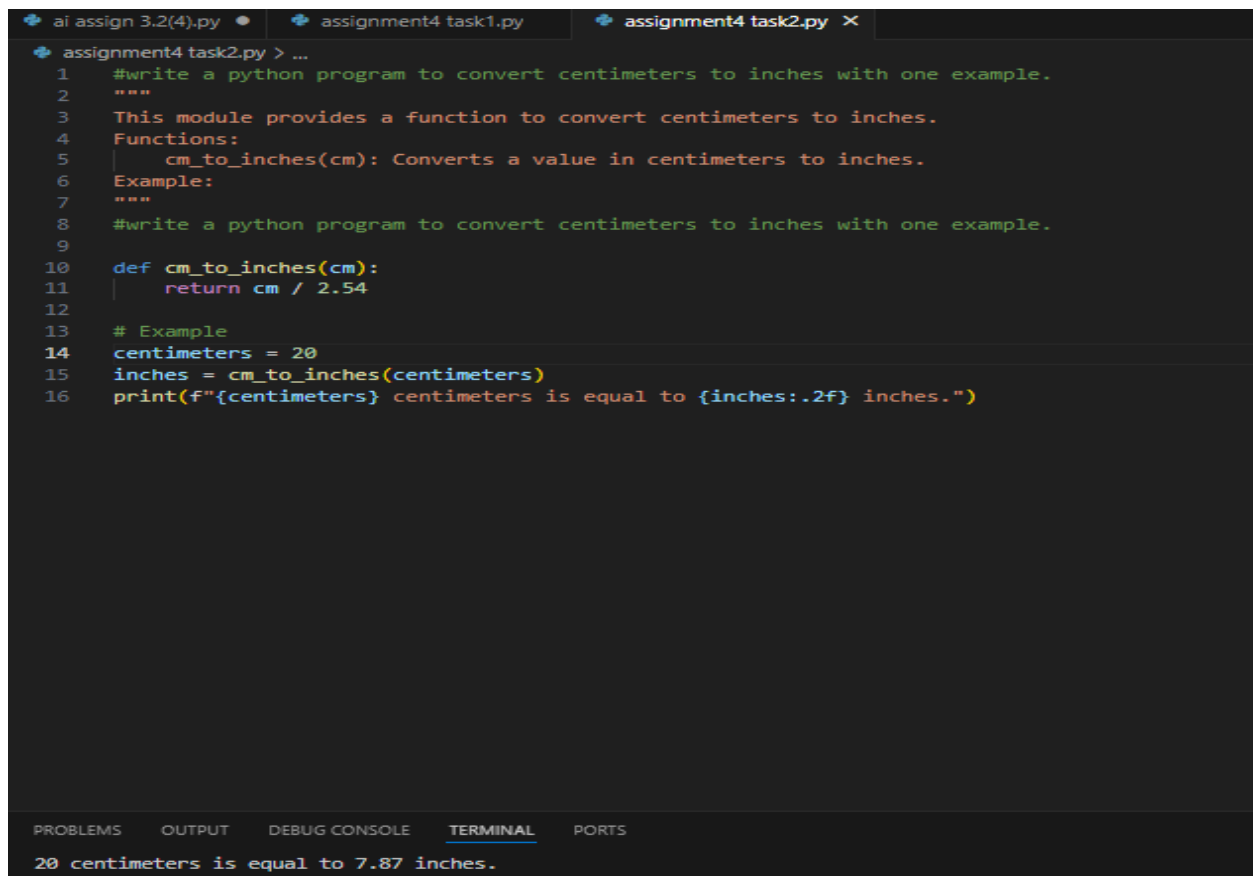
```
"""
This program checks whether a given year is a leap year using a function.
Functions:
    is_leap_year(year): Returns True if the given year is a leap year, otherwise False.
        - A year is a leap year if it is divisible by 4 and not divisible by 100,
          or if it is divisible by 400.
Usage:
    Prompts the user to enter a year, then prints whether the year is a leap year or not.
"""
```

## TASK-2:

### Prompt:

#write a python program to convert centimeters to inches with one example.

