LAB ASSIGNMNET 4.3

Subject : AI Assistant Coding

Hall Ticket No : 2303A52465

Name : M.Avinash

Batch : 50

Date : 28/1/26

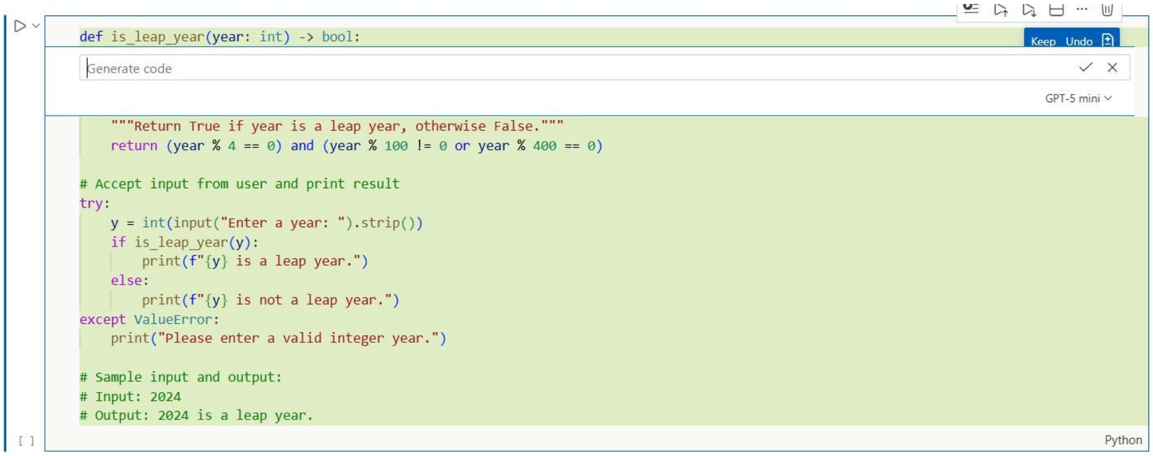
Task - 1 : Zero-Shot Prompting – Leap Year Check Scenario Zero-shot prompting involves giving instructions without providing examples.

Prompt : Generate a Python Code that accepts a year as input and checks whether the given year is a leap year.

The function should return an appropriate result indicating whether the year is a leap year or not.

Also include a sample input and its corresponding output.

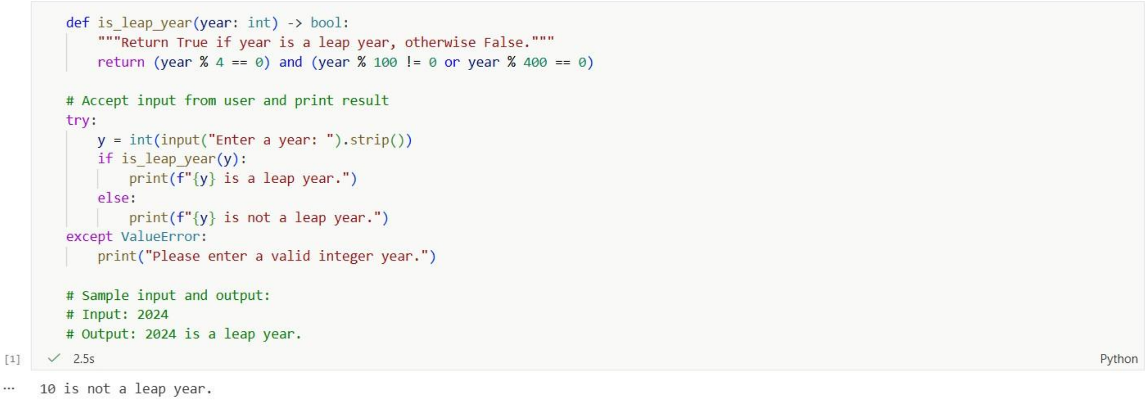
Screenshots:



```python
def is_leap_year(year: int) -> bool:
    """Return True if year is a leap year, otherwise False."""
    return (year % 4 == 0) and (year % 100 != 0 or year % 400 == 0)

# Accept input from user and print result
try:
    y = int(input("Enter a year: ").strip())
    if is_leap_year(y):
        print(f"{y} is a leap year.")
    else:
        print(f"{y} is not a leap year.")
except ValueError:
    print("Please enter a valid integer year.")

# Sample input and output:
# Input: 2024
# Output: 2024 is a leap year.
```

