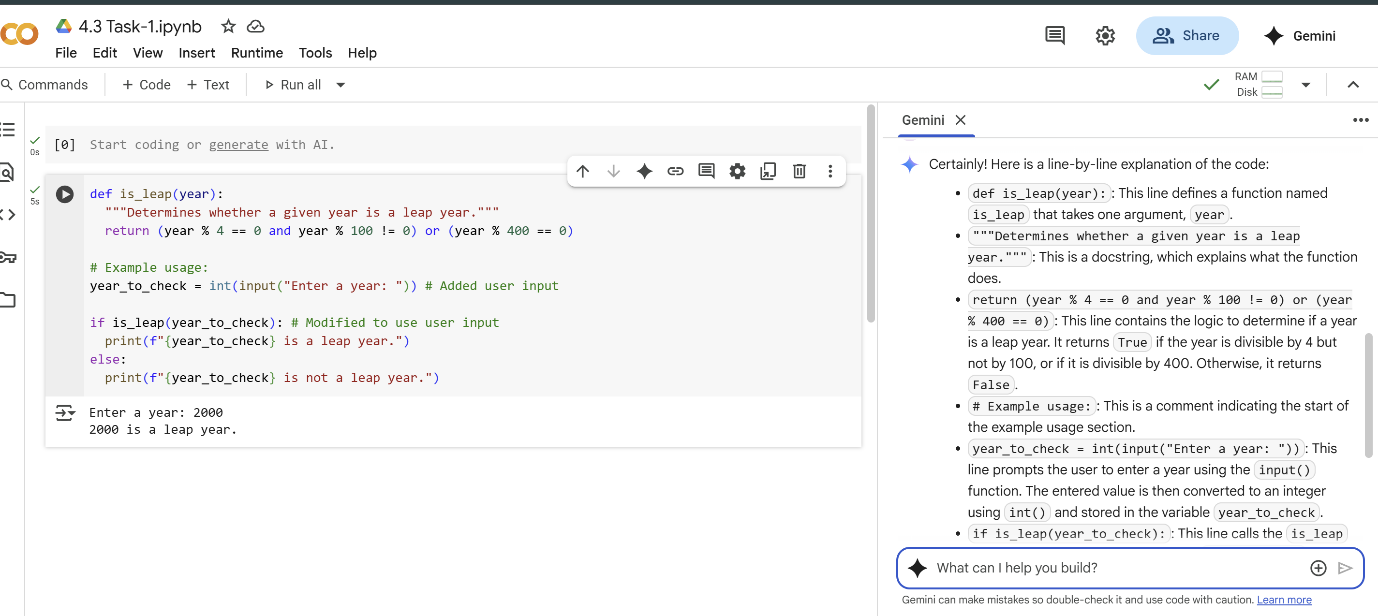
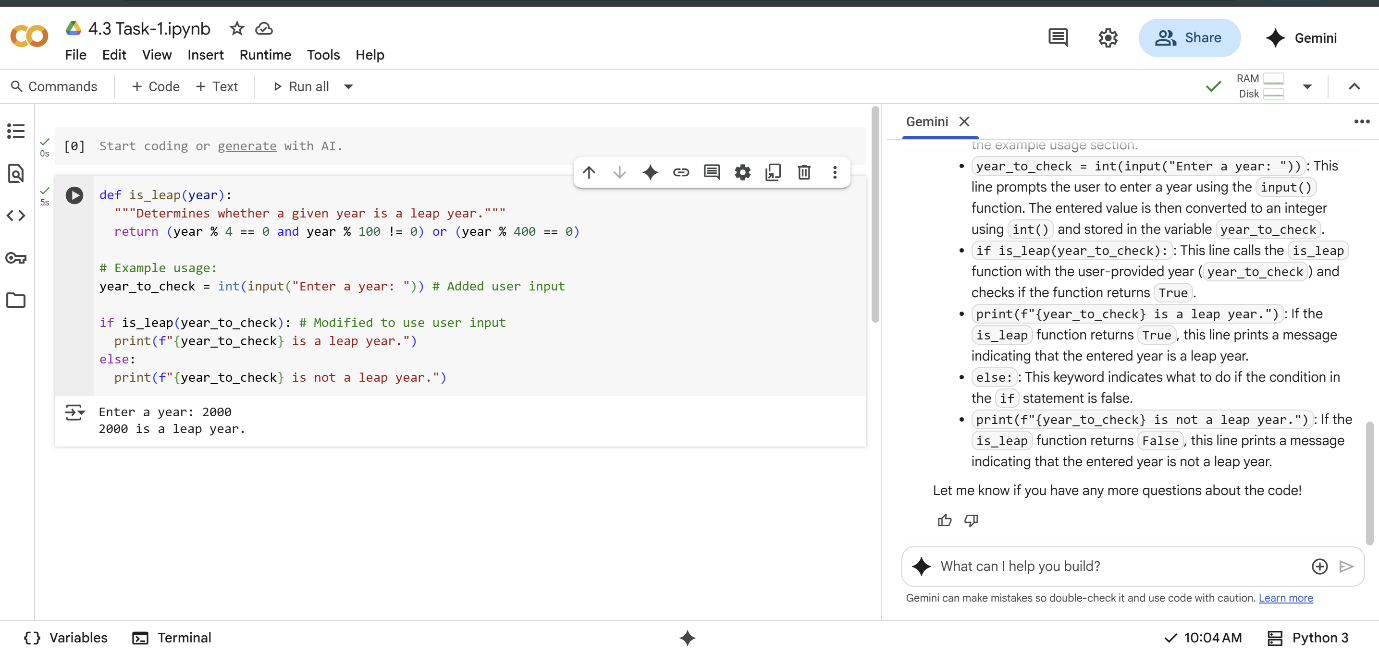
HTNO-2403A52134

