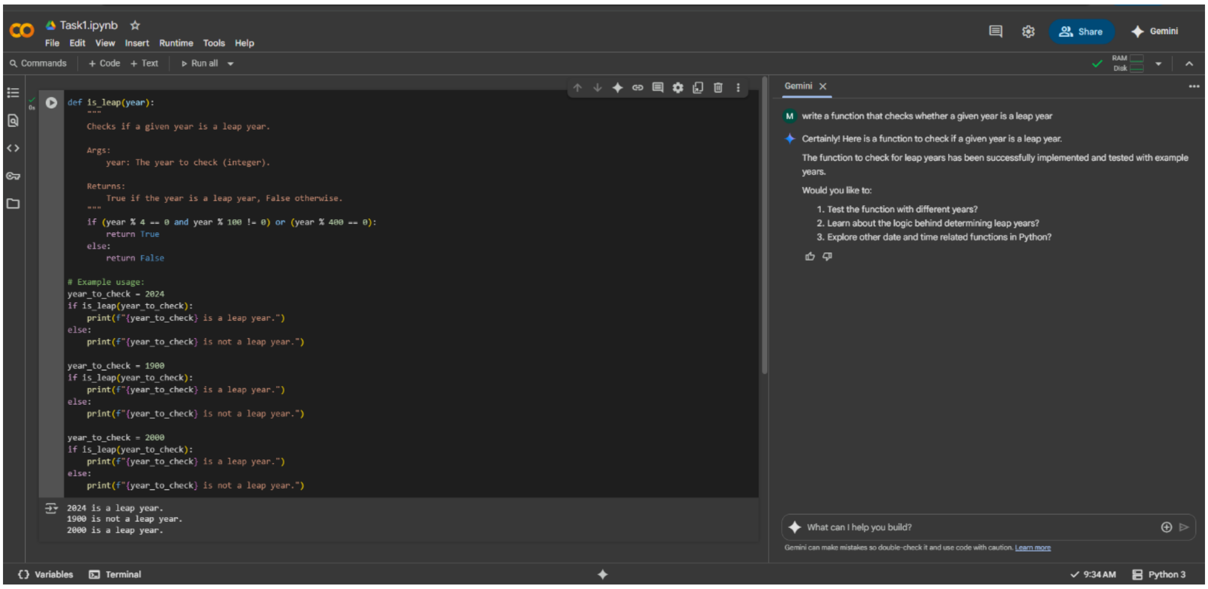
# AI assignment4:

## Task 1:

Zero-shot: Prompt AI to write a function that checks whether a given year is a leapyear.



```python
def is_leap(year):
    """
    Checks if a given year is a leap year.

    Args:
        year: The year to check (integer).

    Returns:
        True if the year is a leap year, False otherwise.
    """
    if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
        return True
    else:
        return False

# 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.")

year_to_check = 2000
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.")
```

```
2024 is a leap year.
1900 is not a leap year.
2000 is a leap year.
```

Gemini

M write a function that checks whether a given year is a leap year

Certainly! Here is a function to check if a given year is a leap year.

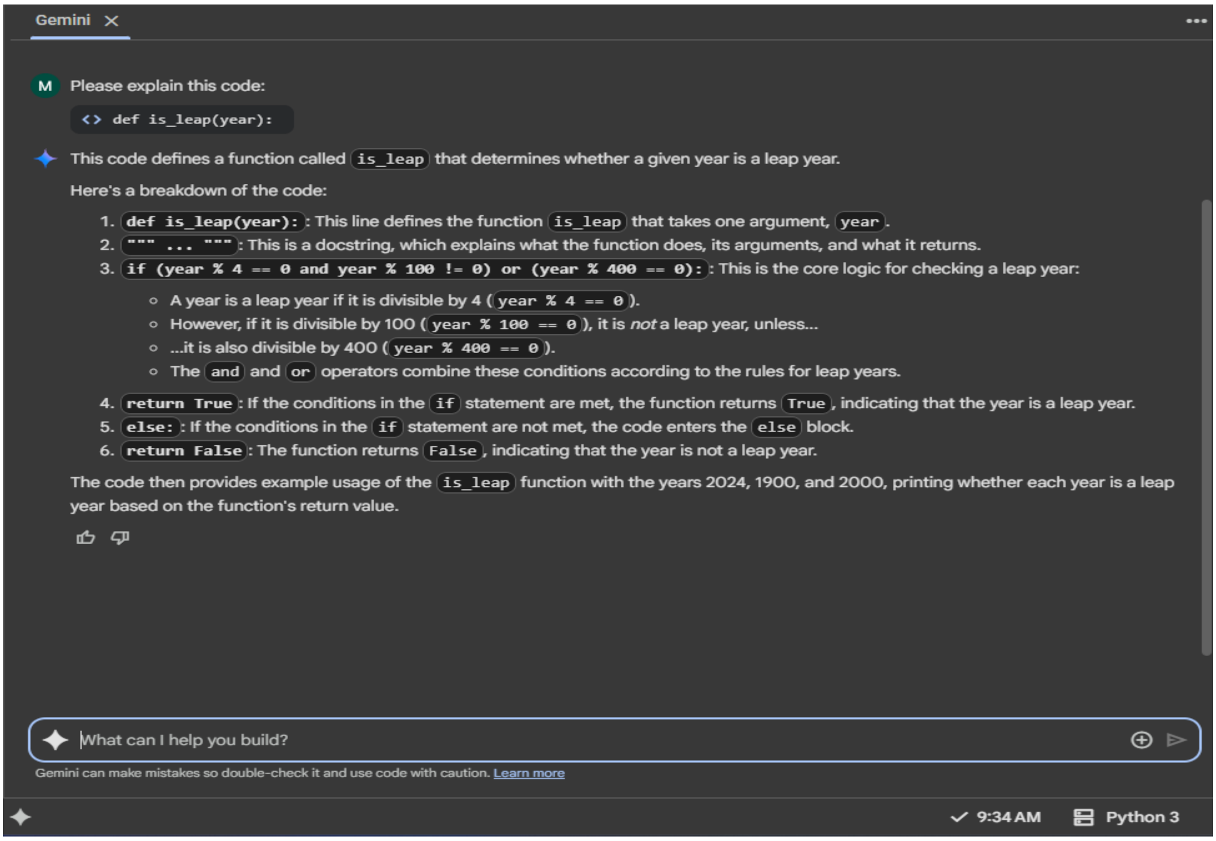
The function to check for leap years has been successfully implemented and tested with example years.

