**Roll no:2403A51327**

**Name: Sanjana Karnakanti**

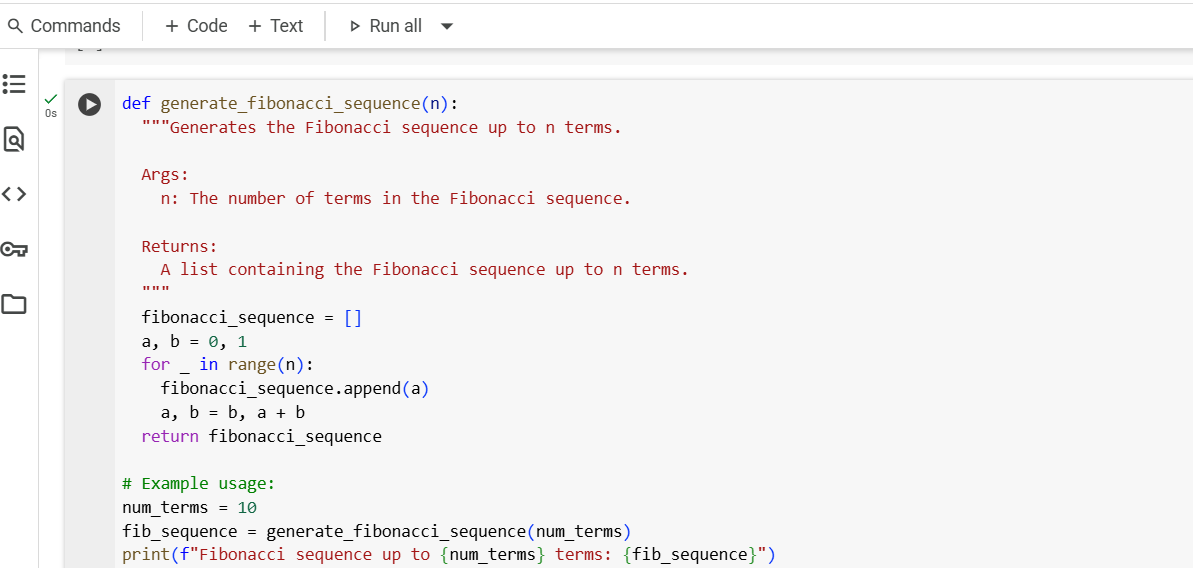
**Batchno:24BtCAICSB13**

**Assignment Number:4.2**

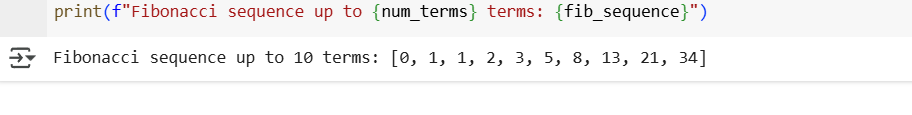
**Lab 4: Advanced Prompt Engineering – Zero-shot, One-shot, and Few-shot Techniques**

**TASK 1:  
PROMPT:** Write a Python function that generates the Fibonacci sequence up to *n* terms.

**CODE:**

****

**OUTPUT:**

****

**Code explanation:**

**1.** Function Definition and Docstring

* The function generate\_fibonacci\_sequence(n) is defined to generate the Fibonacci sequence up to n terms.
* A docstring follows, explaining:
* The purpose of the function.
* The argument n (number of terms).
* The return value (a list of Fibonacci numbers).

**2.** Initialization

* An empty list fibonacci\_sequence is created to store the sequence.
* Two variables a and b are initialized to 0 and 1, representing the first two terms of the Fibonacci sequence.

**3.** Loop to Generate Sequence

* A for loop runs n times. The loop variable is \_ since the index isn't needed.
* In each iteration:
* The current value of a is appended to the list.
* The values of a and b are updated:
* a takes the value of b.
* b becomes the sum of the previous a and b.