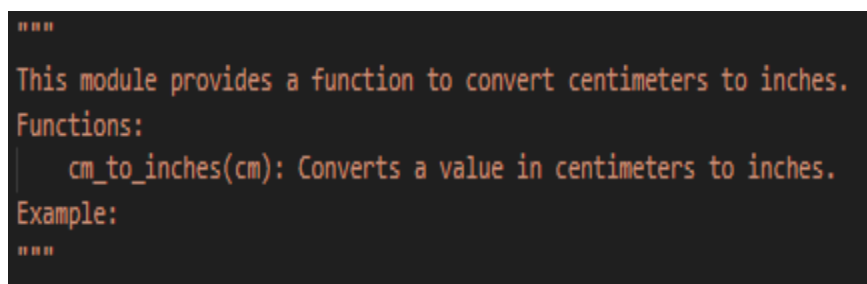
### Code and Output:

A screenshot of a code editor with three tabs: 'ai assign 3.2(4).py', 'assignment4 task1.py', and 'assignment4 task2.py'. The active tab is 'assignment4 task2.py'. The code in the editor is a Python program that defines a function 'cm\_to\_inches' and includes a docstring with an example. The code is as follows:

```
1 #write a python program to convert centimeters to inches with one example.
2 """
3 This module provides a function to convert centimeters to inches.
4 Functions:
5     cm_to_inches(cm): Converts a value in centimeters to inches.
6 Example:
7 """
8 #write a python program to convert centimeters to inches with one example.
9
10 def cm_to_inches(cm):
11     return cm / 2.54
12
13 # Example
14 centimeters = 20
15 inches = cm_to_inches(centimeters)
16 print(f"{centimeters} centimeters is equal to {inches:.2f} inches.")
```

The bottom of the screenshot shows the 'TERMINAL' tab with the output: '20 centimeters is equal to 7.87 inches.'

### Explanation:

A close-up screenshot of the docstring part of the Python code. It shows the following text:

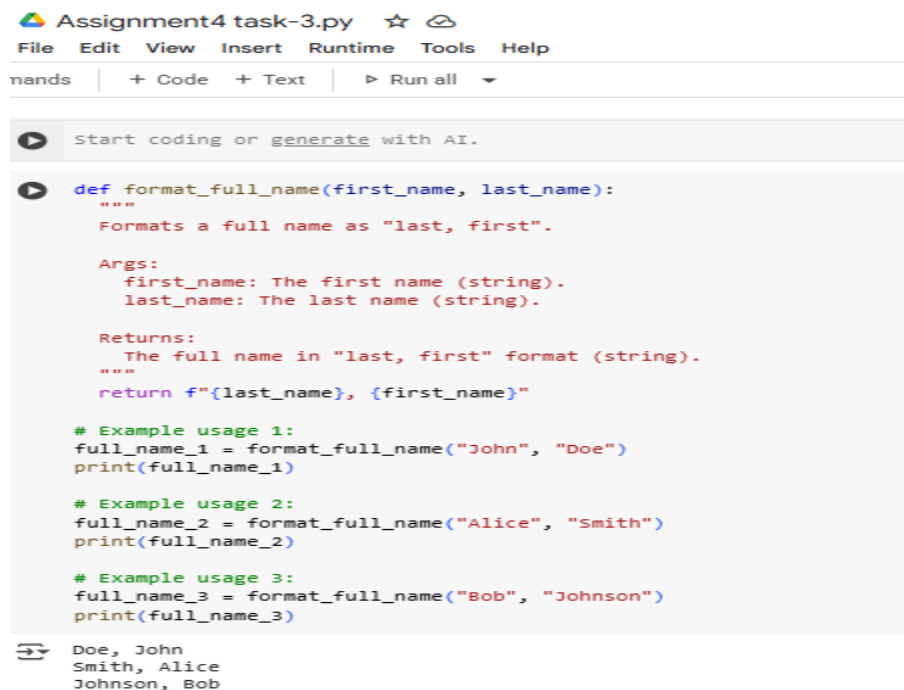
```
"""
This module provides a function to convert centimeters to inches.
Functions:
    cm_to_inches(cm): Converts a value in centimeters to inches.
Example:
"""
```

## TASK-3:

### Prompt:

#write a python program to generate a function that full name as “last, first”.

