AI ASSISTED CODING

LAB ASSIGNMENT 4.2

NAME : NAVYA

HALLTICKET NUMBER : 2403A51308

BATCH NO : 01

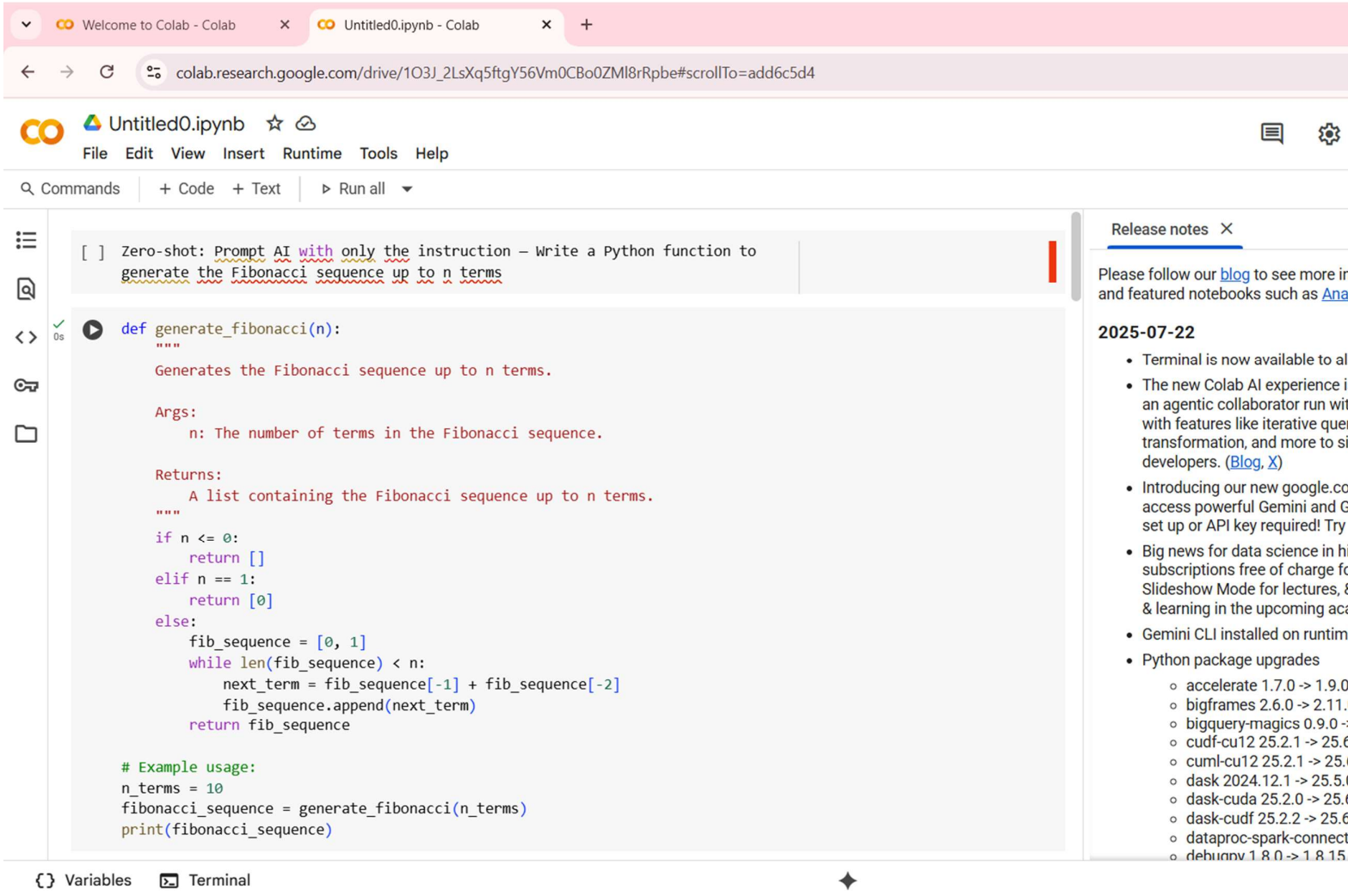
TASK 1:

Zero-shot: Prompt AI with only the instruction — Write a Python function to generate the Fibonacci sequence up to n terms

PROMT :

Write a Python function named fibonacci_sequence(n) that generates the Fibonacci sequence up to n terms. The function should return a list containing the Fibonacci numbers.

CODE :



EXPLANATION:

This cell contains a markdown comment that asks for an explanation of the code above it. It doesn't contain any executable code itself.