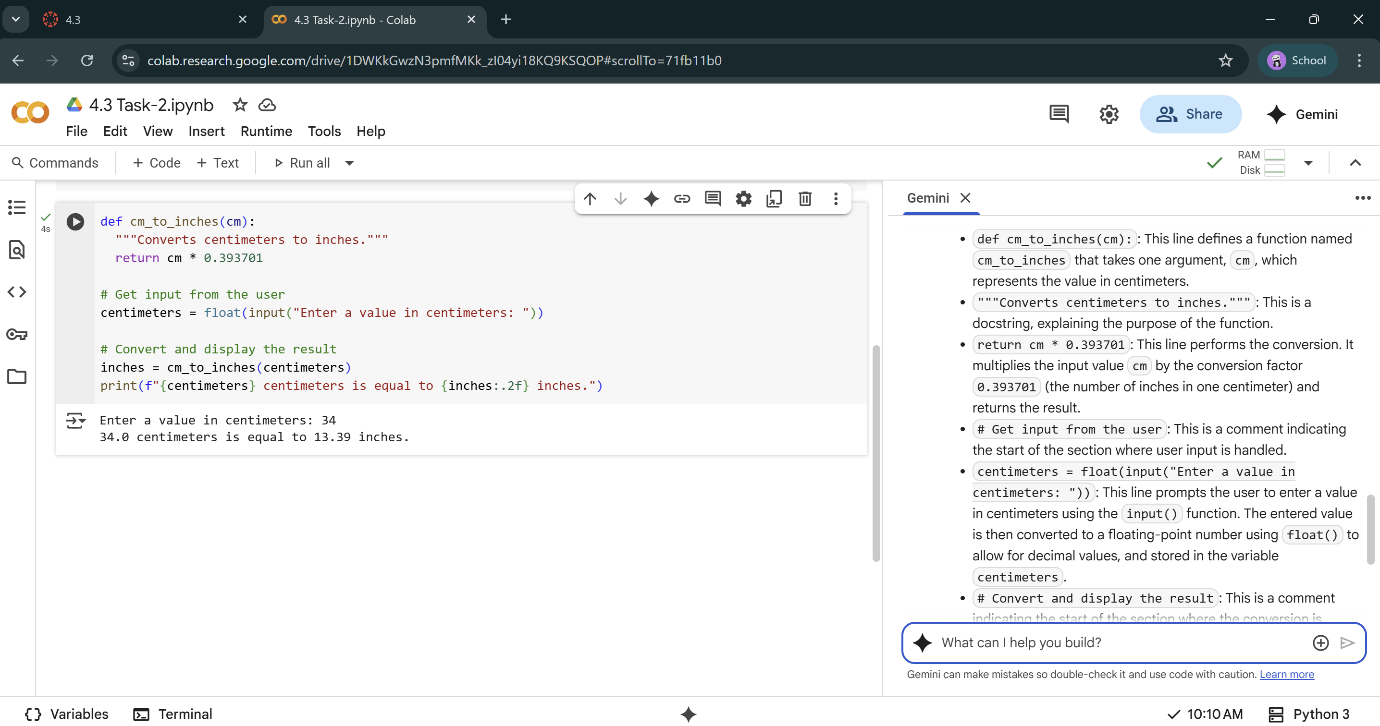
Assisgment-4.3