Input : 10

Output:



```
def is_leap_year(year: int) -> bool:
    """Return True if year is a leap year, otherwise False."""
    return (year % 4 == 0) and (year % 100 != 0 or year % 400 == 0)

# Accept input from user and print result
try:
    y = int(input("Enter a year: ").strip())
    if is_leap_year(y):
        print(f"{y} is a leap year.")
    else:
        print(f"{y} is not a leap year.")
except ValueError:
    print("Please enter a valid integer year.")

# Sample input and output:
# Input: 2024
# Output: 2024 is a leap year.
```

```
10 is not a leap year.
```

<u>Task – 2</u> : One-Shot Prompting – Centimeters to Inches Conversion

Scenario

One-shot prompting guides AI using a single example.

<u>Prompt</u> :Convert centimeters to inches using Python.

Example:

Input: 10 cm

Output: 3.94 inches

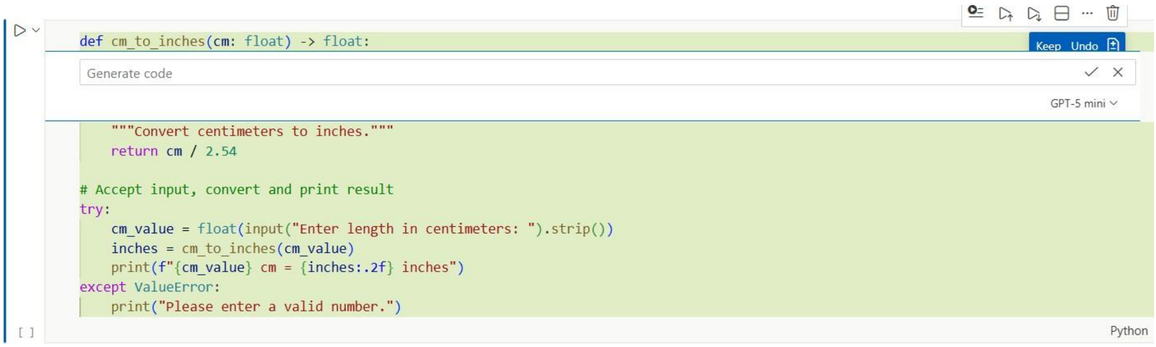
Now generate a Python code that:

- Accepts a value in centimeters as input

- Converts it to inches using the correct formula

- Returns the converted value

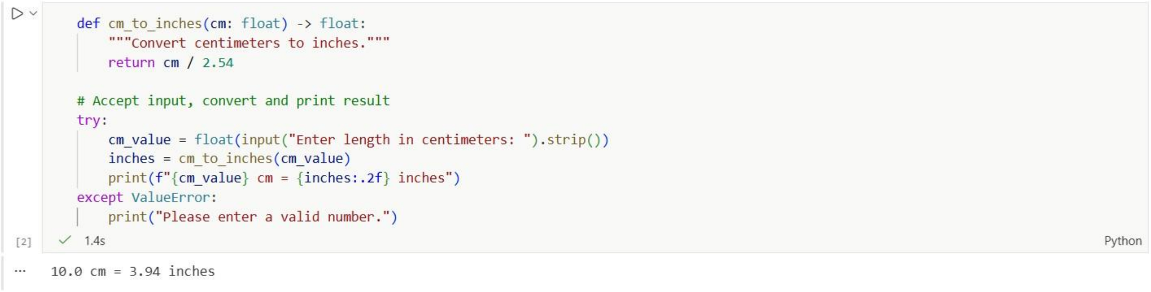
ScreenShots :



```
def cm_to_inches(cm: float) -> float:
    """Convert centimeters to inches."""
    return cm / 2.54

# Accept input, convert and print result
try:
    cm_value = float(input("Enter length in centimeters: ").strip())
    inches = cm_to_inches(cm_value)
    print(f"{cm_value} cm = {inches:.2f} inches")
except ValueError:
    print("Please enter a valid number.")
```

Input : 10

Output :



```python
def cm_to_inches(cm: float) -> float:
    """Convert centimeters to inches."""
    return cm / 2.54

# Accept input, convert and print result
try:
    cm_value = float(input("Enter length in centimeters: ").strip())
    inches = cm_to_inches(cm_value)
    print(f"{cm_value} cm = {inches:.2f} inches")
except ValueError:
    print("Please enter a valid number.")
```

```
10.0 cm = 3.94 inches
```

Task-3 :Few-Shot Prompting – Name Formatting Scenario

Few-shot prompting improves accuracy by providing multiple examples.

