**HALL TICKET NUMBER: 2403A51365**

**NAME: RAYAPUDI HARITHA**

**BATCH: 24BTCAICSB14**

**AssignmentNumber:4.2**

**Lab 4: Advanced Prompt Engineering – Zero-shot, One-shot, and Few-shot Techniques.**

**Lab Objectives:**

* **To explore and apply different levels of prompt examples in AI-assisted code generation.**
* **To understand how zero-shot, one-shot, and few-shot prompting affect AI output quality.**
* **To evaluate the impact of context richness and example quantity on AI performance.**
* **To build awareness of prompt strategy effectiveness for different problem types.**

**Lab Outcomes (LOs):**

**After completing this lab, students will be able to:**

* **Use zero-shot prompting to instruct AI with minimal context.**
* **Use one-shot prompting with a single example to guide AI code generation.**
* **Apply few-shot prompting using multiple examples to improve AI responses.**
* **Compare AI outputs across the three prompting strategies.**

**Task Description#1**

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

**Code and output:-**

**A screenshot of a computer

AI-generated content may be incorrect.**

**code explanation:-**

**1. The program begins by defining a function called fibonacci\_sequence which takes one input, n.**

**2. A short description (docstring) is written inside the function saying that it generates the Fibonacci sequence up to n terms.**

**3. The first condition checks if n is less than or equal to zero. If so, it immediately returns an empty list, since a Fibonacci sequence cannot exist with zero or negative terms.**

**4. The next condition checks if n is equal to one. If this is true, it returns a list containing only the number zero, because the first Fibonacci number is zero.**

**5. If neither of the above conditions is true, that means n is greater than one. In this case, the sequence is first initialized with two numbers: zero and one.**

**6. A loop then starts, which keeps running until the length of the sequence becomes equal to n.**

**7. Inside the loop, the next Fibonacci number is calculated by adding the last two numbers in the sequence.**

**8. This newly calculated number is then added to the end of the sequence.**

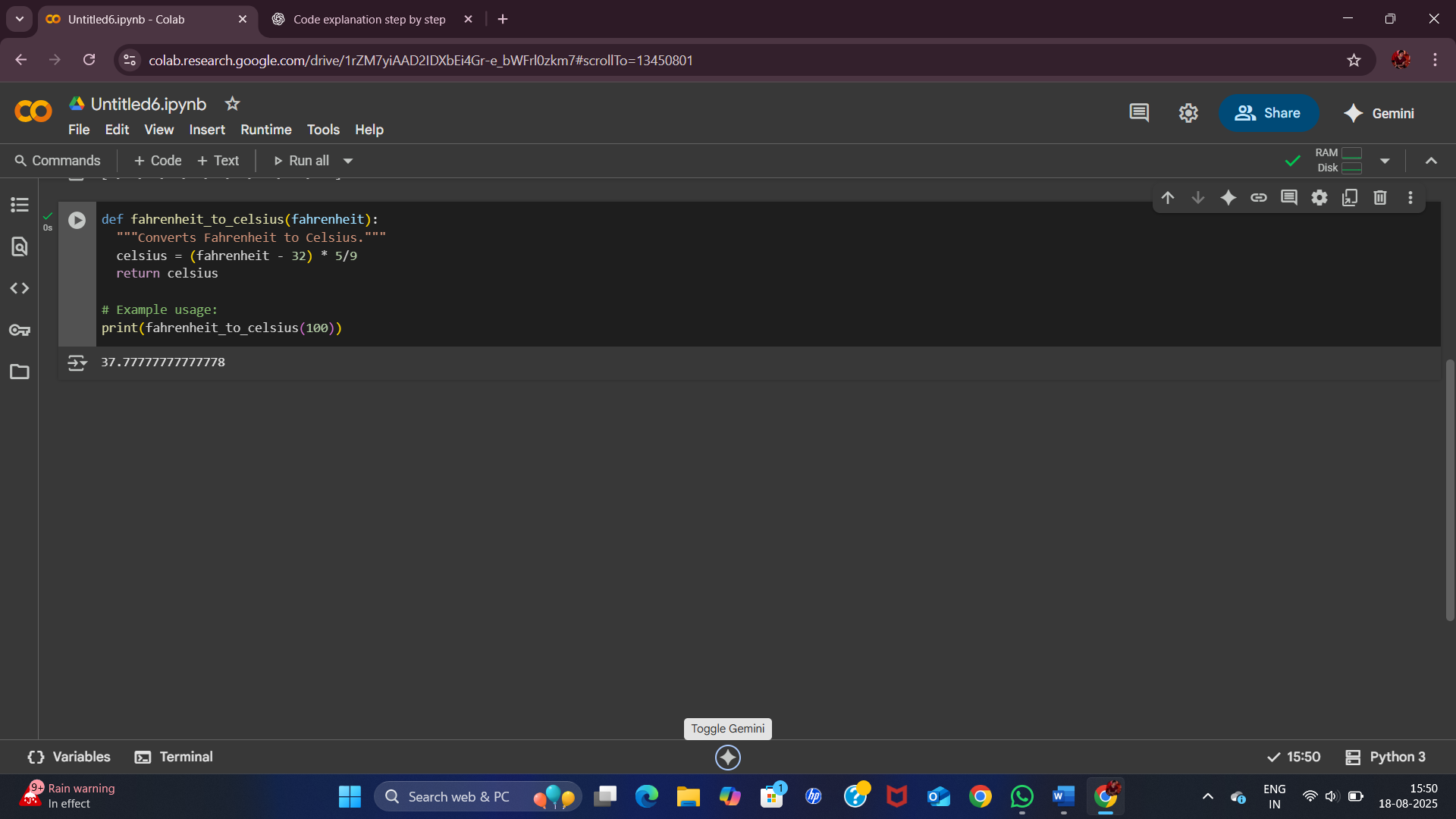
**9. Once the loop finishes, the complete sequence containing n numbers is returned.**