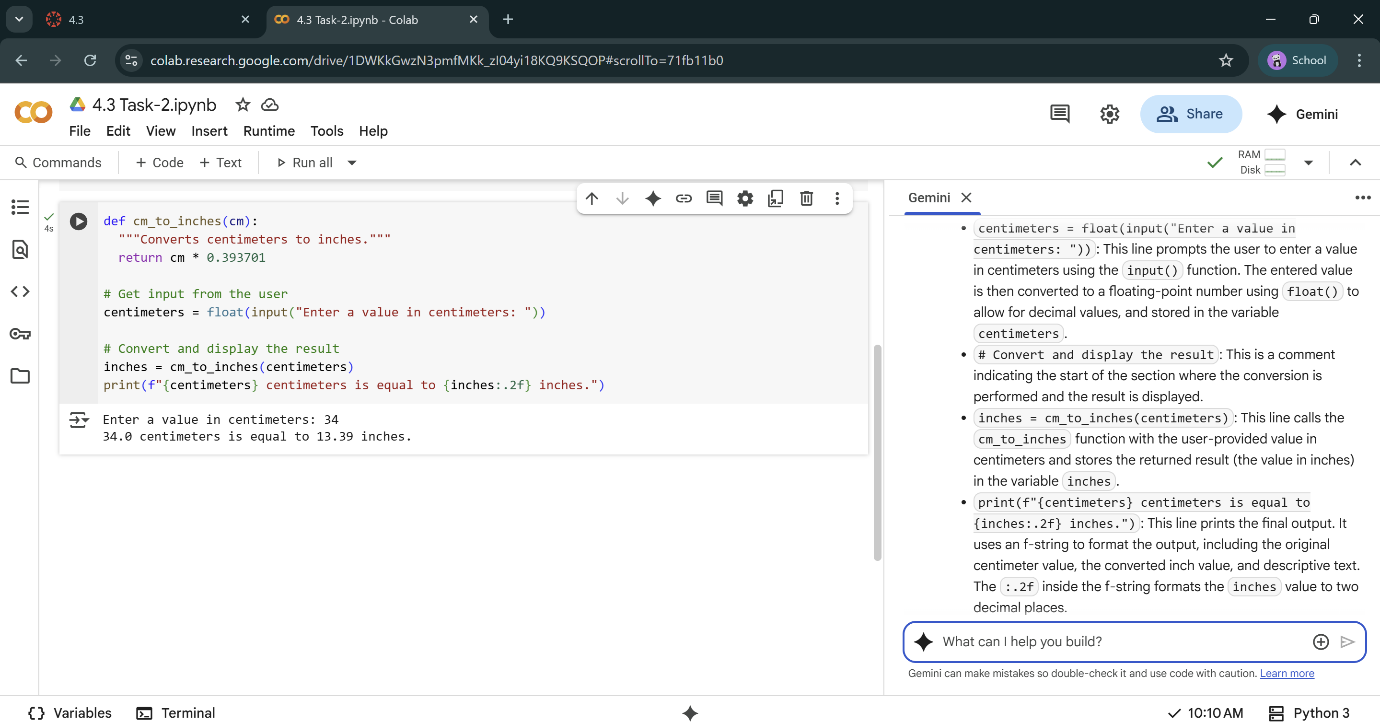
Task-1:





Task-2:



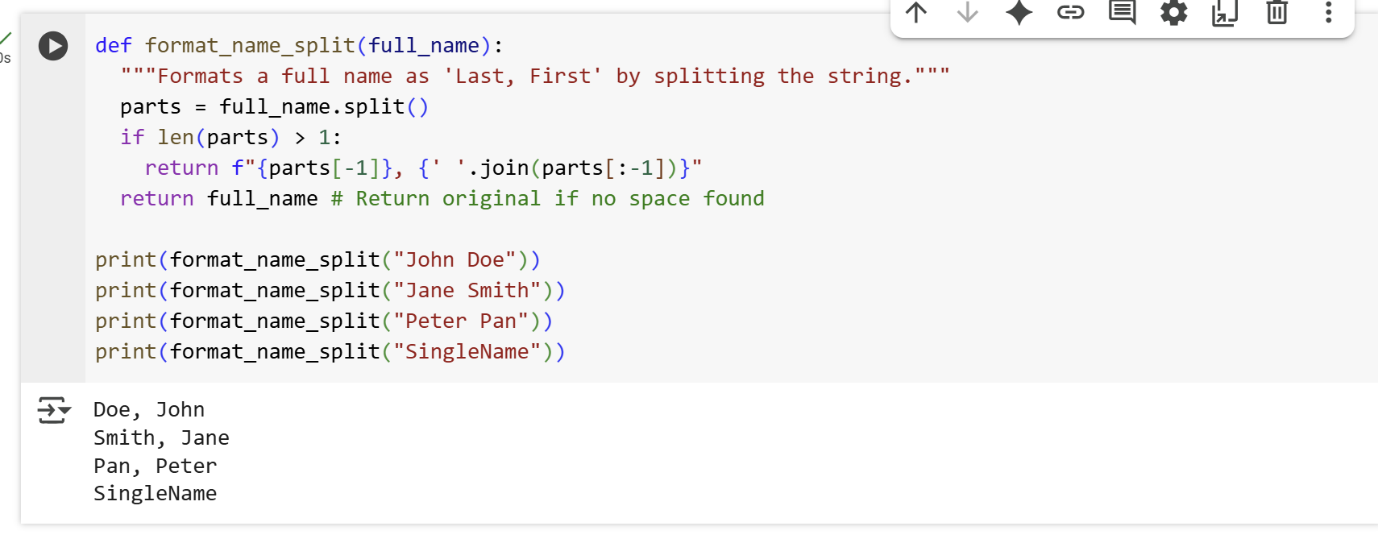


Task-3:

Code:







Explanation:

**Cell 2816e80f (using split)**

This function takes a full name as input and formats it as "Last, First".

* def format\_name\_split(full\_name):: Defines a function named format\_name\_split that accepts one argument, full\_name.
* """Formats a full name as 'Last, First' by splitting the string.""": This is a docstring, explaining what the function does.
* parts = full\_name.split(): This line splits the input full\_name string into a list of words based on spaces and assigns it to the variable parts.
* if len(parts) > 1:: This checks if the list parts has more than one element (meaning there's at least a first and last name).
* return f"{parts[-1]}, {' '.join(parts[:-1])}": If the condition is true, it formats the name. parts[-1] gets the last element (the last name). parts[:-1] gets all elements except the last one (the first name and any middle names). ' '.join(...) joins those elements back into a single string with spaces in between. The f-string then combines the last name, a comma and space, and the joined first name(s).
* return full\_name # Return original if no space found: If the if condition is false (the name has only one part), the original full\_name is returned.
* The print() statements demonstrate how to use the function with different names.