Would you like to:

1. Test the function with different years?
2. Learn about the logic behind determining leap years?
3. Explore other date and time related functions in Python?

What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. Learn more



M Please explain this code:

`<> def is_leap(year):`

This code defines a function called `is_leap` that determines whether a given year is a leap year.

Here's a breakdown of the code:

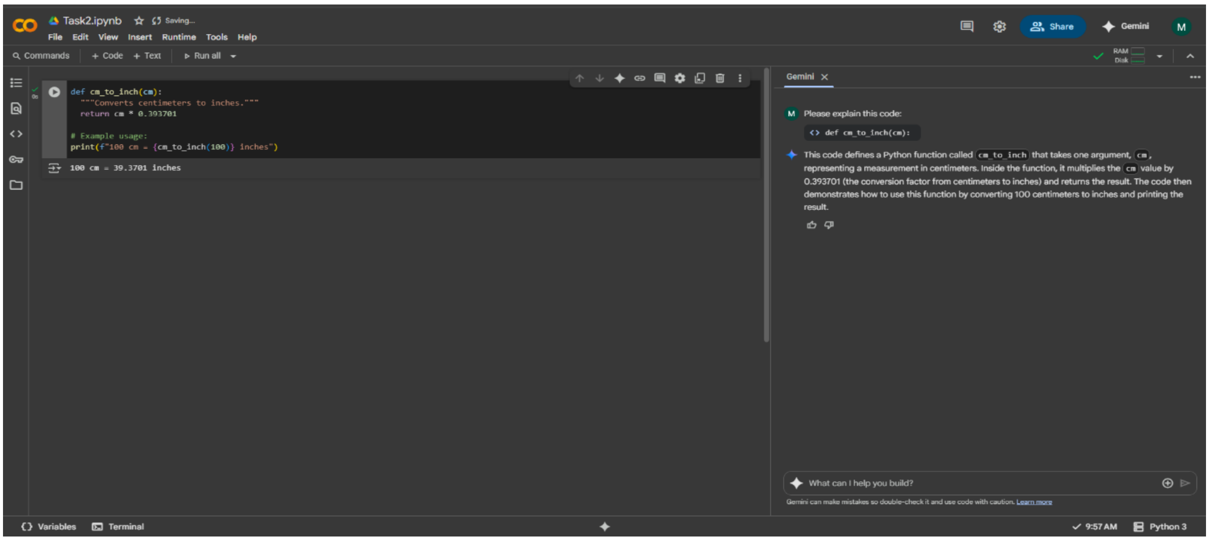
1. `def is_leap(year):`: This line defines the function `is_leap` that takes one argument, `year`.
2. `""" ... """`: This is a docstring, which explains what the function does, its arguments, and what it returns.
3. `if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):`: This is the core logic for checking a leap year:
    - A year is a leap year if it is divisible by 4 (`year % 4 == 0`).
    - However, if it is divisible by 100 (`year % 100 == 0`), it is *not* a leap year, unless...
    - ...it is also divisible by 400 (`year % 400 == 0`).
    - The `and` and `or` operators combine these conditions according to the rules for leap years.
4. `return True`: If the conditions in the `if` statement are met, the function returns `True`, indicating that the year is a leap year.
5. `else:`: If the conditions in the `if` statement are not met, the code enters the `else` block.
6. `return False`: The function returns `False`, indicating that the year is not a leap year.

The code then provides example usage of the `is_leap` function with the years 2024, 1900, and 2000, printing whether each year is a leap year based on the function's return value.