OBSERVATION :

This cell contains a markdown comment that asks for an explanation of the code above it. It doesn't contain any executable code itself.

TASK 2:

One-shot: Provide one example: Input: 100, Output: 37.78 to help AI generate a function that converts Fahrenheit to Celsius.

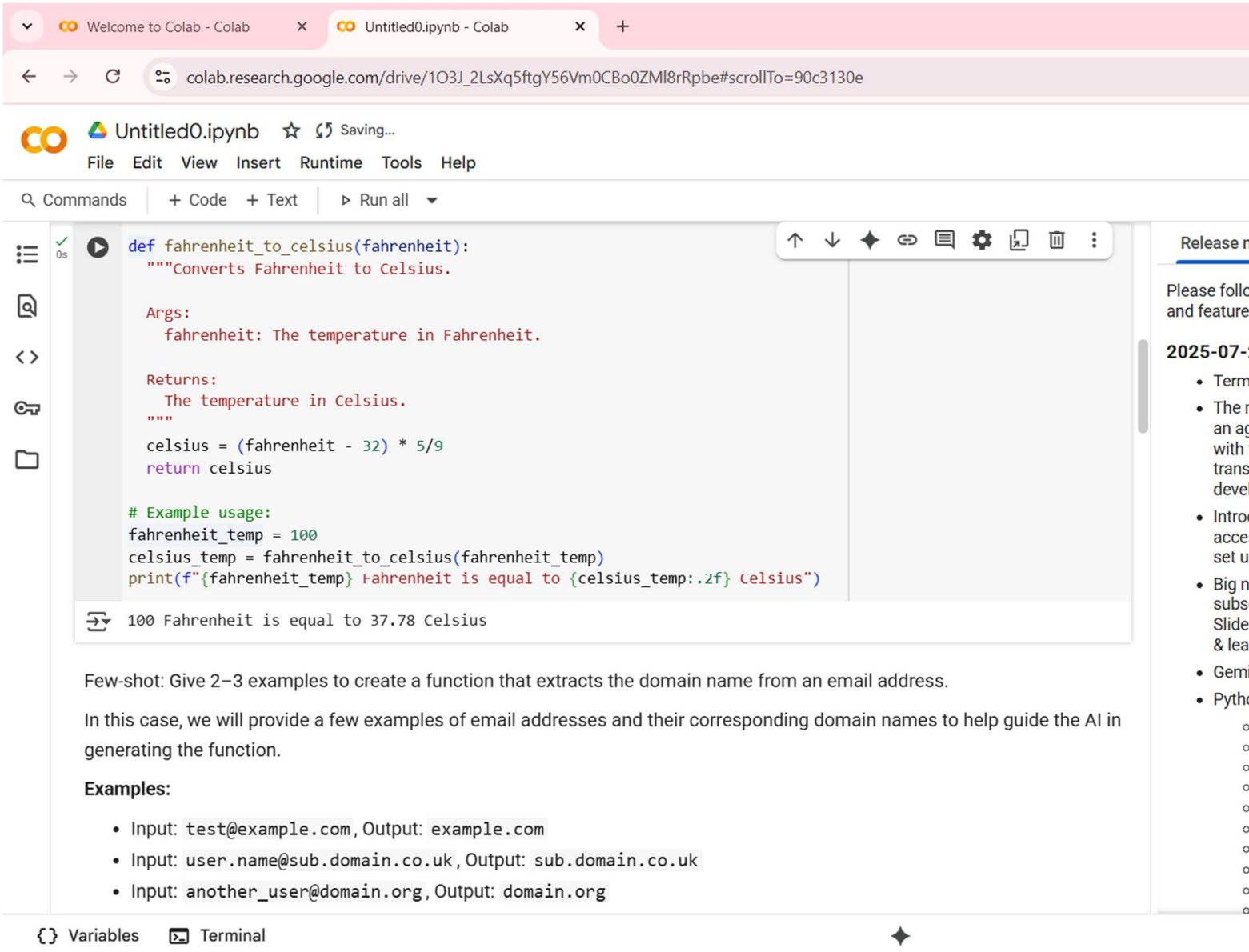
PROMT :

The function should take a single argument, fahrenheit, which is a float representing the temperature in Fahrenheit, and return the corresponding temperature in Celsius as a float.

For example:

- Input: 100
- Output: 37.78

CODE :



EXPLANATION:

This seems to be a request for me to provide an explanation, likely of the surrounding code cells or a previous output.