**Cell abba450f (using regex)**

This function uses regular expressions to format the name.

* def format\_name\_regex(full\_name):: Defines a function named format\_name\_regex that accepts one argument, full\_name.
* """Formats a full name as 'Last, First' using regular expressions.""": The docstring for this function.
* import re: Imports the regular expression module in Python.
* match = re.match(r"^(.\*?)\s+(\S+)$", full\_name): This is the core of the regex approach.
  + re.match(): Attempts to match a pattern at the beginning of the string.
  + r"^(.\*?)\s+(\S+)$": This is the regular expression pattern.

: Matches the beginning of the string.

* + - (.\*?): This is the first capturing group. . matches any character, \* matches zero or more occurrences, and ? makes it non-greedy (matching as few characters as possible). It captures the first part of the name.
    - \s+: Matches one or more whitespace characters.
    - (\S+): This is the second capturing group. \S matches any non-whitespace character, and + matches one or more occurrences. It captures the last part of the name.
    - $: Matches the end of the string.
  + The result of the match is stored in the match variable.
* if match:: Checks if the regex pattern found a match.
* first\_name = match.group(1): If a match is found, match.group(1) retrieves the content of the first capturing group (the first name).
* last\_name = match.group(2): match.group(2) retrieves the content of the second capturing group (the last name).
* return f"{last\_name}, {first\_name}": Formats and returns the name in "Last, First" format using the captured groups.
* return full\_name # Return original if regex doesn't match: If the regex doesn't match (e.g., a single name), the original full\_name is returned.
* The print() statements show the function in action.

**Cell 87f02afe (using reverse)**

This function formats the name by reversing the order of the parts after splitting.

* def format\_name\_reverse(full\_name):: Defines a function named format\_name\_reverse that accepts one argument, full\_name.
* """Formats a full name as 'Last, First' by reversing parts after splitting.""": The docstring for this function.
* parts = full\_name.split(): Splits the input full\_name into a list of words based on spaces.
* if len(parts) > 1:: Checks if the list parts has more than one element.
* parts.reverse(): Reverses the order of elements in the parts list in place.
* last\_name = parts[0]: After reversing, the first element in the list is now the original last name.
* first\_name = ' '.join(parts[1:]): Joins all elements from the second element onwards (which were originally the first and any middle names) back into a string with spaces.
* return f"{last\_name}, {first\_name}": Formats and returns the name in "Last, First" format.
* return full\_name: If the name has only one part, the original full\_name is returned.
* The print() statements demonstrate the function's usage.

Task-4:

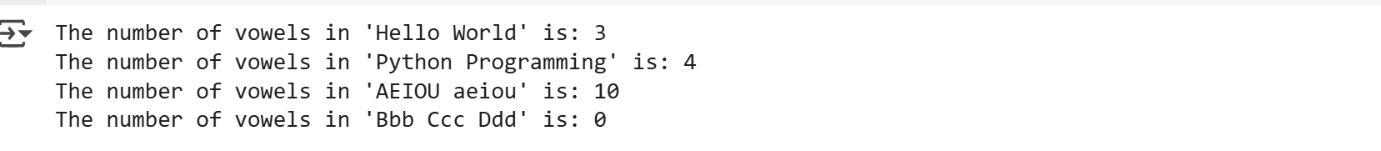
\*\*Comparison:\*\*

\* \*\*Zero-shot:\*\* Relies purely on the model's internal knowledge. Might be sufficient for simple, common tasks. Less overhead in prompt creation.