**4.** Return Statement

* After the loop completes, the list fibonacci\_sequence is returned, containing the first n Fibonacci numbers.

**5.** Example Usage

* A variable num\_terms is set to 10.
* The function is called with num\_terms and the result is stored in fib\_sequence.
* The sequence is printed using an f-string for clear formatting**.**

**TASK 2:**

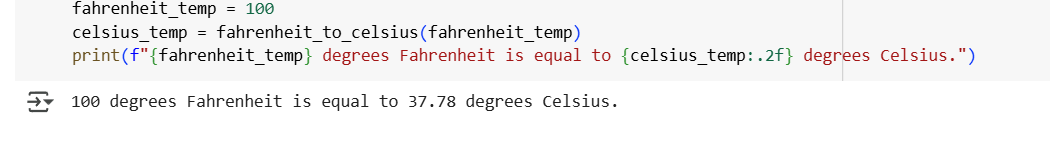
**PROMPT:**

Write a Python function named fahrenheit\_to\_celsius that takes a temperature in Fahrenheit and returns its Celsius equivalent.  
Example:  
fahrenheit\_to\_celsius(100) should return 37.78

**CODE:**

****

**OUTPUT:**

****

**CODE EXPLANATION:**

**1. Function Definition and Docstring**

* def fahrenheit\_to\_celsius(fahrenheit):  
  Defines a function named fahrenheit\_to\_celsius that takes one argument: fahrenheit, representing the temperature in Fahrenheit.
* The docstring explains:
* Purpose: Converts Fahrenheit to Celsius.
* Args: fahrenheit — the input temperature.
* Returns: The equivalent temperature in Celsius.

**2. Conversion Logic**

* celsius = (fahrenheit - 32) \* 5/9  
  Applies the standard formula to convert Fahrenheit to Celsius:

C = \frac{(F - 32) \times 5}{9}

* This calculates the Celsius value and stores it in the variable celsius.

**3. Return Statement**

* return celsius  
  Returns the computed Celsius temperature to the caller.

**4. Example Usage**

* fahrenheit\_temp = 100  
  Sets a sample Fahrenheit temperature to convert.
* celsius\_temp = fahrenheit\_to\_celsius(fahrenheit\_temp)  
  Calls the function with 100 and stores the result in celsius\_temp.
* print(f"{fahrenheit\_temp} degrees Fahrenheit is equal to {celsius\_temp:.2f} degrees Celsius.")  
  Prints the result using an f-string, formatting the Celsius value to 2 decimal places with :.2f.

**TASK 3:**

**PROMPT:**Write a Python function that extracts the domain name from an email address.  
Examples:  
Input: "alice@example.com" → Output: "example.com"

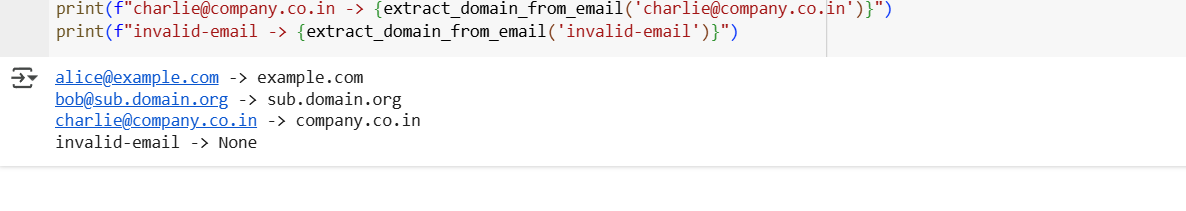
Input: "bob@sub.domain.org" → Output: "sub.domain.org"

Input: "charlie@company.co.in" → Output: "company.co.in"

**CODE:**

****

**OUTPUT:**

****

**CODE EXPLANATION:**

**1. Function Definition and Docstring**

* def extract\_domain\_from\_email(email):  
  Defines a function named extract\_domain\_from\_email that takes one argument: email, a string representing an email address.