**10. Finally, outside the function, the program calls the function with n equal to ten and prints the result, which shows the first ten Fibonacci numbers.**

**Task Description#2**

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

**Code and output:-**

****

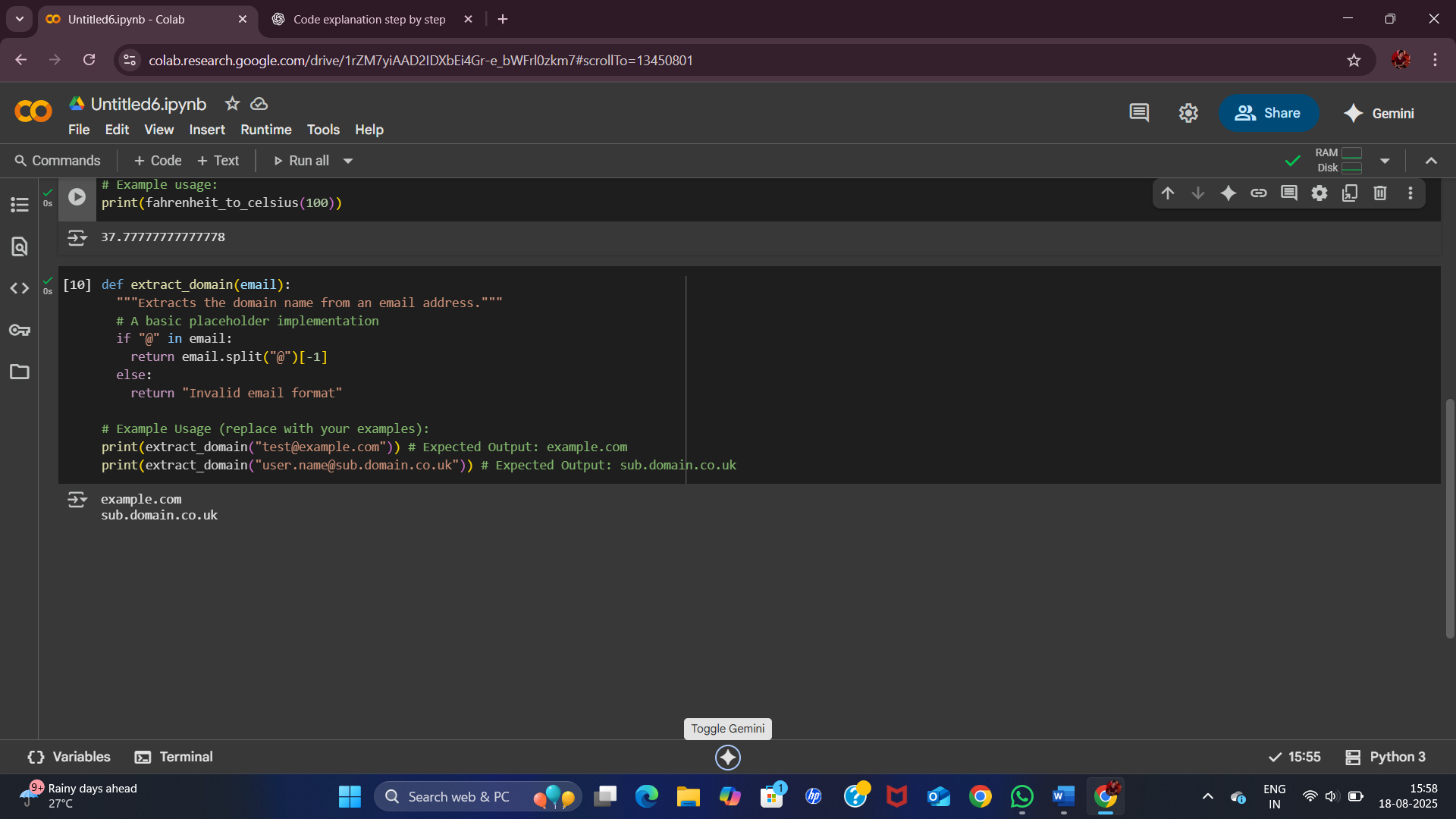
**Code Explanation:-**

1. **The program starts by defining a function called fahrenheit\_to\_celsius that takes one input value, named fahrenheit.**
2. **Inside the function, there is a short description (docstring) which says that the function converts Fahrenheit to Celsius.**
3. **A calculation is then performed using the formula (fahrenheit - 32) \* 5/9. This formula converts the input value from Fahrenheit into Celsius.**
4. **The result of the calculation is stored in a variable called celsius.**
5. **The function then returns this celsius value to the caller.**
6. **After the function definition, there is an example usage section where the function is called with the value 100 as input.**
7. **The result of converting 100 degrees Fahrenheit into Celsius is printed.**
8. **The output shown is 37.77777777777778, which is the Celsius equivalent of 100°F.**