\* \*\*Few-shot:\*\* Provides explicit examples, which can be helpful for more complex or specific tasks, reducing ambiguity and improving the likelihood of the desired output format or logic. Requires more effort in crafting the prompt with relevant examples.

For a task like counting vowels, a zero-shot prompt is likely to be sufficient for most modern language models. However, for tasks requiring a specific output format, handling edge cases, or involving less common programming patterns, few-shot prompting can be beneficial.

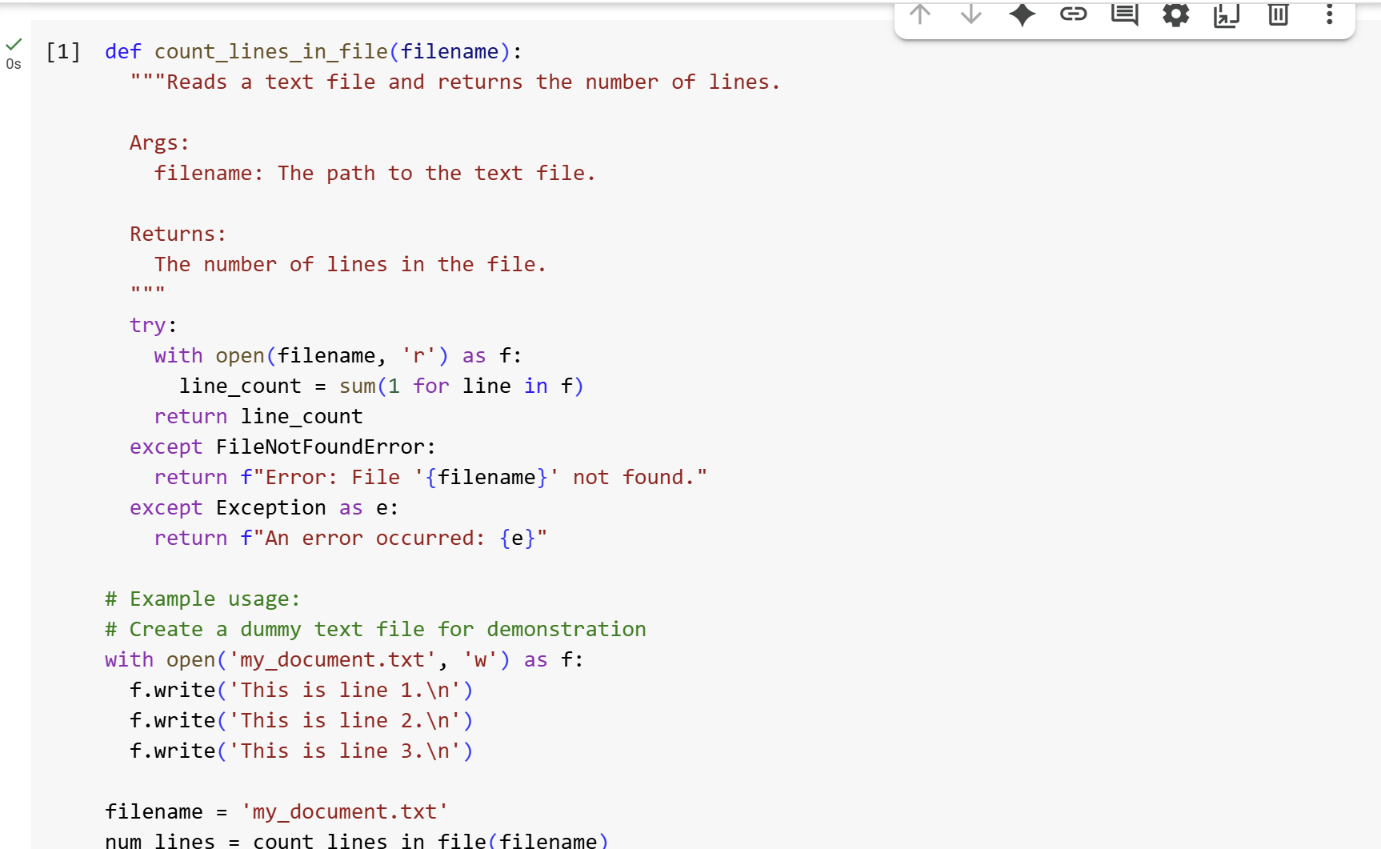
Code:

Output:

Explanation:

* def count\_vowels(text):: This line defines a function named count\_vowels that takes one argument, text, which is expected to be a string.
* """Counts the number of vowels in a given string.""": This is a docstring. It explains in plain language what the function does.
* vowels = "aeiouAEIOU": This line creates a string variable named vowels and assigns it 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 named count to 0. This variable will keep track of the number of vowels found.
* for char in text:: This line starts 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.
* if char in vowels:: Inside the loop, this line checks if the current character char is present in the vowels string. The in operator checks for membership.
* count += 1: If the if condition is true (the character is a vowel), this line increments the count variable by 1.
* return count: After the loop has finished iterating through all characters in the text, this line returns the final value of the count variable, which is the total number of vowels found.
* # Example usage:: This is a comment indicating the start of the example usage section.
* text = "Hello World": This line assigns the string "Hello World" to the text variable.
* vowel\_count = count\_vowels(text): This line calls the count\_vowels function with the current text and stores the returned value (the vowel count) in the vowel\_count variable.
* print(f"The number of vowels in '{text}' is: {vowel\_count}"): This line prints the result using an f-string to include the original text and the calculated vowel count in the output.
* The following lines repeat the process for different input strings ("Python Programming", "AEIOU aeiou", and "Bbb Ccc Ddd") to demonstrate the function's behavior with various inputs, including all vowels and no vowels.

Task-5:

Code and Output:



Explanation:

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.  
  """

This block defines a function called count\_lines\_in\_file that takes one argument, filename. The triple-quoted string is a docstring, which explains what the function does, its arguments (Args), and what it returns (Returns).

  try:  
    with open(filename, 'r') as f:  
      line\_count = sum(1 for line in f)  
    return line\_count

This is a try block, used for error handling.

* with open(filename, 'r') as f: opens the file specified by filename in read mode ('r'). The with statement ensures 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) is a concise way to count 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 all these 1s, effectively counting the lines.
* return line\_count returns the calculated number of lines.

  except FileNotFoundError:  
    return f"Error: File '{filename}' not found."

This except block catches a FileNotFoundError if the specified file does not exist. It returns an informative error message.

  except Exception as e:  
    return f"An error occurred: {e}"

This except block is a general catch-all for any other exceptions that might occur during file processing. It returns a message indicating that an error occurred and includes the error details (e).

# Example usage:  
# Create a dummy text file for demonstration  
with open('my\_document.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')

This section demonstrates how to use the function.

* # Example usage: and # Create a dummy text file for demonstration are comments explaining the purpose of the following code.
* with open('my\_document.txt', 'w') as f: creates a file named my\_document.txt in write mode ('w'). If the file already exists, its contents will be overwritten.
* f.write('This is line 1.\n'), f.write('This is line 2.\n'), and f.write('This is line 3.\n') write three lines of text to the file, each followed by a newline character (\n).

filename = 'my\_document.txt'  
num\_lines = count\_lines\_in\_file(filename)  
print(f"The file '{filename}' has {num\_lines} lines.")

This final section calls the function and prints the result.

* filename = 'my\_document.txt' sets the filename variable to the name of the file to be processed.
* num\_lines = count\_lines\_in\_file(filename) calls the count\_lines\_in\_file function with the specified filename and stores the returned value (the number of lines) in the num\_lines variable.
* print(f"The file '{filename}' has {num\_lines} lines.") prints an f-string (formatted string literal) to the console, displaying the filename and the number of lines counted.