### Code and Output:




The screenshot shows a code editor window titled "Assignment4 task-3.py". The menu bar includes "File", "Edit", "View", "Insert", "Runtime", "Tools", and "Help". Below the menu bar, there are tabs for "nands", "+ Code", and "+ Text", along with a "Run all" button. The main code area contains a Python function `format_full_name` with a docstring and three example usage blocks. The output area at the bottom shows the results of the function calls: "Doe, John", "Smith, Alice", and "Johnson, Bob".

```
def format_full_name(first_name, last_name):  
    """  
    Formats a full name as "last, first".  
  
    Args:  
        first_name: The first name (string).  
        last_name: The last name (string).  
  
    Returns:  
        The full name in "last, first" format (string).  
    """  
    return f"{last_name}, {first_name}"  
  
# Example usage 1:  
full_name_1 = format_full_name("John", "Doe")  
print(full_name_1)  
  
# Example usage 2:  
full_name_2 = format_full_name("Alice", "Smith")  
print(full_name_2)  
  
# Example usage 3:  
full_name_3 = format_full_name("Bob", "Johnson")  
print(full_name_3)
```

Doe, John  
Smith, Alice  
Johnson, Bob

### Explanation:

 explain the code

◆ No problem! This code defines a Python function called `format_name` that takes a full name as input and returns the name formatted as "last, first".

Here's a breakdown of the code:

- `def format_name(full_name):`: This line defines the function `format_name` that accepts one argument, `full_name`.
- `"""Docstring"""`: This is a docstring that explains what the function does, its arguments, and what it returns.
- `names = full_name.split()`: This line splits the input `full_name` string into a list of strings using spaces as delimiters. For example, "John Doe" becomes `['John', 'Doe']`.
- `if len(names) >= 2:`: This checks if the list of names has at least two elements (meaning there's at least a first and a last name).
- `return f"{names[-1]}, {' '.join(names[:-1])}"`: If there are at least two names, this line formats the output string.
  - `names[-1]` gets the last element of the `names` list (the last name).
  - `names[:-1]` gets all elements except the last one (the first and any middle names).
  - `' '.join(names[:-1])` joins the elements of `names[:-1]` back into a single string with spaces in between.
  - The f-string `f"{names[-1]}, {' '.join(names[:-1])}"` combines the last name, a comma and a space, and the rest of the names.
- `else: return full_name`: If the input name has fewer than two parts, the function returns the original `full_name` without any formatting.
- `print(...)`: These lines demonstrate how to use the `format_name` function with different examples.