**Task Description#3**

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

**Code and output:-**

****

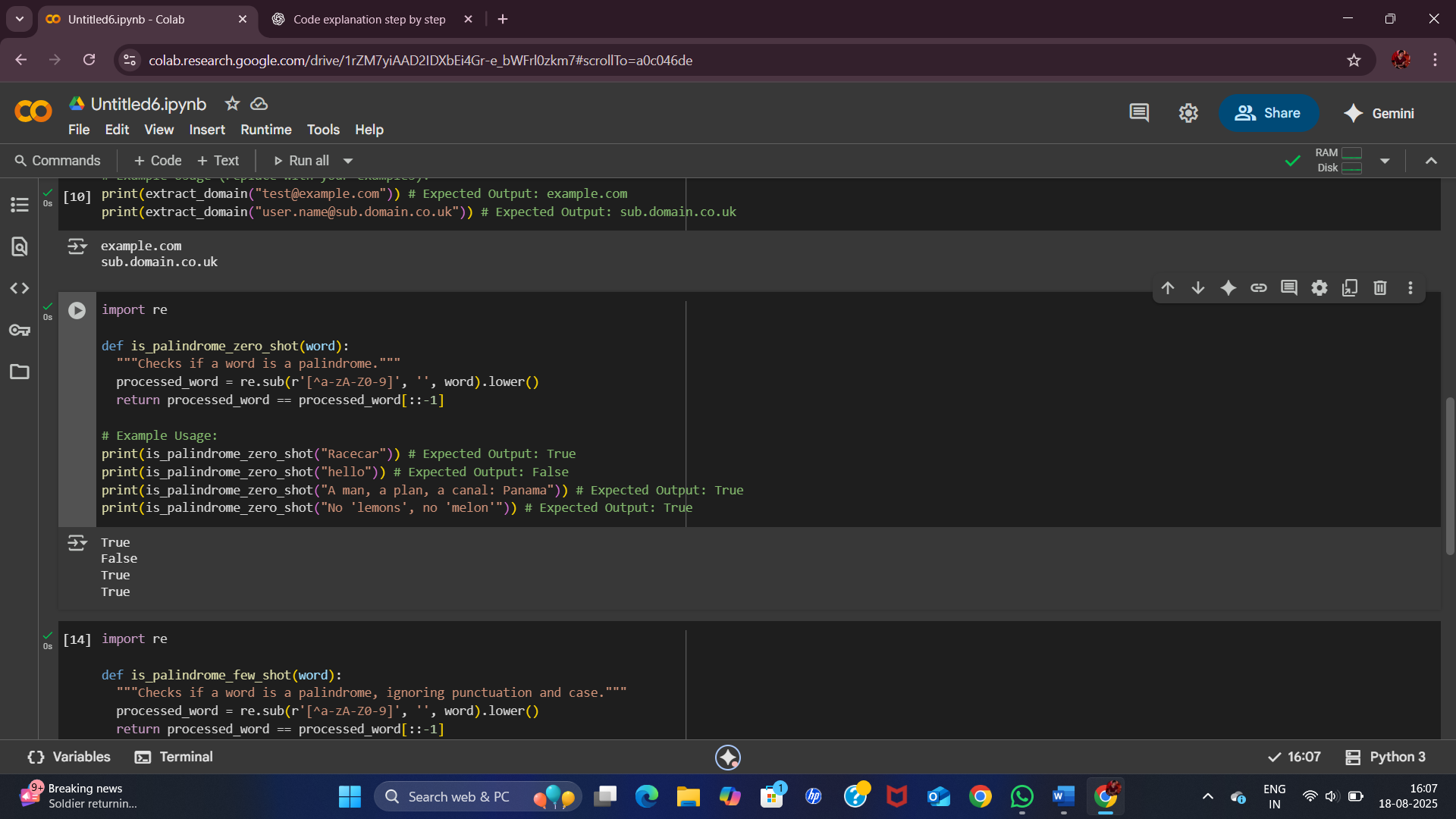
**Code Explanation:-**

1. **The program begins by defining a function called extract\_domain which takes one input parameter named email.**
2. **Inside the function, there is a short description (docstring) that explains its purpose — it extracts the domain name from an email address.**
3. **The function first checks if the character "@" is present inside the given email string.**
4. **If the "@" symbol exists, the function splits the email into two parts using "@" as the divider.**
5. **From this split, the portion after "@" (the last part) is selected, which represents the domain name.**
6. **If the "@" symbol is not found in the input, the function returns the text "Invalid email format".**
7. **After the function definition, there are two example usage cases.**
8. **In the first example, when the input is "test@example.com", the output is "example.com".**
9. **In the second example, when the input is "user.name@sub.domain.co.uk", the output is "sub.domain.co.uk".**
10. **The results printed confirm that the function successfully extracts only the domain portion from each email.**