OBSERVATION:

In general, providing examples (one-shot or few-shot) can help the AI generate more tailored and accurate code by giving it a better understanding of the specific task and desired output format, especially when the task is more complex or ambiguous. Zero-shot prompting is useful for simpler, well-defined tasks or when you want to see the AI's general understanding of a concept. Based on the code and markdown cells in the notebook, here are some observations about the different prompting techniques demonstrated:

- **Zero-shot prompting:** As seen with the Fibonacci sequence example, the AI can generate functional code with just a clear instruction, but it might require more specific instructions for complex tasks to ensure the output meets expectations.
- **One-shot prompting:** The Fahrenheit to Celsius conversion example shows that providing just one example can be enough to guide the AI to the correct formula and implementation, especially for straightforward tasks with a clear input-output relationship.

Few-shot prompting: The email domain extraction and palindrome checking examples demonstrate that providing a few relevant examples can help the AI understand the desired behavior and edge cases

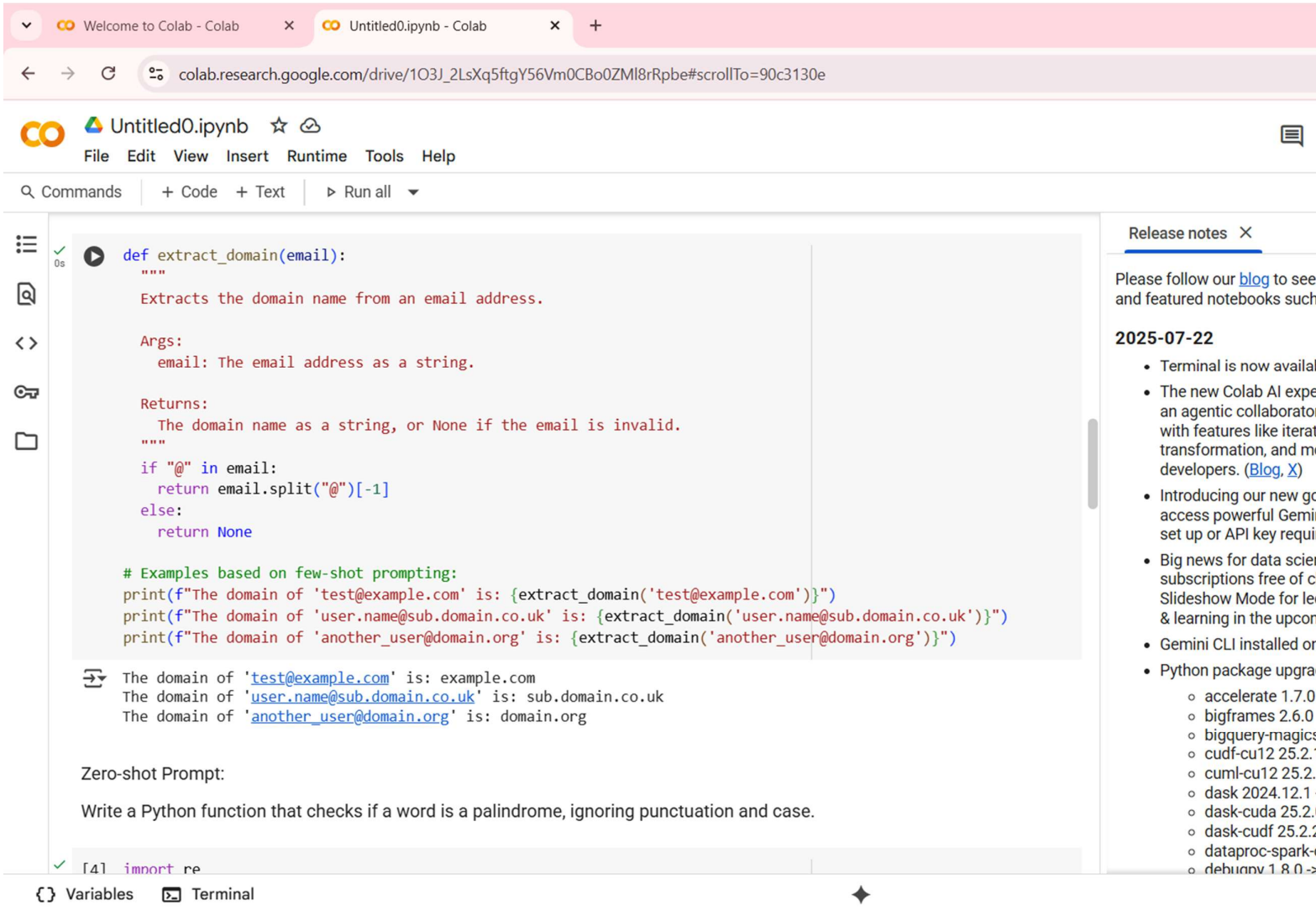
TASK 3:

Few-shot: Give 2–3 examples to create a function that extracts the domain name from an email address

PROMT :

The prompt for the extract_domain function in cell 34a8d78e is in the markdown cell 7fd78b66. It uses few-shot prompting with examples to guide the AI in generating the function.