Prompt : Format a person's full name into the format "Last, First" using Python.

Examples:

Input: John Smith

Output: Smith, John

Input: Anita Rao

Output: Rao, Anita

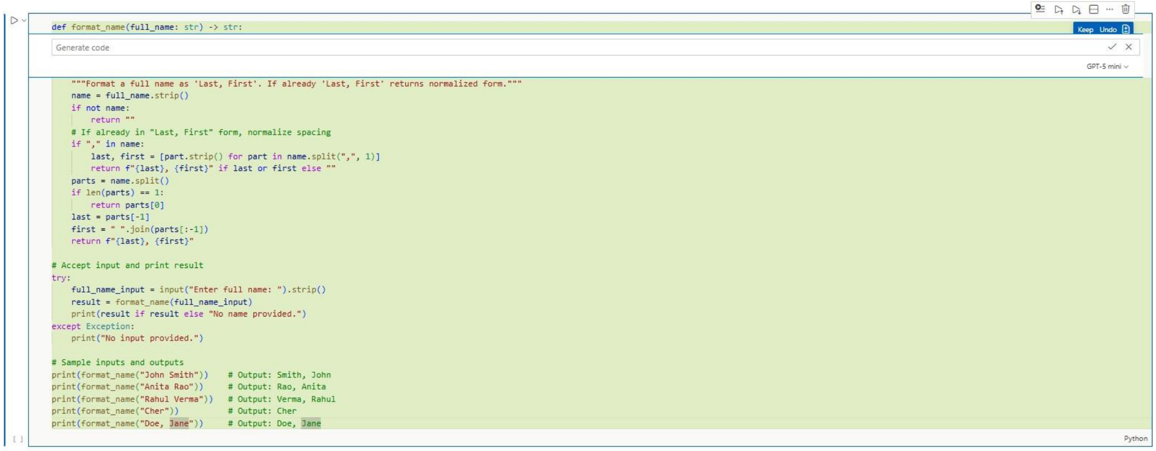
Input: Rahul Verma

Output: Verma, Rahul

Now generate a Python Code that:

- Accepts a full name as input
- Formats the name as "Last, First"
- Returns the formatted name
- Includes sample inputs and outputs