**2. Validation and Splitting Logic**

* if "@" in email:  
  Checks if the email string contains the "@" symbol, which is required for a valid email format.
* return email.split("@")[-1]  
  If "@" is present:
* Splits the string at the "@" symbol.
* Returns the last part of the split result, which is the domain name (e.g., "example.com").

**3. Invalid Format Handling**

* return None  
  If the "@" symbol is not found, the function returns None, indicating an invalid email format.

**4. Example Usage**

Each print() statement demonstrates how the function behaves with different inputs:

* "alice@example.com" → returns "example.com"
* "bob@bug.domain.org" → returns "bug.domain.org"
* "charlie@company.co.in" → returns "company.co.in"
* "invalid-email" → returns None (no "@" present)

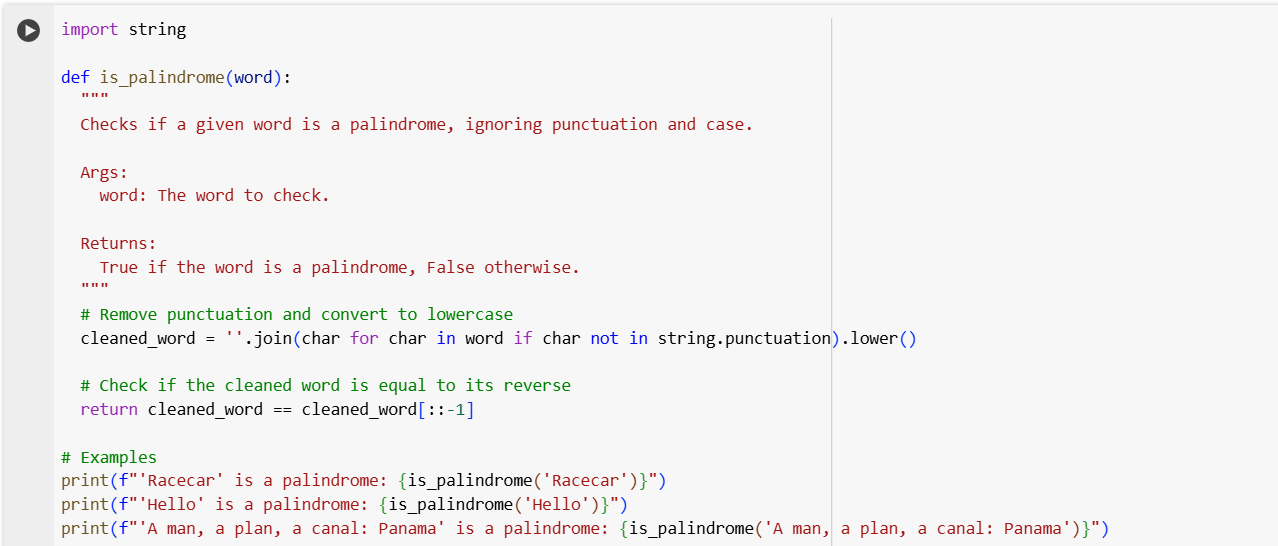
**TASK 4:**

**ZERO SHOT:**

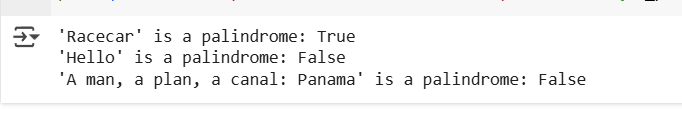
**Prompt:**

Write a Python function that checks whether a given word is a palindrome. The function should ignore punctuation and case.

**CODE:**

****

**OUTPUT:**

****

**FEW-SHOT:  
Prompt:**Write a Python function that checks whether a given word is a palindrome, ignoring punctuation and case.

Examples:

Input: "Madam!"