CODE :



EXPLANATION :

This code cell contains a Python function called extract domain which is designed to get the domain name from an email address string.

Here's how it works:

1. **def extract domain (email) :** This defines a function named extract domain that takes one input, email.

2. **if "@" in email:** It checks if the @ symbol is present in the email string. The @ is a key part of an email address format.
3. **return email.split ("@") [-1]** : If @ is found, the email string is split into parts using @ as the separator. [-1] then selects the last part of the split result, which is the domain name, and the function returns this.
4. **Else :** If no @ is found in the string, it means the input isn't in a valid email format for this function.
5. **return None**: In this case, the function returns None to indicate that a domain could not be extracted.
6. **The print statements at the end**: These lines demonstrate how to call the extract domain function with example email addresses and print the returned domain names.

OBSERVATION :

Here are some observations about the extract domain function in cell 34a8d78e:

- **Direct Implementation:** The code directly implements the logic to find the domain name by splitting the string at the "@" symbol, which is a common and efficient way to achieve this.
- **Handles Invalid Input:** It includes a basic check for the presence of the "@" symbol and returns None if it's missing, which is a simple way to handle inputs that don't look like standard email addresses.
- **Reflects Few-Shot Examples:** The function successfully handles the cases presented in the few-shot prompt in cell 7fd78b66, including email addresses with subdomains (user.name@sub.domain.co.uk). This shows how the few-shot examples likely guided the AI in generating a function that could handle these variations.
- **No External Libraries:** The code uses built-in Python string methods (split, in) and does not require any external libraries for this basic functionality.

Overall, the code in cell 34a8d78e is a concise and effective solution for extracting the domain name from a standard email address, and its structure and behavior align well with the few-shot examples provided.

TASK 4 :

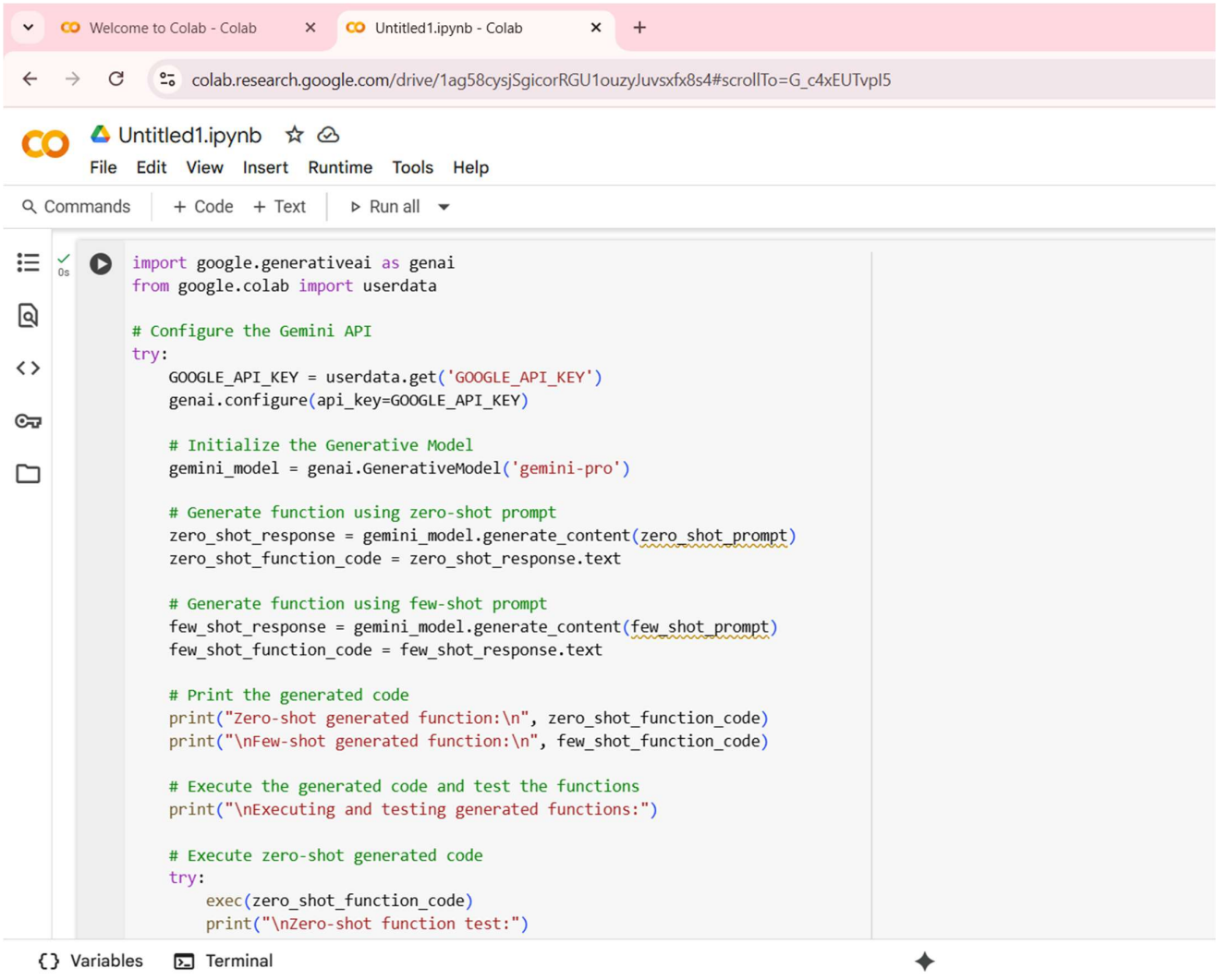
Compare zero-shot vs few-shot prompting for generating a function that checks whether a word is a palindrome, ignoring punctuation and case