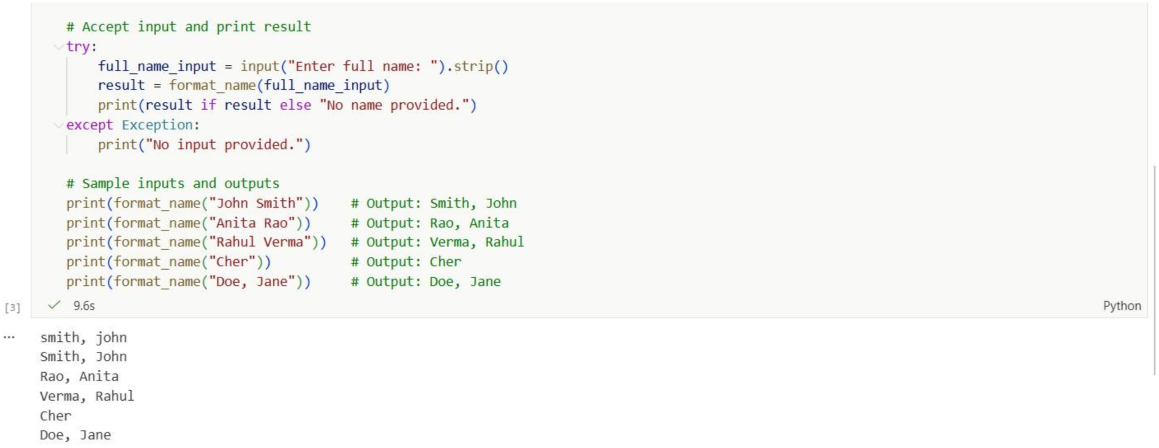
ScreenShots :



```
def format_name(full_name: str) -> str:
    """Format a full name as 'Last, First'. If already 'Last, First' returns normalized form."""
    name = full_name.strip()
    if not name:
        return ""
    # If already in "Last, First" form, normalize spacing
    if "," in name:
        last, first = [part.strip() for part in name.split(",", 1)]
        return f"{last}, {first}" if last or first else ""
    parts = name.split()
    if len(parts) == 1:
        return parts[0]
    last = parts[-1]
    first = " ".join(parts[:-1])
    return f"{last}, {first}"

# Accept input and print result
try:
    full_name_input = input("Enter full name: ").strip()
    result = format_name(full_name_input)
    print(result if result else "No name provided.")
except Exception:
    print("No input provided.")

# Sample inputs and outputs
print(format_name("John Smith"))    # Output: Smith, John
print(format_name("Anita Rao"))     # Output: Rao, Anita
print(format_name("Rahul Verma"))   # Output: Verma, Rahul
print(format_name("Cher"))          # Output: Cher
print(format_name("Doe, Jane"))     # Output: Doe, Jane
```

Input :John smith

Output :



```
# Accept input and print result
try:
    full_name_input = input("Enter full name: ").strip()
    result = format_name(full_name_input)
    print(result if result else "No name provided.")
except Exception:
    print("No input provided.")

# Sample inputs and outputs
print(format_name("John Smith"))    # Output: Smith, John
print(format_name("Anita Rao"))     # Output: Rao, Anita
print(format_name("Rahul Verma"))   # Output: Verma, Rahul
print(format_name("Cher"))          # Output: Cher
print(format_name("Doe, Jane"))     # Output: Doe, Jane
```

```
smith, john
Smith, John
Rao, Anita
Verma, Rahul
Cher
Doe, Jane
```

Task-4 : Comparative Analysis – Zero-Shot vs Few-Shot Scenario

Different prompt strategies may produce different code quality.

Prompt : 1. Generate a Python Code that accepts a string as input and counts the number of vowels in the string.
The function should return the total vowel count.
Also include a sample input and output.
2.Count the number of vowels in a string using Python.
Examples:
Input: "hello"
Output: 2

Input: "Education"

Output: 5

Input: "sky"

Output: 0

Now generate a Python Code that:

- Accepts a string as input
- Counts the number of vowels in the string
- Returns the total count
- Includes sample inputs and outputs

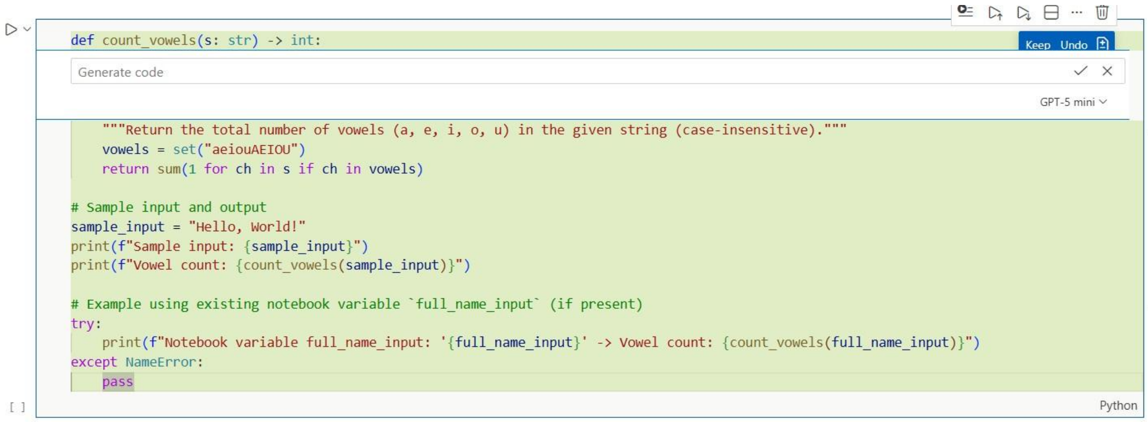
3.Compare the zero-shot and few-shot vowel counting functions based on accuracy, readability, and logical clarity.