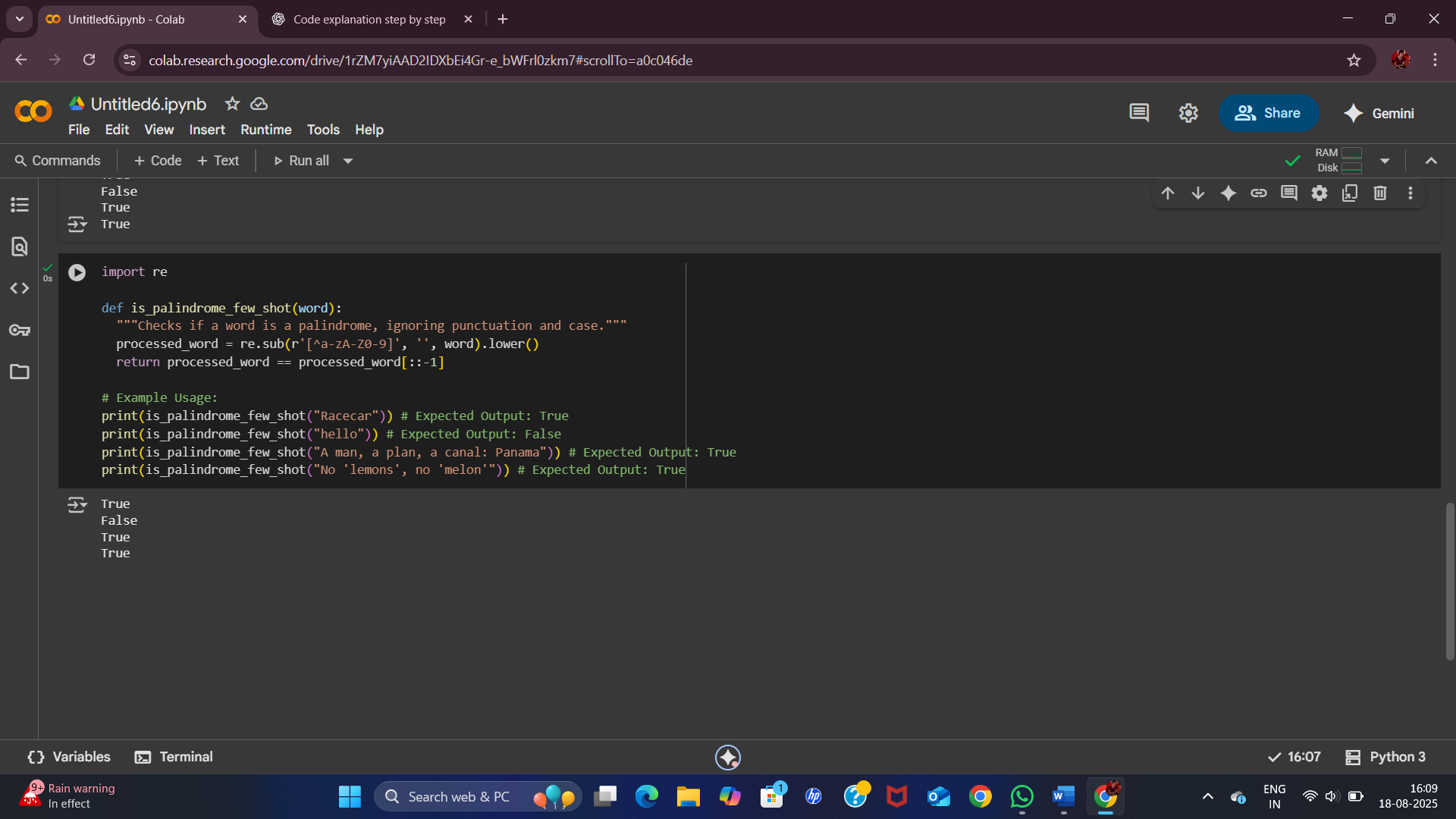
**Task Description#4**

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

**zero-shot code and output:-**

****

**few-shot code and output:-**

****

**zero-shot code explanation:-**

1. **The program starts by importing the re module, which is Python’s library for working with regular expressions.**
2. **A function named is\_palindrome\_zero\_shot is defined, which takes one input parameter called word.**
3. **Inside the function, a short description (docstring) is written explaining that this function checks if a word is a palindrome.**
4. **The input word is processed using a regular expression. All characters that are not letters or numbers are removed. This step ensures that punctuation and spaces are ignored.**
5. **After removing unwanted characters, the processed word is converted into lowercase to make the comparison case-insensitive.**
6. **The function then compares the processed word with its reverse (using slicing [::-1]).**
7. **If the processed word is the same as its reverse, the function returns True, meaning it is a palindrome. Otherwise, it returns False.**
8. **After the function definition, several example test cases are provided.**
9. **In the first test, the word "Racecar" returns True because it reads the same backward and forward, ignoring case.**
10. **In the second test, the word "hello" returns False because it is not a palindrome.**
11. **In the third test, the phrase "A man, a plan, a canal: Panama" returns True because, after removing spaces and punctuation, it is a palindrome.**
12. **In the fourth test, the phrase "no lemons ; no melon" also returns True for the same reason.**