Gemini can make mistakes so double-check it and use code with caution. Learn more

Task 2:

One-shot: Give one input-output example to guide AI in writing a function that converts centimeters to inches.



```python
def cm_to_inch(cm):
  """Converts centimeters to inches."""
  return cm * 0.393701

# Example usage:
print(f"100 cm = {cm_to_inch(100)} inches")
```

```
100 cm = 39.3701 inches
```

Gemini

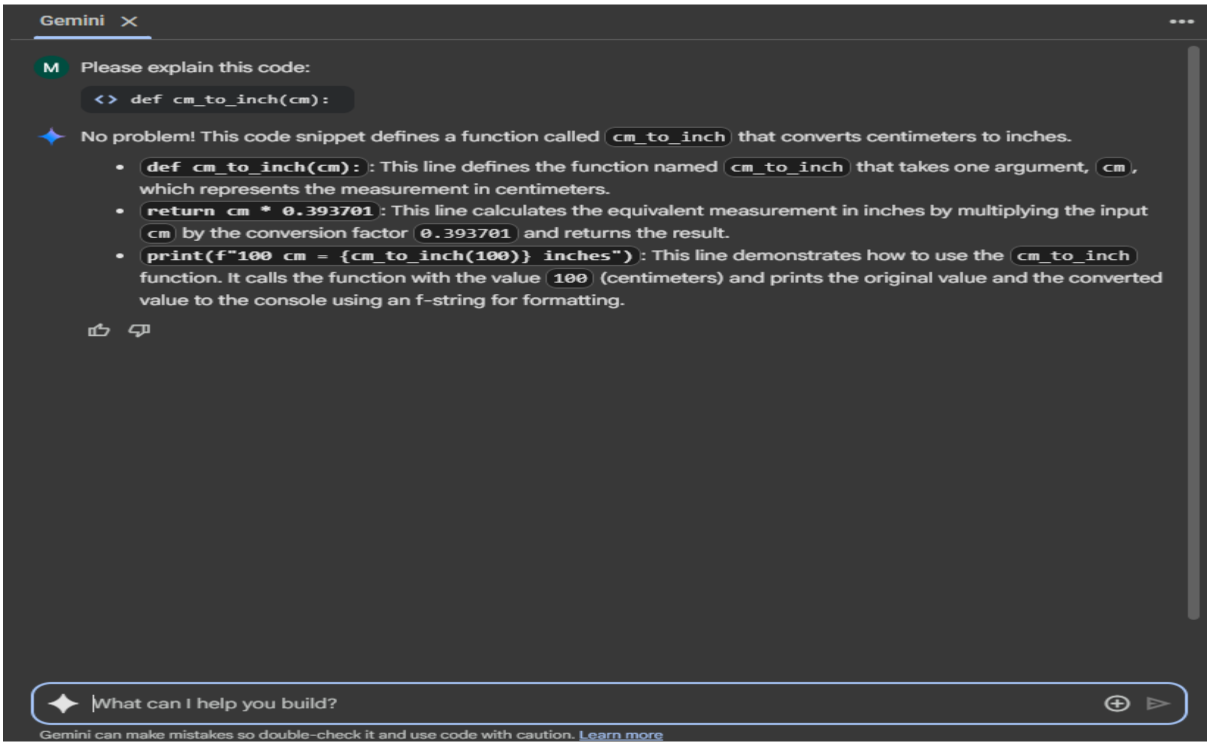
Please explain this code:

`def cm_to_inch(cm):`

This code defines a Python function called `cm_to_inch` that takes one argument, `cm`, representing a measurement in centimeters. Inside the function, it multiplies the `cm` value by 0.393701 (the conversion factor from centimeters to inches) and returns the result. The code then demonstrates how to use this function by converting 100 centimeters to inches and printing the result.

What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. Learn more



Please explain this code:

`def cm_to_inch(cm):`

No problem! This code snippet defines a function called `cm_to_inch` that converts centimeters to inches.