Present the comparison in a table or a short reflection paragraph.

Conclude which prompting technique is more effective and why.

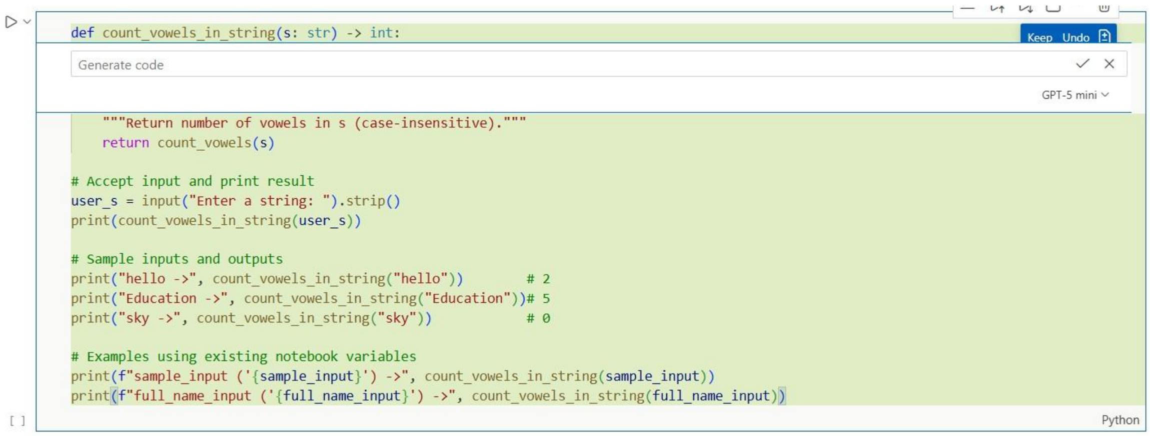
Screenshots :

1.



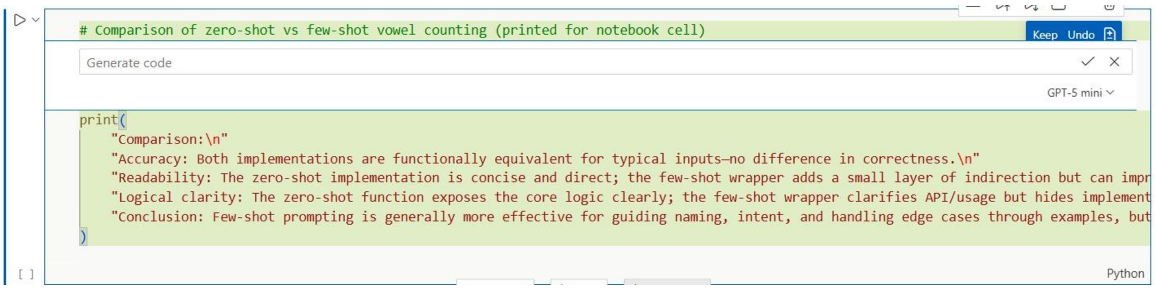
```python
def count_vowels(s: str) -> int:
    """Return the total number of vowels (a, e, i, o, u) in the given string (case-insensitive)."""
    vowels = set("aeiouAEIOU")
    return sum(1 for ch in s if ch in vowels)

# Sample input and output
sample_input = "Hello, World!"
print(f"Sample input: {sample_input}")
print(f"Vowel count: {count_vowels(sample_input)}")

# Example using existing notebook variable `full_name_input` (if present)
try:
    print(f"Notebook variable full_name_input: '{full_name_input}' -> Vowel count: {count_vowels(full_name_input)}")
except NameError:
    pass
```