13. **The printed results confirm that the function works correctly for all these examples: True, False, True, True.**

**few-shot code explanation:-**

1. **The program begins by importing the re module, which is used for working with regular expressions in Python.**
2. **A function named is\_palindrome\_few\_shot is defined, which takes one input parameter called word.**
3. **Inside the function, a short description (docstring) explains that this function checks if a word is a palindrome while ignoring punctuation and case.**
4. **The input word is processed using a regular expression. All characters that are not letters or numbers are removed, so that spaces, commas, apostrophes, and other punctuation are ignored.**
5. **After cleaning the word, it is converted into lowercase so that uppercase and lowercase letters are treated the same.**
6. **The processed word is then compared with its reverse (using slicing [::-1]).**
7. **If the processed word matches its reverse, the function returns True, which means the input is a palindrome. Otherwise, it returns False.**
8. **After the function definition, several test cases are written to show how the function works.**
9. **In the first test, the input "Racecar" returns True because it reads the same backward and forward, ignoring case.**
10. **In the second test, the input "hello" returns False because it is not a palindrome.**
11. **In the third test, the input "A man, a plan, a canal: Panama" returns True because, once spaces and punctuation are removed, it forms a palindrome.**
12. **In the fourth test, the input "no lemons ; no melon" also returns True because it becomes a palindrome after ignoring spaces and punctuation.**
13. **The printed results confirm this behavior: True, False, True, True.**