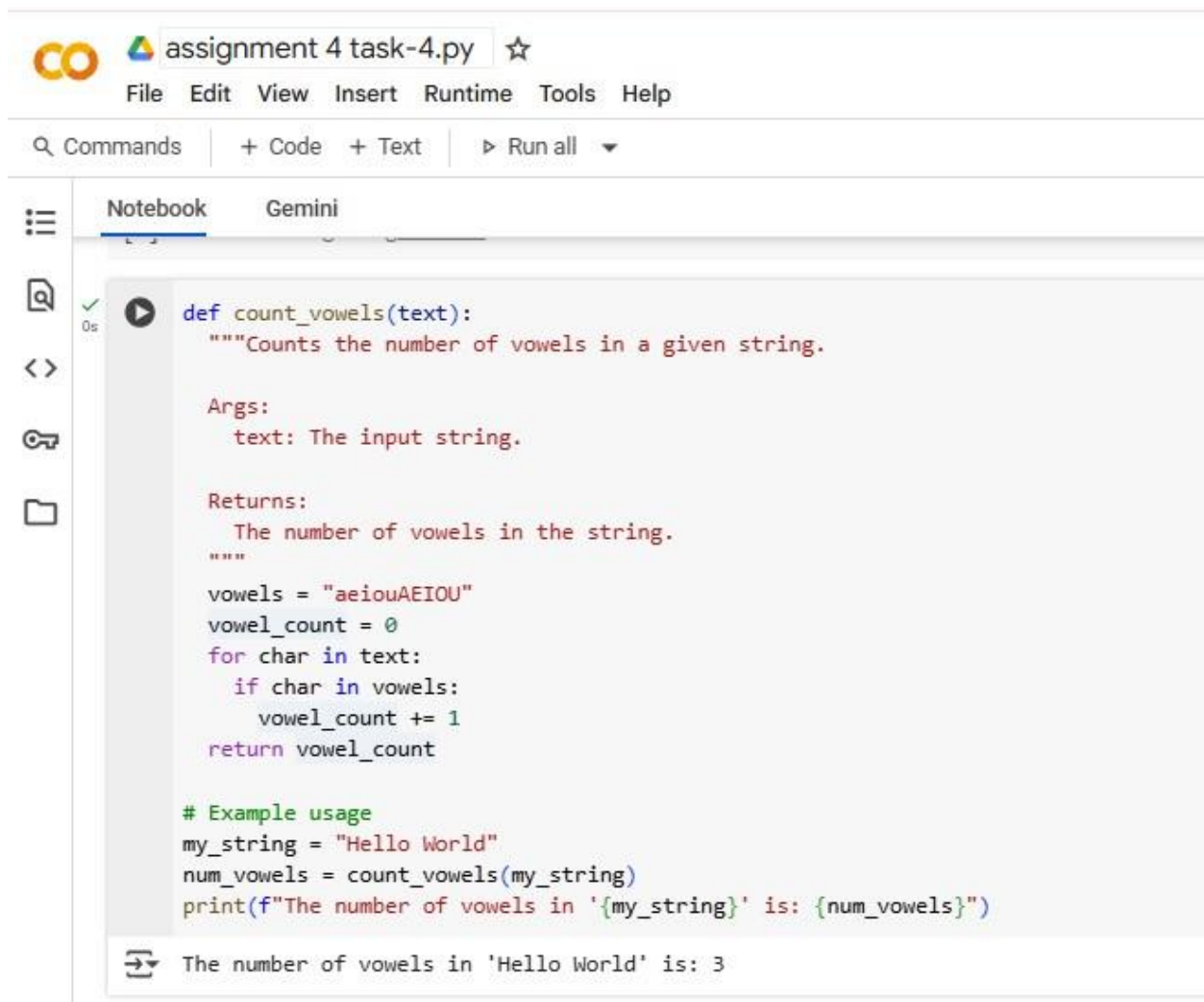
## Task-4:

### Prompt:

#write a python program to count number vowels in a string using functions.

### Code with Output:

**Zero Shot:** count number vowels in a string using functions.





The screenshot shows a code editor interface with a file named "assignment 4 task-4.py". The editor has a menu bar with "File", "Edit", "View", "Insert", "Runtime", "Tools", and "Help". Below the menu bar is a toolbar with "Commands", "+ Code", "+ Text", and "Run all". The editor is in "Notebook" mode, and the code is written in a Python file. The code defines a function "count\_vowels(text)" that counts the number of vowels in a given string. The function has a docstring, arguments, and returns the count. An example usage is provided at the bottom, showing the function being called with "Hello World" and the output being printed.


```
def count_vowels(text):  
    """Counts the number of vowels in a given string.  
  
    Args:  
        text: The input string.  
  
    Returns:  
        The number of vowels in the string.  
    """  
    vowels = "aeiouAEIOU"  
    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(my_string)  
print(f"The number of vowels in '{my_string}' is: {num_vowels}")
```