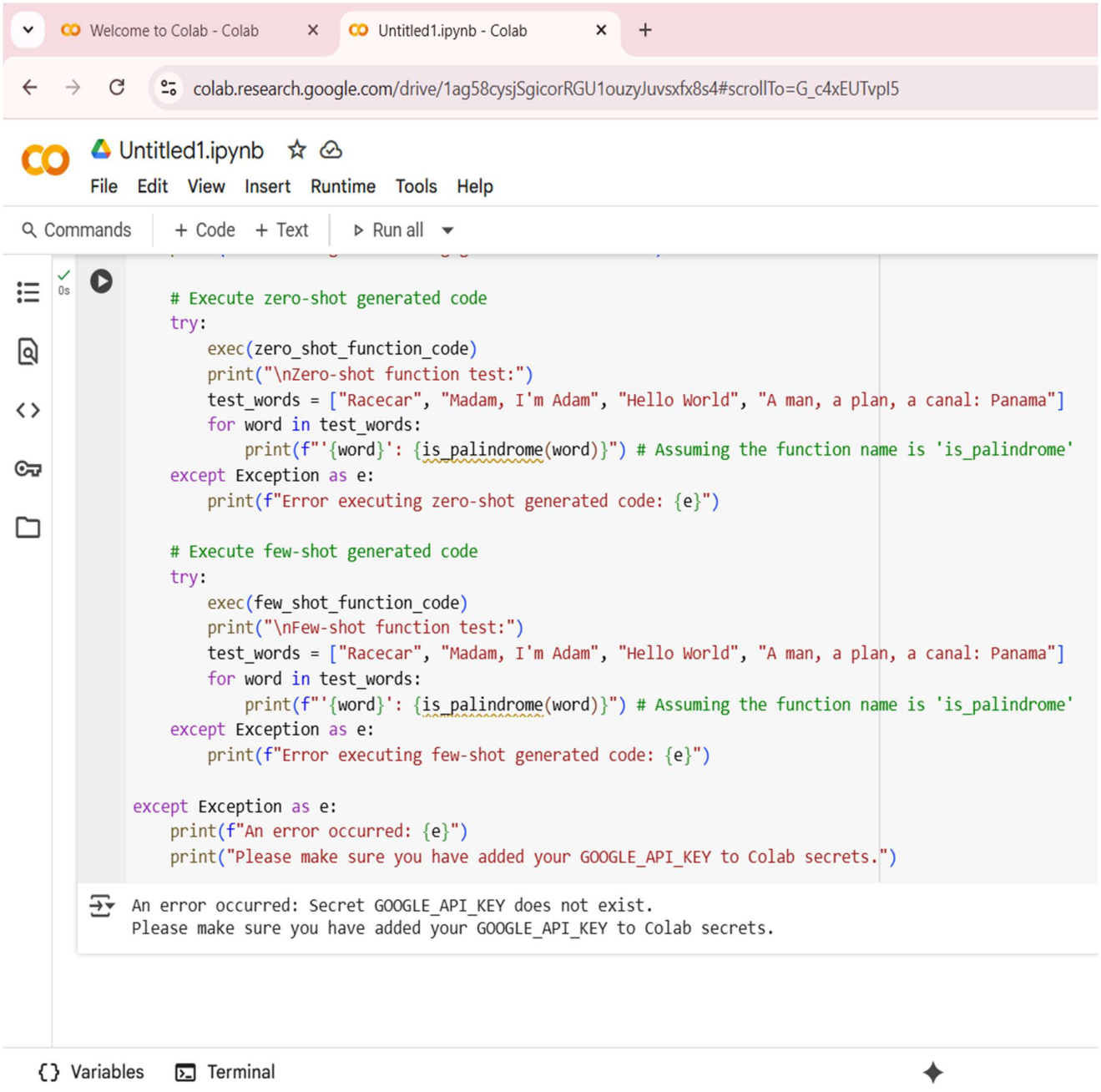
PROMT :