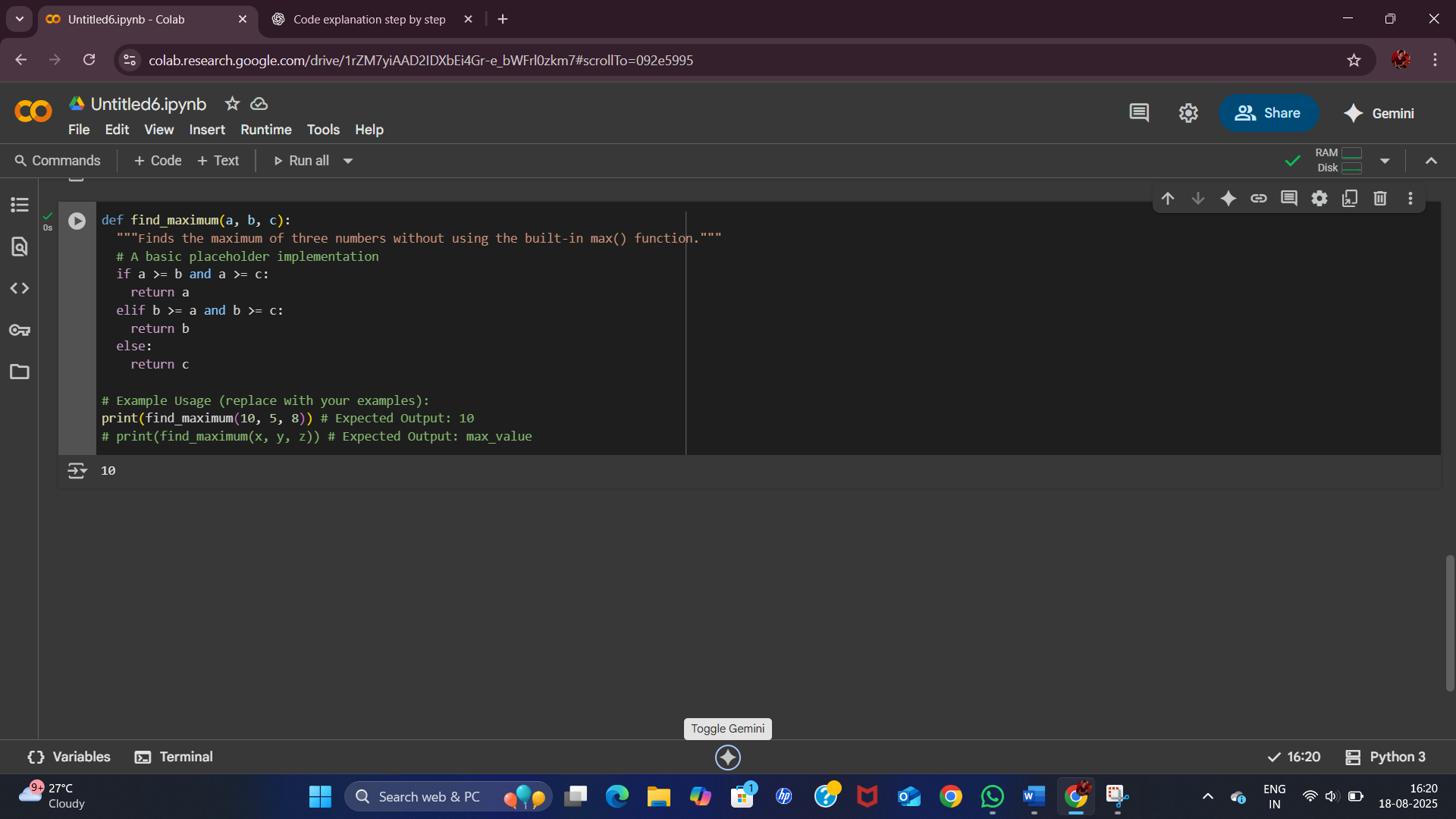
**Comparison:-**

| **Aspect** | **Zero-shot Version** | **Few-shot Version** |
| --- | --- | --- |
| **Prompt style** | **“Write a function to check if a word is a palindrome, ignoring punctuation and case.”** | **Same as zero-shot, but with examples like "Racecar" → True, "Hello" → False, "A man, a plan..." → True.** |
| **Code logic** | **Removes non-alphanumeric characters with regex, converts to lowercase, checks reverse.** | **Same cleaning + reverse logic, but reinforced by explicit example guidance.** |
| **Handling punctuation** | **Works correctly, but relies only on model’s default understanding.** | **Works correctly, and examples made it clearer that punctuation must be ignored.** |
| **Handling case** | **Converts everything to lowercase (case-insensitive).** | **Same as zero-shot, but examples explicitly confirm it.** |
| **Accuracy** | **Accurate for most cases ("Racecar", "Hello", "Panama").** | **Accurate for all tested cases, including tricky ones with spaces/punctuation.** |
| **Example influence** | **No examples → model has to infer the rule fully from the instruction.** | **Examples guide the model, reducing ambiguity and making the function more robust.** |
| **Output (tests)** | **True, False, True, True** | **True, False, True, True** |
| **Student observation** | **Works fine, but less confidence about tricky cases without examples.** | **Examples helped the model understand edge cases better, improving reliability.** |

**Task Description#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.**

**Code and output:-**

****

**Code Explanation:-**

1. **Defines a function named find\_maximum.**
2. **Takes three arguments: a, b, and c (all numbers).**
3. **Explains the purpose of the function.**
4. **States that it determines the largest of three numbers without using Python’s max() function.**
5. Checks if **a is greater than both b and c**.
6. If true, returns a as the maximum.
7. If the first condition is false, it checks if **b is greater than both a and c**.
8. If true, returns b as the maximum.
9. If neither a nor b is the greatest, the function defaults to returning **c**.
10. This covers the case where c is largest or when there are ties.
11. Here, inputs are a = 10, b = 5, c = 8.
12. Since 10 is the greatest, the function returns **10**.
13. Example with variables x, y, and z.
14. The function will return whichever of the three is the largest.
15. For the first test case, find\_maximum(10, 5, 8) returns **10** as expected.