The output of the code is: The number of vowels in 'Hello World' is: 3


**Few Shot:** Count number of vowels in a string using functions.

 assignment 4 task-4.py ☆  
File Edit View Insert Runtime Tools Help  
Commands | + Code + Text | ▶ Run all ▼  


```
def count_vowels(text):  
    vowels = "aeiouAEIOU"  
    count = 0  
    for char in text:  
        if char in vowels:  
            count += 1  
    return count  
  
# Example 1  
input_string1 = "Hello"  
vowel_count1 = count_vowels(input_string1)  
print(f"The number of vowels in '{input_string1}' is: {vowel_count1}")  
  
# Example 2  
input_string2 = "Programming"  
vowel_count2 = count_vowels(input_string2)  
print(f"The number of vowels in '{input_string2}' is: {vowel_count2}")  
  
# Example 3  
input_string3 = "few-shot"  
vowel_count3 = count_vowels(input_string3)  
print(f"The number of vowels in '{input_string3}' is: {vowel_count3}")
```

  
 The number of vowels in 'Hello' is: 2  
The number of vowels in 'Programming' is: 3  
The number of vowels in 'few-shot' is: 2

**Explanation of both example of zero shot code and example of few shot code:**

 explain the code step by step

◆ Sure, I can explain the code step by step.

Here's a breakdown of the code:

1. `def count_vowels(text):`: This line defines a function named `count_vowels` that takes one argument, `text`. This function will contain the logic for counting vowels.
2. `"""Counts the number of vowels in a given string. ... """`: 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. `vowels = "aeiouAEIOU"`: This line creates a string variable named `vowels` and assigns it the lowercase and uppercase letters that are considered vowels.
4. `vowel_count = 0`: This line initializes an integer variable named `vowel_count` to 0. This variable will keep track of the number of vowels found in the input string.
5. `for char in text:`: This is a `for` loop that iterates through each character in the input `text` string. In each iteration, the current character is assigned to the variable `char`.
6. `if char in vowels:`: Inside the loop, this `if` statement checks if the current character (`char`) is present in the `vowels` string. This is how we determine if a character is a vowel.
7. `vowel_count += 1`: If the `if` condition is true (meaning the character is a vowel), this line increments the `vowel_count` by 1.
8. `return vowel_count`: After the loop has finished iterating through all the characters in the `text` string, this line returns the final value of `vowel_count`, which is the total number of vowels found.
9. `my_string = "Hello World"`: This line creates a string variable named `my_string` and assigns it the value "Hello World". This is the string we will test our function with.
10. `num_vowels = count_vowels(my_string)`: This line calls the `count_vowels` function with `my_string` as the argument. The value returned by the function (the vowel count) is then stored in the variable `num_vowels`.
11. `print(f"The number of vowels in '{my_string}' is: {num_vowels}")`: This line prints the result to the console. It uses an f-string (formatted string literal) to embed the values of `my_string` and `num_vowels` directly into the output string.

```
def count_vowels(text):
```

This line defines a function named `count_vowels` that accepts one argument, `text`.

```
vowels = "aeiouAEIOU"
```

This line initializes a string variable called `vowels` containing all lowercase and uppercase vowels. This string will be used to check if a character is a vowel.

```
count = 0
```

This line initializes an integer variable called `count` to 0. This variable will store the total number of vowels found in the input text.

```
for char in text:
```

This line starts a `for` loop that iterates through each character in the `text` string. In each iteration, the current character is assigned to the variable `char`.

```
    if char in vowels:
```

This line checks if the current character (`char`) is present in the `vowels` string.

```
        count += 1
```

If the condition in the `if` statement is true (meaning the character is a vowel), this line increments the `count` variable by 1.

```
    return count
```

After the loop finishes iterating through all characters in the `text`, this line returns the final value of `count`, which represents the total number of vowels in the input string.