Output: True

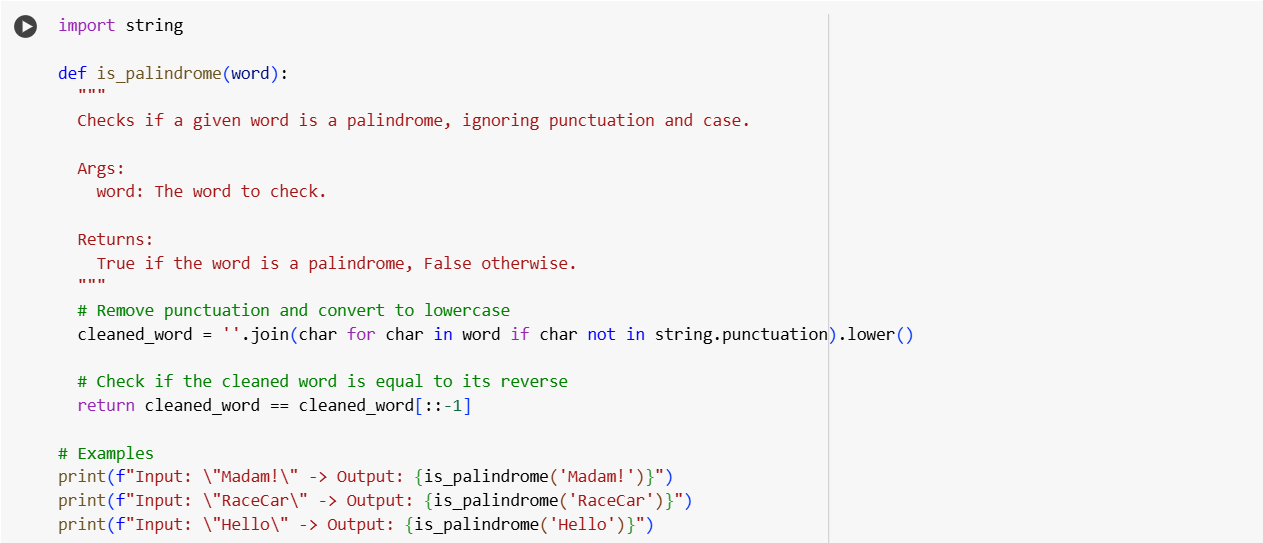
Input: "RaceCar"

Output: True

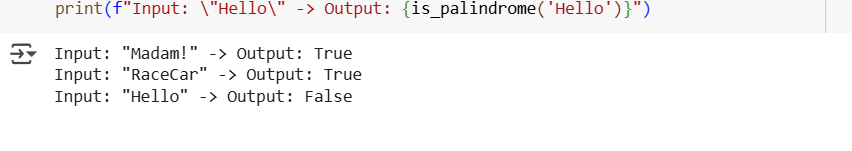
Input: "Hello"

Output: False

**CODE:**

****

**OUTPUT:**

****

**COMPARISON:**

| **Aspect** | **Zero-Shot Prompting** | **Few**-**Shot Prompting** |
| --- | --- | --- |
| Definition | Asking the model to solve the task without examples | Asking the model to solve the task with a few examples |
| Prompt Example | "Write a Python function to check if a word is a palindrome, ignoring punctuation and case." | "Write a Python function that checks if a word is a palindrome. Example: 'Racecar' → True, 'Hello' → False" |
| Context for the Task | Relies solely on the natural language description of the task | Leverages both the description and examples, providing stronger guidance |
| Performance Consistency | May produce less accurate or incomplete functions (e.g., may forget to ignore punctuation) | Tends to produce more accurate and robust solutions due to clarity in expectations |
| Error Handling | Higher chance of logical or implementation errors | Lower chance of errors—examples help the model learn nuances like case folding and character filtering |
| Output Quality | Varies more with prompt wording | More reliable and aligned with expected functionality |
| Best Use Case | When the task is simple and well-known, or speed is more important than precision | When accuracy and robustness are critical, especially with edge cases |
| Adaptability to Edge Cases | Weaker—likely to miss punctuation-related nuances unless explicitly stated | Stronger—examples can illustrate edge cases like "A man, a plan, a canal, Panama!" |

**TASK 5:**

Write a Python function that takes three numbers as input and returns the maximum value without using the built-in max() function.