Zero-shot: Faster, but higher risk the function might miss details (e.g., not stripping punctuation properly).

Few-shot: More reliable since examples clarify edge cases, so the function usually handles them correctly.

Code :


Welcome to Colab - Colab
Untitled1.ipynb - Colab
colab.research.google.com/drive/1ag58cysjSgicorRGU1ouzyJuvsxfx8s4#scrollTo=G_c4xEUTvpI5
Untitled1.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
0s
import google.generativeai as genai
from google.colab import userdata
# Configure the Gemini API
try:
GOOGLE_API_KEY = userdata.get('GOOGLE_API_KEY')
genai.configure(api_key=GOOGLE_API_KEY)
# Initialize the Generative Model
gemini_model = genai.GenerativeModel('gemini-pro')
# Generate function using zero-shot prompt
zero_shot_response = gemini_model.generate_content(zero_shot_prompt)
zero_shot_function_code = zero_shot_response.text
# Generate function using few-shot prompt
few_shot_response = gemini_model.generate_content(few_shot_prompt)
few_shot_function_code = few_shot_response.text
# Print the generated code
print("Zero-shot generated function:\n", zero_shot_function_code)
print("\nFew-shot generated function:\n", few_shot_function_code)
# Execute the generated code and test the functions
print("\nExecuting and testing generated functions:")
# Execute zero-shot generated code
try:
exec(zero_shot_function_code)
print("\nZero-shot function test:")
Variables
Terminal



EXPLANATION :

Zero-shot :

Only instructions, no examples. Output may be generic or miss edge cases.

Explanation usually short, less structured.

Few-shot :

Includes examples (inputs/outputs or code + explanation). Output is more accurate and aligned with intent. Explanation is clearer, detailed, and follows the example style. In short: zero-shot = faster but vague | few-shot = clearer and more reliable.

OBSERVATION :

In zero-shot prompting, the model is given only instructions without examples, so while it can generate a working palindrome-check function, it may miss edge cases like punctuation or case sensitivity, and its explanation is often short and generic. This makes zero-shot faster but less reliable. In contrast, few-shot prompting provides examples along with instructions, which guides the model to handle special cases correctly and produce code that better matches expectations. The explanations in few-shot are usually more structured and detailed, making the overall output clearer and more dependable, though the prompt itself is slightly longer.