2.



```python
def count_vowels_in_string(s: str) -> int:
    """Return number of vowels in s (case-insensitive)."""
    return count_vowels(s)

# Accept input and print result
user_s = input("Enter a string: ").strip()
print(count_vowels_in_string(user_s))

# Sample inputs and outputs
print("hello ->", count_vowels_in_string("hello"))        # 2
print("Education ->", count_vowels_in_string("Education"))# 5
print("sky ->", count_vowels_in_string("sky"))            # 0

# Examples using existing notebook variables
print(f"sample_input ('{sample_input}') ->", count_vowels_in_string(sample_input))
print(f"full_name_input ('{full_name_input}') ->", count_vowels_in_string(full_name_input))
```

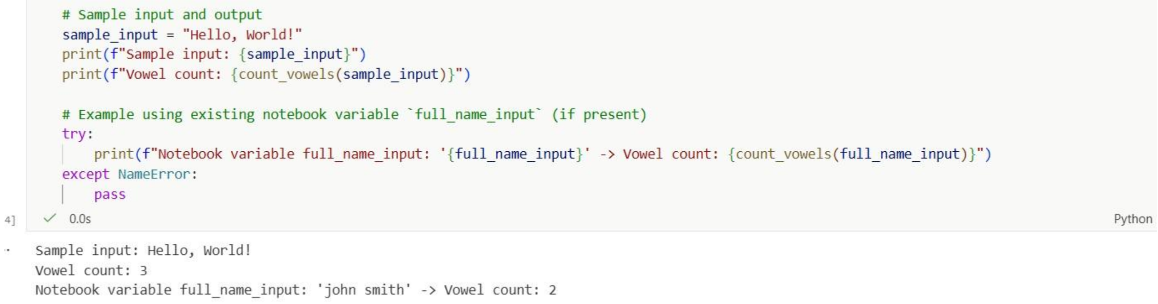
3.



```
# Comparison of zero-shot vs few-shot vowel counting (printed for notebook cell)

print(
    "Comparison:\n"
    "Accuracy: Both implementations are functionally equivalent for typical inputs—no difference in correctness.\n"
    "Readability: The zero-shot implementation is concise and direct; the few-shot wrapper adds a small layer of indirection but can impr
    "Logical clarity: The zero-shot function exposes the core logic clearly; the few-shot wrapper clarifies API/usage but hides implement
    "Conclusion: Few-shot prompting is generally more effective for guiding naming, intent, and handling edge cases through examples, but
)
```

Output :

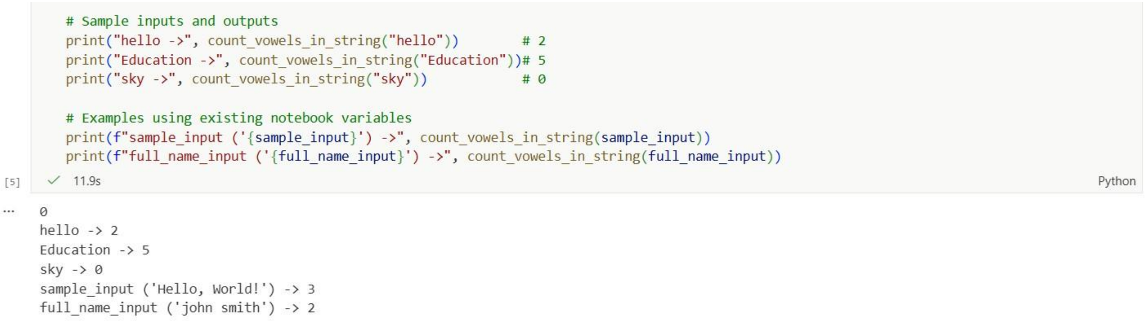
1.



```
# Sample input and output
sample_input = "Hello, World!"
print(f"Sample input: {sample_input}")
print(f"Vowel count: {count_vowels(sample_input)}")

# Example using existing notebook variable `full_name_input` (if present)
try:
    print(f"Notebook variable full_name_input: '{full_name_input}' -> Vowel count: {count_vowels(full_name_input)}")
except NameError:
    pass
```

```
Sample input: Hello, World!
Vowel count: 3
Notebook variable full_name_input: 'john smith' -> Vowel count: 2
```