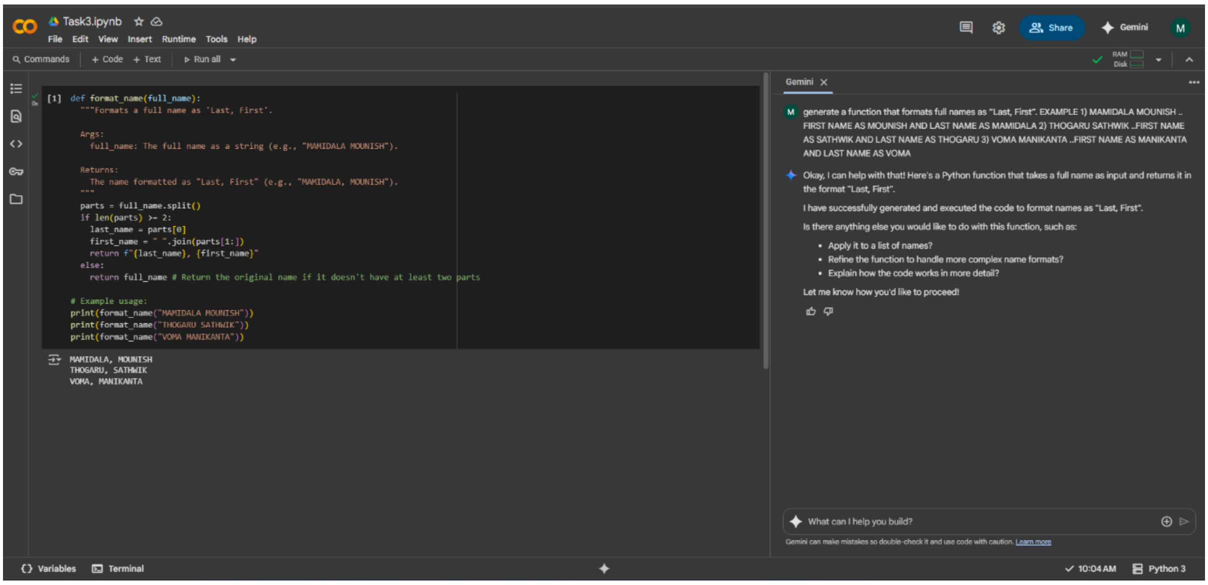
- `def cm_to_inch(cm):`: This line defines the function named `cm_to_inch` that takes one argument, `cm`, which represents the measurement in centimeters.
- `return cm * 0.393701`: This line calculates the equivalent measurement in inches by multiplying the input `cm` by the conversion factor `0.393701` and returns the result.
- `print(f"100 cm = {cm_to_inch(100)} inches")`: This line demonstrates how to use the `cm_to_inch` function. It calls the function with the value `100` (centimeters) and prints the original value and the converted value to the console using an f-string for formatting.

What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. Learn more

# Task :3

Few-shot: Provide 2–3 examples to generate a function that formats full names as “Last, First”


Task3.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
Gemini
RAM
Disk
Variables
Terminal
10:04 AM
Python 3

```
[1] def format_name(full_name):
      """Formats a full name as 'Last, First'.

      Args:
        full_name: The full name as a string (e.g., "MAMIDALA MOUNISH").

      Returns:
        The name formatted as "Last, First" (e.g., "MAMIDALA, MOUNISH").
      """
      parts = full_name.split()
      if len(parts) >= 2:
        last_name = parts[0]
        first_name = " ".join(parts[1:])
        return f"{last_name}, {first_name}"
      else:
        return full_name # Return the original name if it doesn't have at least two parts

    # Example usage:
    print(format_name("MAMIDALA MOUNISH"))
    print(format_name("THOGARU SATHWIK"))
    print(format_name("VOMA MANIKANTA"))
```

```
MAMIDALA, MOUNISH
THOGARU, SATHWIK
VOMA, MANIKANTA
```

Gemini

generate a function that formats full names as “Last, First”. EXAMPLE 1) MAMIDALA MOUNISH .. FIRST NAME AS MOUNISH AND LAST NAME AS MAMIDALA 2) THOGARU SATHWIK ..FIRST NAME AS SATHWIK AND LAST NAME AS THOGARU 3) VOMA MANIKANTA ..FIRST NAME AS MANIKANTA AND LAST NAME AS VOMA

Okay, I can help with that! Here's a Python function that takes a full name as input and returns it in the format "Last, First".

I have successfully generated and executed the code to format names as "Last, First".

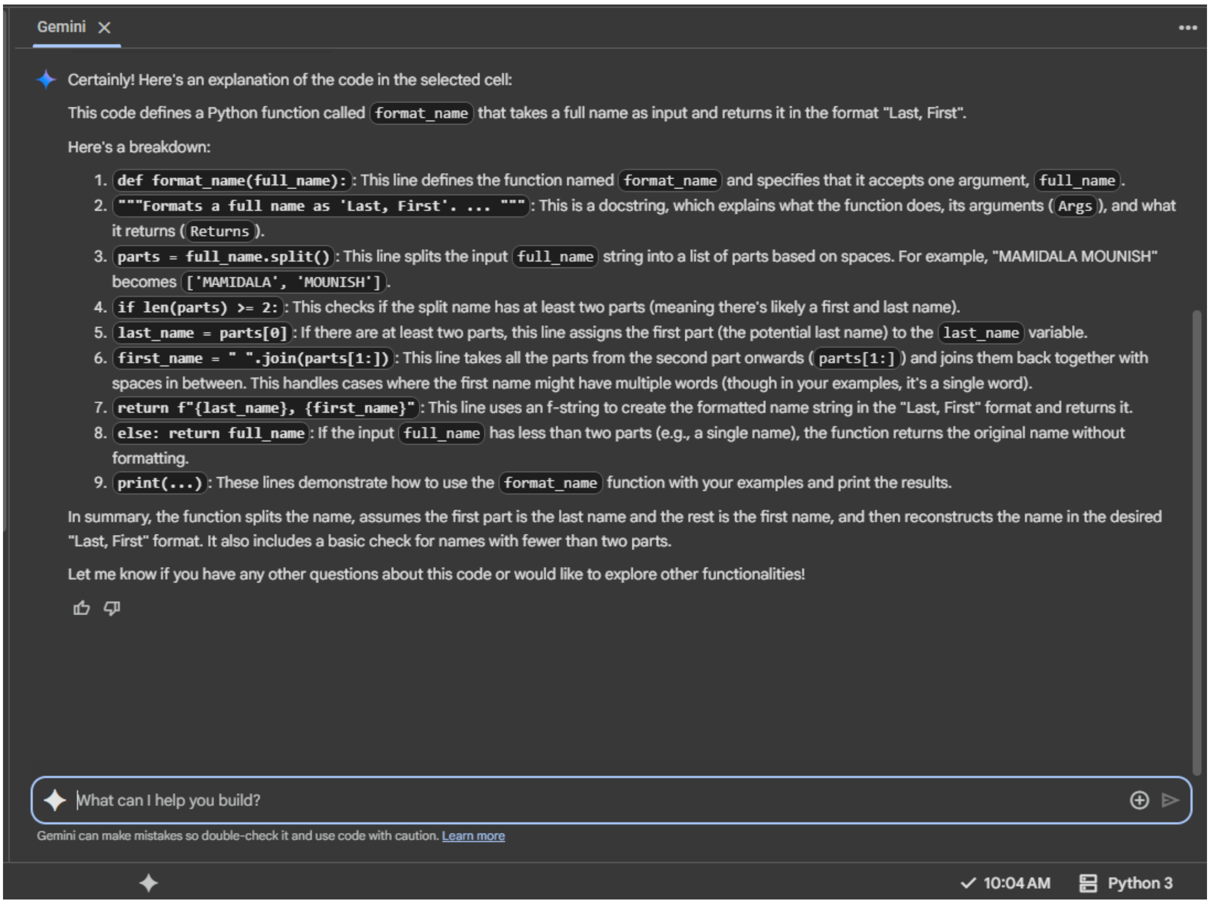
Is there anything else you would like to do with this function, such as:

- Apply it to a list of names?
- Refine the function to handle more complex name formats?
- Explain how the code works in more detail?

Let me know how you'd like to proceed!

What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. Learn more



Certainly! Here's an explanation of the code in the selected cell:

This code defines a Python function called `format_name` that takes a full name as input and returns it in the format "Last, First".

Here's a breakdown:

1. `def format_name(full_name):`: This line defines the function named `format_name` and specifies that it accepts one argument, `full_name`.
2. `"""Formats a full name as 'Last, First'. ... """`: This is a docstring, which explains what the function does, its arguments (`Args`), and what it returns (`Returns`).
3. `parts = full_name.split()`: This line splits the input `full_name` string into a list of parts based on spaces. For example, "MAMIDALA MOUNISH" becomes `['MAMIDALA', 'MOUNISH']`.
4. `if len(parts) >= 2:`: This checks if the split name has at least two parts (meaning there's likely a first and last name).
5. `last_name = parts[0]`: If there are at least two parts, this line assigns the first part (the potential last name) to the `last_name` variable.
6. `first_name = " ".join(parts[1:])`: This line takes all the parts from the second part onwards (`parts[1:]`) and joins them back together with spaces in between. This handles cases where the first name might have multiple words (though in your examples, it's a single word).
7. `return f"{last_name}, {first_name}"`: This line uses an f-string to create the formatted name string in the "Last, First" format and returns it.
8. `else: return full_name`: If the input `full_name` has less than two parts (e.g., a single name), the function returns the original name without formatting.
9. `print(...)`: These lines demonstrate how to use the `format_name` function with your examples and print the results.

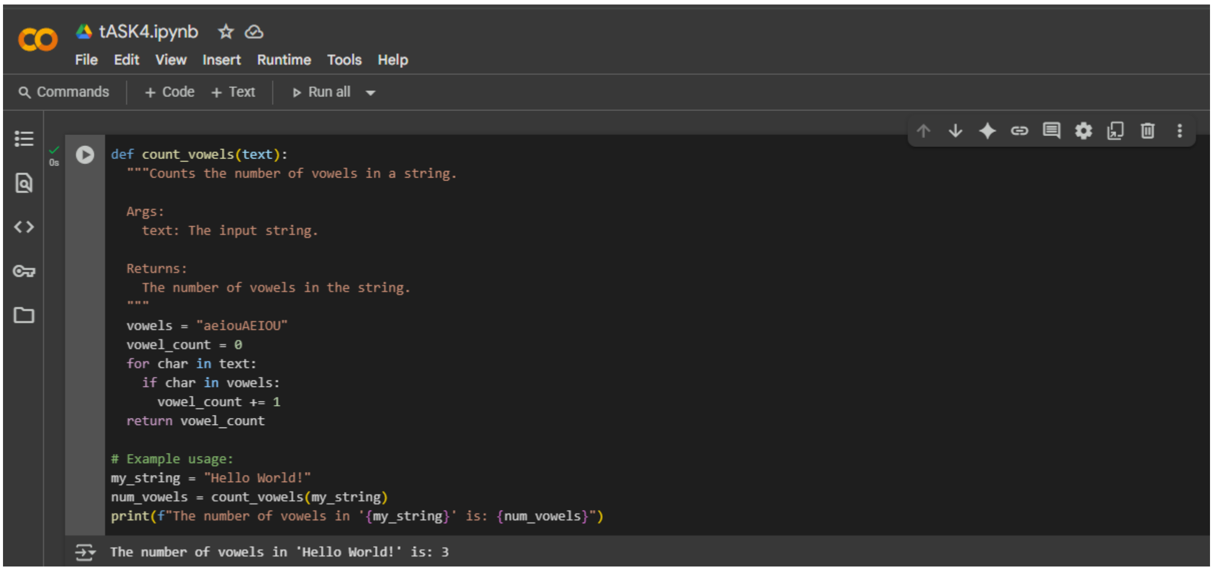
In summary, the function splits the name, assumes the first part is the last name and the rest is the first name, and then reconstructs the name in the desired "Last, First" format. It also includes a basic check for names with fewer than two parts.