---

The rest of the code demonstrates how to use the `count_vowels` function with example strings and print the results.

```
# Example 1
input_string1 = "Hello"
vowel_count1 = count_vowels(input_string1)
print(f"The number of vowels in '{input_string1}' is: {vowel_count1}")

# Example 2
input_string2 = "Programming"
vowel_count2 = count_vowels(input_string2)
print(f"The number of vowels in '{input_string2}' is: {vowel_count2}")

# Example 3
input_string3 = "few-shot"
vowel_count3 = count_vowels(input_string3)
print(f"The number of vowels in '{input_string3}' is: {vowel_count3}")
```

These lines define three example strings, call the `count_vowels` function with each string, store the returned vowel count in variables, and then print the results in a user-friendly format using f-strings.


## TASK-5:

### Prompt:

#write a python program that reads a txt file and returns the number of lines.

### Code with Output:

```
def count_lines_in_file(filename):  
    """  
    Reads a text file and returns the number of lines.  
  
    Args:  
        filename: The path to the text file.  
  
    Returns:  
        The number of lines in the file, or -1 if the file is not found.  
    """  
    try:  
        with open(filename, 'r') as f:  
            line_count = sum(1 for line in f)  
            return line_count  
    except FileNotFoundError:  
        print(f"Error: File '{filename}' not found.")  
        return -1  
  
# Example usage:  
# Create a dummy file for testing  
with open("my_test_file.txt", "w") as f:  
    f.write("This is line 1.\n")  
    f.write("This is line 2.\n")  
    f.write("This is line 3.\n")  
  
file_name = "my_test_file.txt"  
number_of_lines = count_lines_in_file(file_name)  
  
if number_of_lines != -1:  
    print(f"The file '{file_name}' has {number_of_lines} lines.")
```

 The file 'my\_test\_file.txt' has 3 lines.



## Explanation:

💡 Certainly! Here's an explanation of the code you provided:

This Python code defines a function called `count_lines_in_file` that aims to count the number of lines in a given text file.

- `def count_lines_in_file(filename):`: This line defines the function named `count_lines_in_file` that takes one argument, `filename`, which is expected to be the path to the file.
- `""" Reads a text file and returns the number of lines. ... """`: This is a docstring, which explains what the function does, its arguments (`Args`), and what it returns (`Returns`).
- `try:`: This block starts a `try...except` block, which is used for error handling. The code inside the `try` block is attempted first. If an error occurs, the code in the corresponding `except` block is executed.
- `with open(filename, 'r') as f:`: This line opens the file specified by `filename` in read mode (`'r'`). The `with` statement ensures that the file is automatically closed even if errors occur. The opened file object is assigned to the variable `f`.
- `line_count = sum(1 for line in f)`: This is a concise way to count the lines. It uses a generator expression `(1 for line in f)` which yields `1` for each line in the file. The `sum()` function then sums up these `1`s, effectively counting the lines.
- `return line_count`: If the file is opened and read successfully, the function returns the calculated `line_count`.
- `except FileNotFoundError:`: This block catches the specific error `FileNotFoundError`, which occurs if the file specified by `filename` does not exist.
- `print(f"Error: File '{filename}' not found.")`: If a `FileNotFoundError` occurs, this line prints an error message indicating that the file was not found.
- `return -1`: If a `FileNotFoundError` occurs, the function returns `-1` to indicate that the file was not found and the line count could not be determined.

The code then demonstrates how to use this function:

- `with open("my_test_file.txt", "w") as f:`: This part creates a dummy file named `my_test_file.txt` in write mode (`"w"`) and writes three lines of text into it. This is done so that the `count_lines_in_file` function has a file to operate on for the example.
- `file_name = "my_test_file.txt"`: This line assigns the filename to a variable.
- `number_of_lines = count_lines_in_file(file_name)`: This line calls the `count_lines_in_file` function with the dummy filename and stores the returned value (the number of lines or `-1`) in the `number_of_lines` variable.
- `if number_of_lines != -1:`: This checks if the function returned a valid line count (i.e., not `-1`).
- `print(f"The file '{file_name}' has {number_of_lines} lines.")`: If the file was found and processed successfully, this line prints the number of lines in the file.

In summary, the code defines a reusable function to count lines in a file and includes basic error handling. The example usage demonstrates how to create a file and then use the function to count its lines.