Example 1:

Input: 3, 7, 5

Output: 7

Example 2:

Input: -10, -5, -20

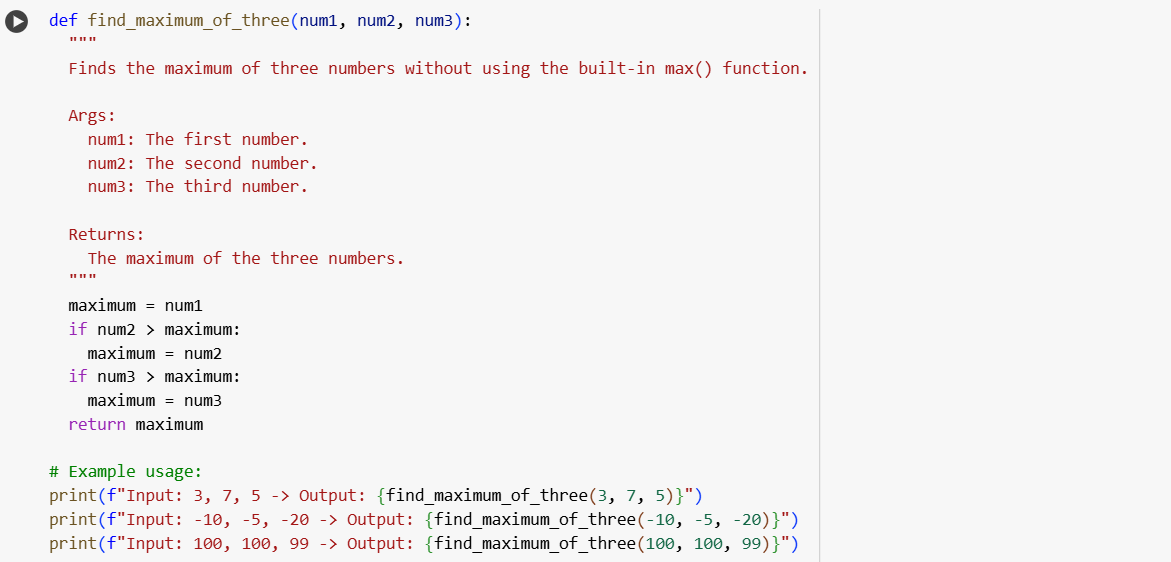
Output: -5

Example 3:

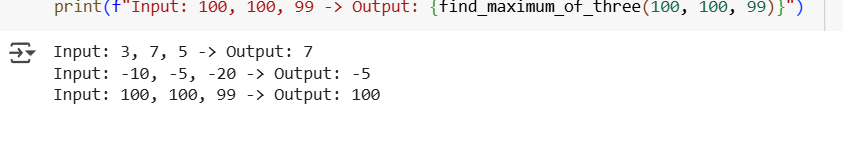
Input: 100, 100, 99

Output: 100

**CODE:**

****

**OUTPUT:**

****

**CODE EXPLANATION:**

* def find\_maximum\_of\_three(a, b, c):  
  Defines a function that takes three arguments: a, b, and c.
* max\_val = a  
  Initializes max\_val with the value of a. This assumes a is the largest until proven otherwise.
* if b > max\_val:  
  Checks if b is greater than the current max\_val. If true, updates max\_val to b.
* if c > max\_val:  
  Similarly, checks if c is greater than the current max\_val. If true, updates max\_val to c.
* return max\_val  
  Returns the final value of max\_val, which is the maximum of the three inputs.