Let me know if you have any other questions about this code or would like to explore other functionalities!

What can I help you build?

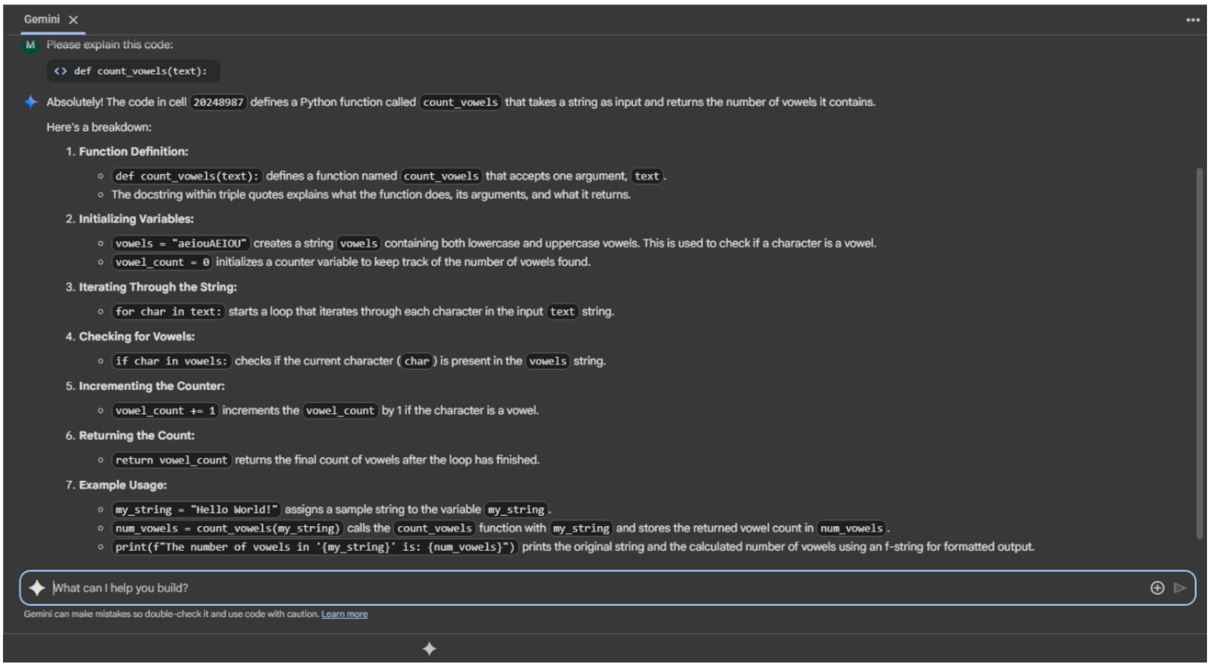
Gemini can make mistakes so double-check it and use code with caution. Learn more

Task Description#4
Compare zero-shot and few-shot prompts for writing a function that counts the number of vowels in a string.



```python
def count_vowels(text):
  """Counts the number of vowels in a 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 number of vowels in 'Hello World!' is: 3
```



Please explain this code:

`def count_vowels(text):`

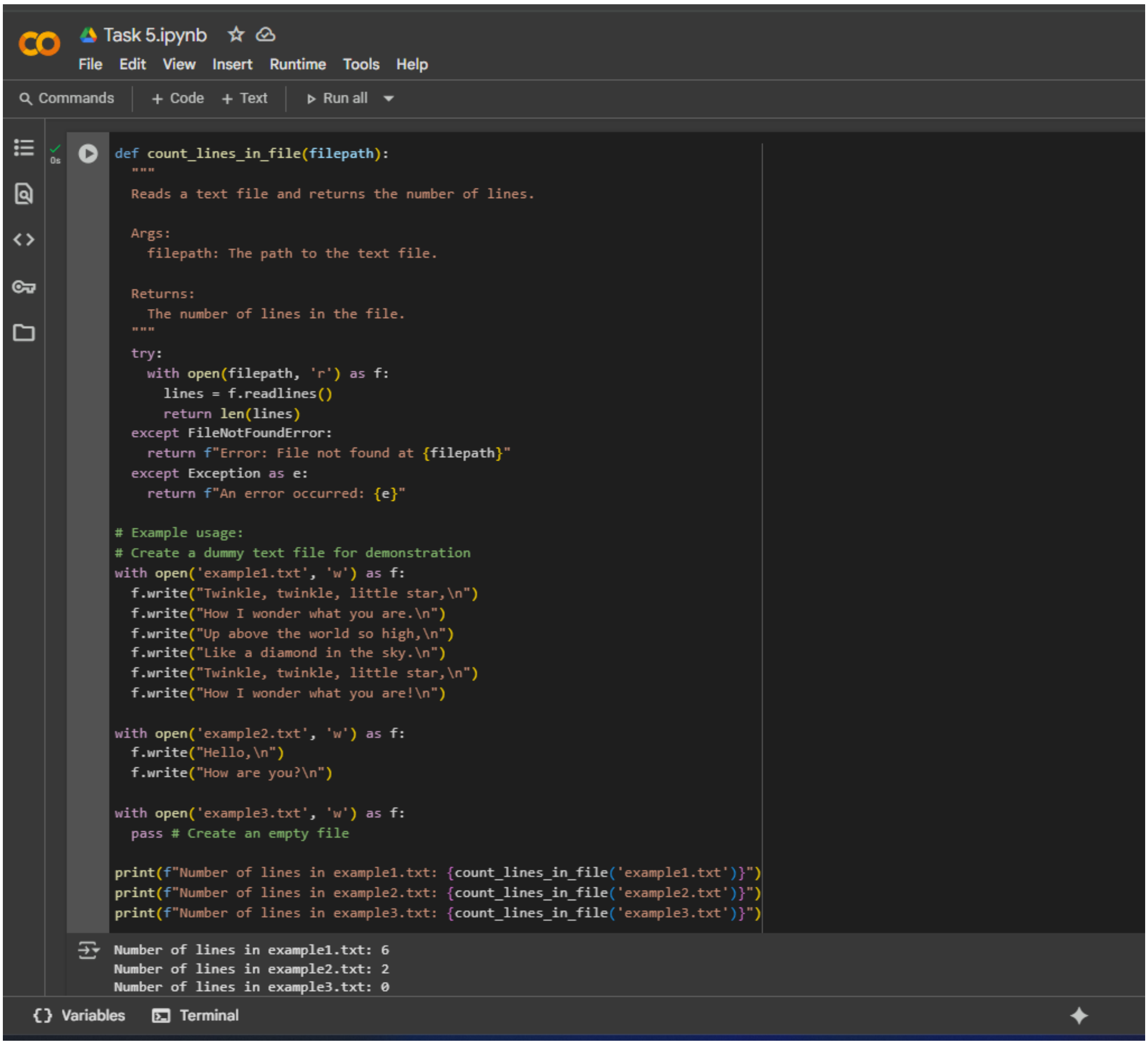
Absolutely! The code in cell `20248987` defines a Python function called `count_vowels` that takes a string as input and returns the number of vowels it contains.

Here's a breakdown:

1. **Function Definition:**
   - `def count_vowels(text):` defines a function named `count_vowels` that accepts one argument, `text`.
   - The docstring within triple quotes explains what the function does, its arguments, and what it returns.
2. **Initializing Variables:**
   - `vowels = "aeiouAEIOU"` creates a string `vowels` containing both lowercase and uppercase vowels. This is used to check if a character is a vowel.
   - `vowel_count = 0` initializes a counter variable to keep track of the number of vowels found.
3. **Iterating Through the String:**
   - `for char in text:` starts a loop that iterates through each character in the input `text` string.
4. **Checking for Vowels:**
   - `if char in vowels:` checks if the current character ( `char` ) is present in the `vowels` string.
5. **Incrementing the Counter:**
   - `vowel_count += 1` increments the `vowel_count` by 1 if the character is a vowel.
6. **Returning the Count:**
   - `return vowel_count` returns the final count of vowels after the loop has finished.
7. **Example Usage:**
   - `my_string = "Hello World!"` assigns a sample string to the variable `my_string`.
   - `num_vowels = count_vowels(my_string)` calls the `count_vowels` function with `my_string` and stores the returned vowel count in `num_vowels`.
   - `print(f"The number of vowels in '{my_string}' is: {num_vowels}")` prints the original string and the calculated number of vowels using an f-string for formatted output.