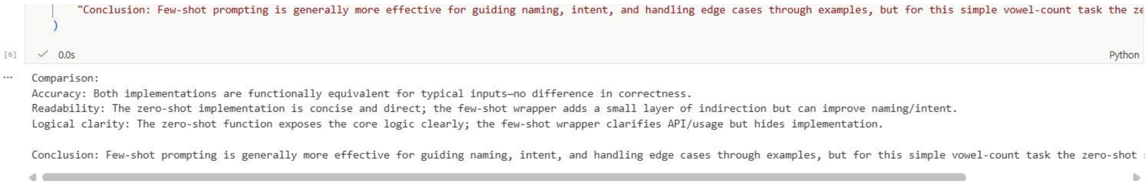
2.



```
# Sample inputs and outputs
print("hello ->", count_vowels_in_string("hello"))        # 2
print("Education ->", count_vowels_in_string("Education"))# 5
print("sky ->", count_vowels_in_string("sky"))            # 0

# Examples using existing notebook variables
print(f"sample_input ('{sample_input}') ->", count_vowels_in_string(sample_input))
print(f"full_name_input ('{full_name_input}') ->", count_vowels_in_string(full_name_input))
```

```
0
hello -> 2
Education -> 5
sky -> 0
sample_input ('Hello, World!') -> 3
full_name_input ('john smith') -> 2
```

3.



```
    "Conclusion: Few-shot prompting is generally more effective for guiding naming, intent, and handling edge cases through examples, but for this simple vowel-count task the ze
)
```

```
Comparison:
Accuracy: Both implementations are functionally equivalent for typical inputs—no difference in correctness.
Readability: The zero-shot implementation is concise and direct; the few-shot wrapper adds a small layer of indirection but can improve naming/intent.
Logical clarity: The zero-shot function exposes the core logic clearly; the few-shot wrapper clarifies API/usage but hides implementation.

Conclusion: Few-shot prompting is generally more effective for guiding naming, intent, and handling edge cases through examples, but for this simple vowel-count task the zero-shot
```

<u>Task-5</u> : Few-Shot Prompting – File Handling Scenario File processing requires clear logical understanding.

Prompt : Read a text file and count the number of lines using Python.

Examples:

Example 1:

File content:

Hello

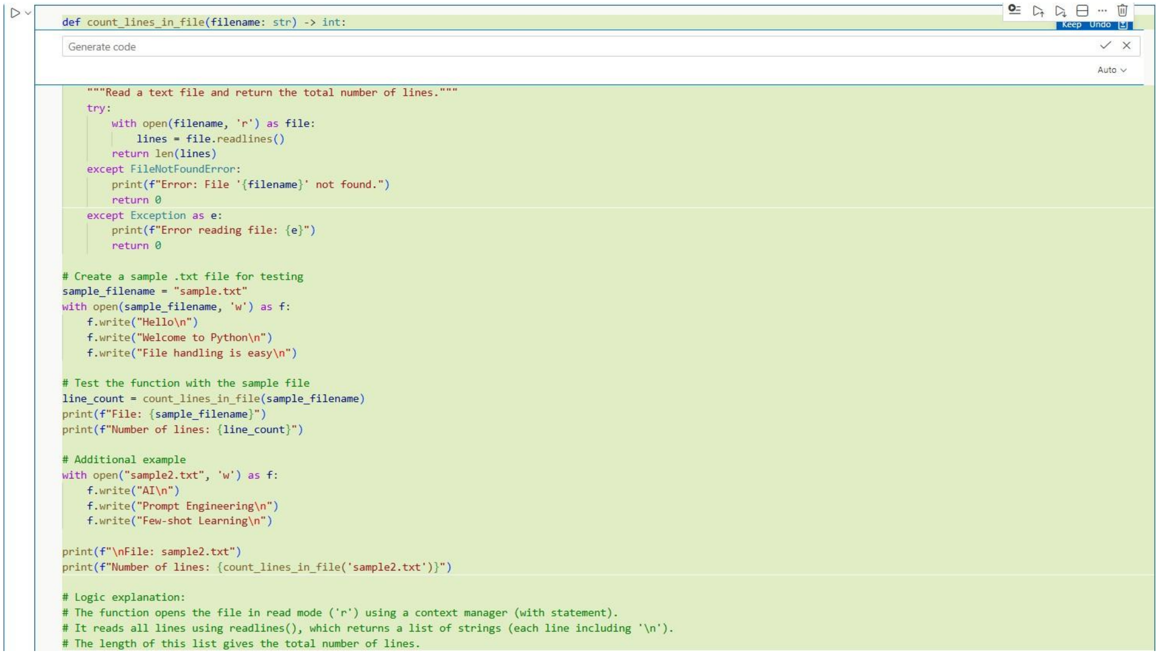
Welcome to Python File

handling is easy

Output:

Number of lines: 3
Example 2:
File content:
AI
Prompt Engineering
Fewshot Learning Output:
Number of lines: 3
Now generate a Python Code that:
- Reads a .txt file given its filename
- Counts the total number of lines in the file
- Returns the line count
- Includes a sample .txt file input and its output
- Explains the logic used in the function

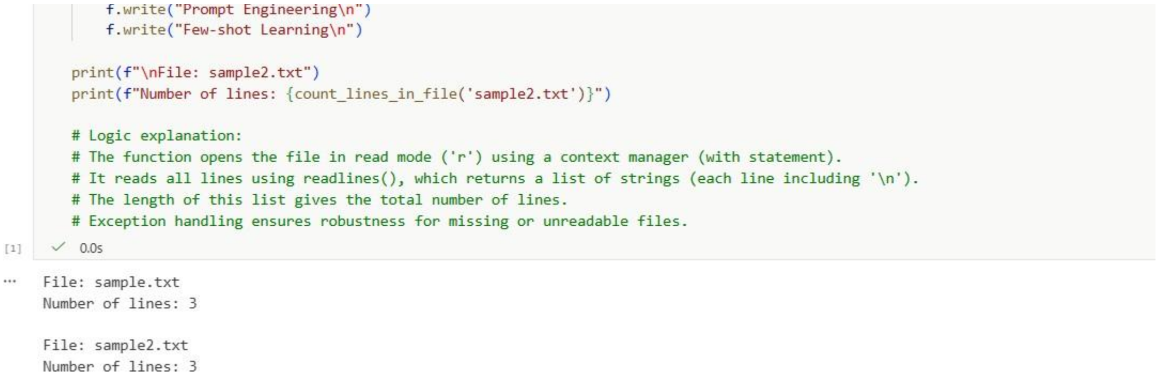
Screenshots :



```python
def count_lines_in_file(filename: str) -> int:
    """Read a text file and return the total number of lines."""
    try:
        with open(filename, 'r') as file:
            lines = file.readlines()
        return len(lines)
    except FileNotFoundError:
        print(f"Error: File '{filename}' not found.")
        return 0
    except Exception as e:
        print(f"Error reading file: {e}")
        return 0

# Create a sample .txt file for testing
sample_filename = "sample.txt"
with open(sample_filename, 'w') as f:
    f.write("Hello\n")
    f.write("Welcome to Python\n")
    f.write("File handling is easy\n")

# Test the function with the sample file
line_count = count_lines_in_file(sample_filename)
print(f"File: {sample_filename}")
print(f"Number of lines: {line_count}")

# Additional example
with open("sample2.txt", 'w') as f:
    f.write("AI\n")
    f.write("Prompt Engineering\n")
    f.write("Few-shot Learning\n")

print(f"\nFile: sample2.txt")
print(f"Number of lines: {count_lines_in_file('sample2.txt')}")

# Logic explanation:
# The function opens the file in read mode ('r') using a context manager (with statement).
# It reads all lines using readlines(), which returns a list of strings (each line including '\n').
# The length of this list gives the total number of lines.
```

Input :
Hello
Welcome to python
File handling is easy

Output :



```
    f.write("Prompt Engineering\n")
    f.write("Few-shot Learning\n")

print(f"\nFile: sample2.txt")
print(f"Number of lines: {count_lines_in_file('sample2.txt')}")

# Logic explanation:
# The function opens the file in read mode ('r') using a context manager (with statement).
# It reads all lines using readlines(), which returns a list of strings (each line including '\n').
# The length of this list gives the total number of lines.
# Exception handling ensures robustness for missing or unreadable files.
```

```
File: sample.txt
Number of lines: 3

File: sample2.txt
Number of lines: 3
```