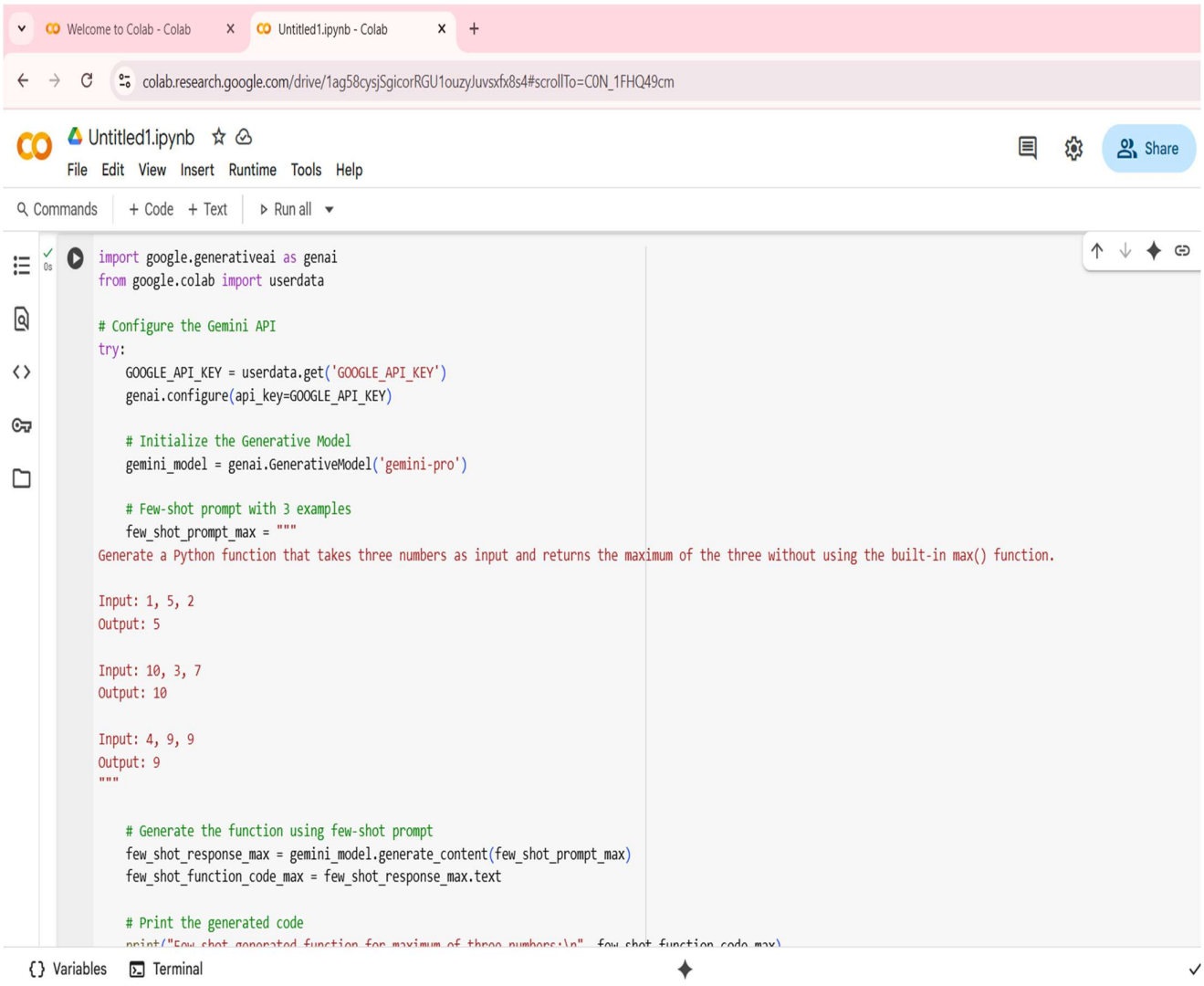
TASK 5 :