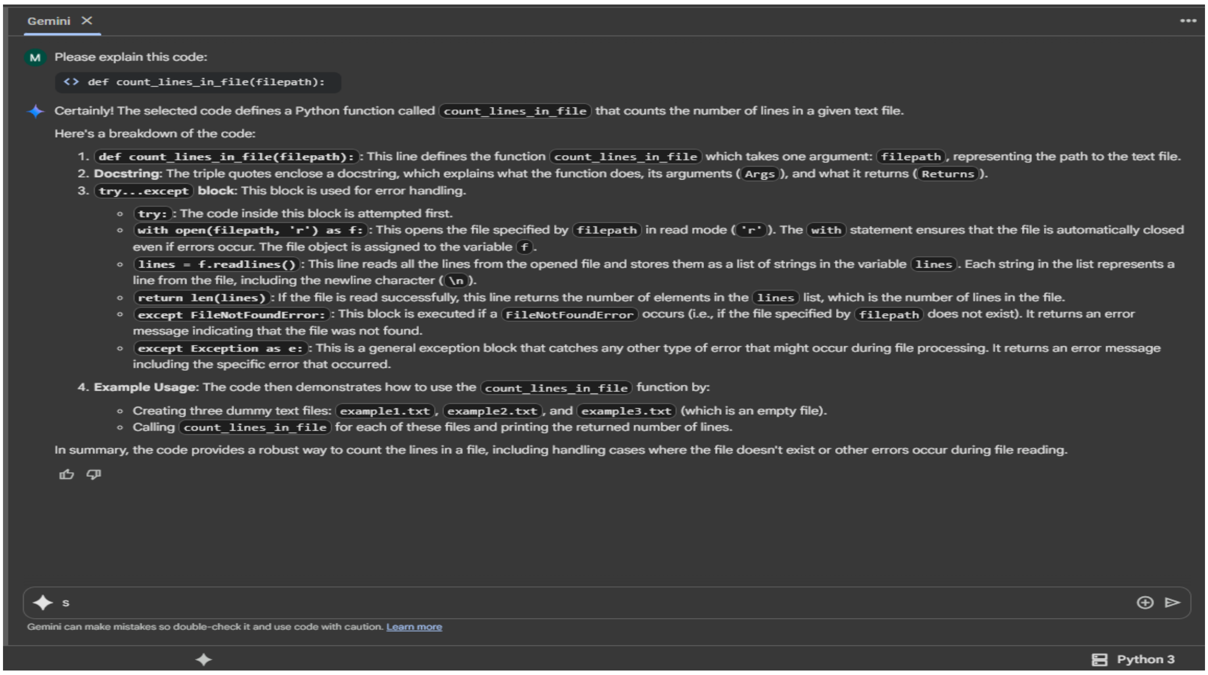
Gemini can make mistakes so double-check it and use code with caution. Learn more

Task Description#5
Use few-shot prompting to generate a function that reads a .txt file and returns the number of lines



```python
def count_lines_in_file(filepath):
  """
  Reads a text file and returns the number of lines.

  Args:
    filepath: The path to the text file.

  Returns:
    The number of lines in the file.
  """
  try:
    with open(filepath, 'r') as f:
      lines = f.readlines()
      return len(lines)
  except FileNotFoundError:
    return f"Error: File not found at {filepath}"
  except Exception as e:
    return f"An error occurred: {e}"

# Example usage:
# Create a dummy text file for demonstration
with open('example1.txt', 'w') as f:
  f.write("Twinkle, twinkle, little star,\n")
  f.write("How I wonder what you are.\n")
  f.write("Up above the world so high,\n")
  f.write("Like a diamond in the sky.\n")
  f.write("Twinkle, twinkle, little star,\n")
  f.write("How I wonder what you are!\n")

with open('example2.txt', 'w') as f:
  f.write("Hello,\n")
  f.write("How are you?\n")

with open('example3.txt', 'w') as f:
  pass # Create an empty file

print(f"Number of lines in example1.txt: {count_lines_in_file('example1.txt')}")
print(f"Number of lines in example2.txt: {count_lines_in_file('example2.txt')}")
print(f"Number of lines in example3.txt: {count_lines_in_file('example3.txt')}")
```

```
Number of lines in example1.txt: 6
Number of lines in example2.txt: 2
Number of lines in example3.txt: 0
```



Please explain this code:

`def count_lines_in_file(filepath):`

Certainly! The selected code defines a Python function called `count_lines_in_file` that counts the number of lines in a given text file.

Here's a breakdown of the code:

1. `def count_lines_in_file(filepath):`: This line defines the function `count_lines_in_file` which takes one argument: `filepath`, representing the path to the text file.
2. **Docstring**: The triple quotes enclose a docstring, which explains what the function does, its arguments (`Args`), and what it returns (`Returns`).
3. `try...except` **block**: This block is used for error handling.
   - `try:`: The code inside this block is attempted first.
   - `with open(filepath, 'r') as f:`: This opens the file specified by `filepath` in read mode (`'r'`). The `with` statement ensures that the file is automatically closed even if errors occur. The file object is assigned to the variable `f`.
   - `lines = f.readlines()`: This line reads all the lines from the opened file and stores them as a list of strings in the variable `lines`. Each string in the list represents a line from the file, including the newline character (`\n`).
   - `return len(lines)`: If the file is read successfully, this line returns the number of elements in the `lines` list, which is the number of lines in the file.
   - `except FileNotFoundError:`: This block is executed if a `FileNotFoundError` occurs (i.e., if the file specified by `filepath` does not exist). It returns an error message indicating that the file was not found.
   - `except Exception as e:`: This is a general exception block that catches any other type of error that might occur during file processing. It returns an error message including the specific error that occurred.
4. **Example Usage**: The code then demonstrates how to use the `count_lines_in_file` function by:
   - Creating three dummy text files: `example1.txt`, `example2.txt`, and `example3.txt` (which is an empty file).
   - Calling `count_lines_in_file` for each of these files and printing the returned number of lines.

In summary, the code provides a robust way to count the lines in a file, including handling cases where the file doesn't exist or other errors occur during file reading.

Gemini can make mistakes so double-check it and use code with caution. Learn more