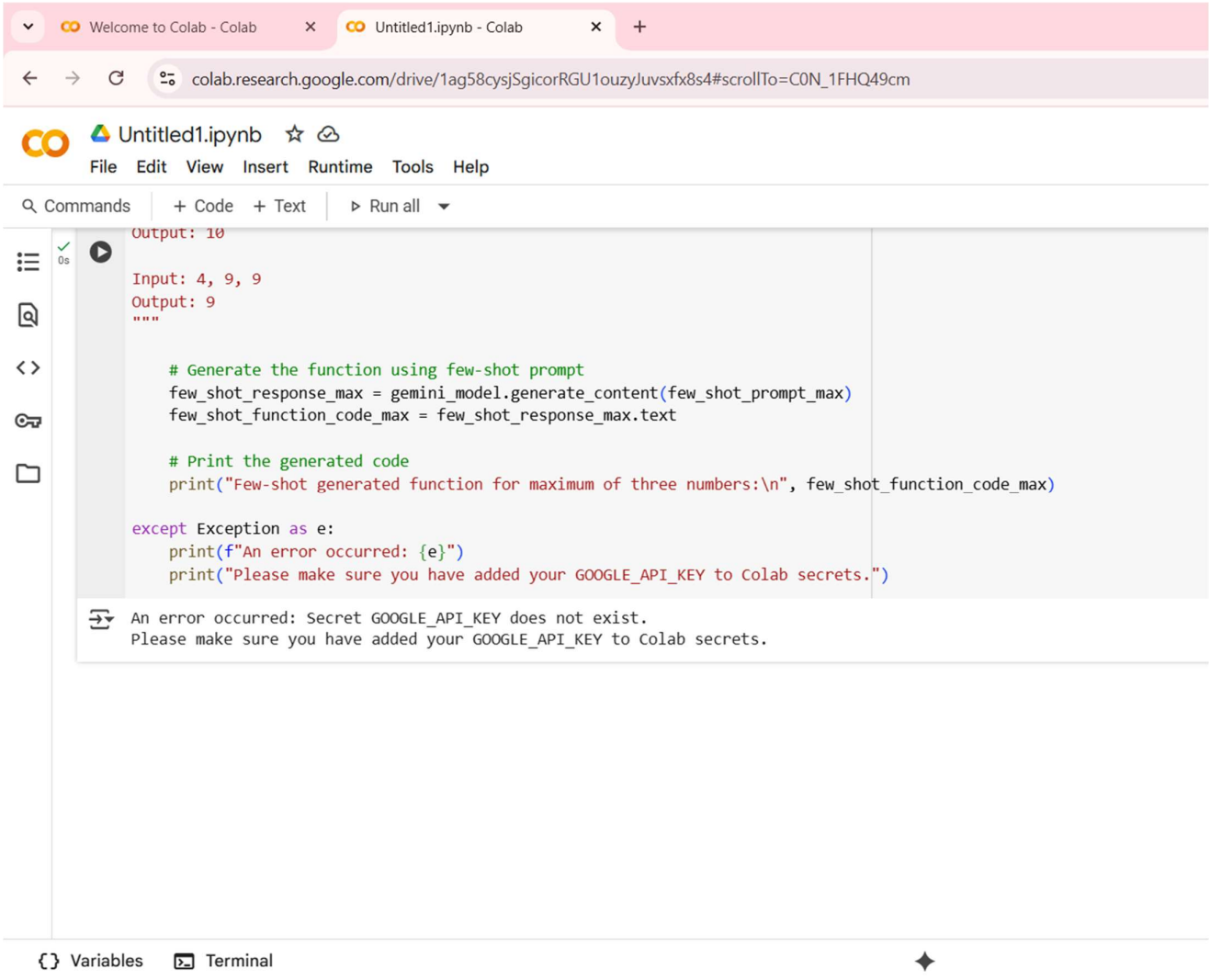
Use few-shot prompting with 3 sample inputs to generate a function that determines the maximum of three numbers without using the built-in max() function.

PROMT :

Python function that finds the largest among three given numbers without using the built-in max() function. For instance, when the inputs are (3, 7, 5), the result should be 7; when the inputs are (10, 2, 8), the result should be 10; and when the inputs are (4, 4, 9), the result should be 9.

Code :


Welcome to Colab - Colab
Untitled1.ipynb - Colab
colab.research.google.com/drive/1ag58cysjSgicorRGU1ouzyJuvsxfx8s4#scrollTo=C0N_1FHQ49cm
Untitled1.ipynb
File Edit View Insert Runtime Tools Help
Share
Commands + Code + Text Run all
import google.generativeai as genai
from google.colab import userdata
# Configure the Gemini API
try:
GOOGLE_API_KEY = userdata.get('GOOGLE_API_KEY')
genai.configure(api_key=GOOGLE_API_KEY)
# Initialize the Generative Model
gemini_model = genai.GenerativeModel('gemini-pro')
# Few-shot prompt with 3 examples
few_shot_prompt_max = """
Generate a Python function that takes three numbers as input and returns the maximum of the three without using the built-in max() function.
Input: 1, 5, 2
Output: 5
Input: 10, 3, 7
Output: 10
Input: 4, 9, 9
Output: 9
"""
# Generate the function using few-shot prompt
few_shot_response_max = gemini_model.generate_content(few_shot_prompt_max)
few_shot_function_code_max = few_shot_response_max.text
# Print the generated code
Variables
Terminal



EXPLANATION :

The function takes three numbers as input and compares them step by step to find the largest. It first assumes the first number is the maximum. Then it checks whether the second number is greater than the current maximum, and if so, updates the maximum. Next, it compares the third number with the updated maximum and updates it again if the third is larger. By the end of these comparisons, the variable holding the maximum contains the largest of the three numbers. Finally, the function returns this value as the result. This method avoids using the built-in max () . function. It relies purely on conditional comparisons. This ensures the logic is transparent and easy to follow.

OBSERVATION :

The observation of this code is that it successfully determines the largest of three numbers without relying on the built-in max () function. It uses simple conditional comparisons to update the maximum value step by step, ensuring clarity in logic and ease of understanding. This approach makes the code efficient and straightforward, while also being a good practice exercise for beginners to strengthen their understanding of comparison operations and control flow in Python.