School of Computer Science and Artificial Intelligence

Lab Assignment # 3.2

Program : B. Tech (CSE)

Specialization : AIML

Course Title : AI Assisted

Semester : VI

Academic Session : 2025-2026

Name of Student : G. Shravani

Enrollment No. : 2303A52361

Batch No. : 34

Date : 13/01/26

Lab 3: Prompt Engineering – Improving Prompts and Context Management

---

Lab Objectives

- To understand how **prompt structure and wording** influence AI-generated code.
- To explore how **context (comments, function names, and examples)** helps AI generate more relevant output.
- To evaluate the **quality, accuracy, and structure** of AI-generated code based on prompt clarity.
- To develop **effective prompting strategies** for AI-assisted programming.

Lab Outcomes (LOs)

After completing this lab, students will be able to:

- Generate Python code using **Google Gemini** in **Google Colab**.
- Analyze the effectiveness of **code explanations and suggestions** provided by Gemini.
- Set up and use **Cursor AI** for AI-powered coding assistance.
- Evaluate and refactor Python code using **Cursor AI features**.
- Compare **AI tool behavior and code quality** across different platforms.

Tools Used

- Google Gemini (Google Colab)
- Cursor AI
- Python 3.x

Task Description – 1

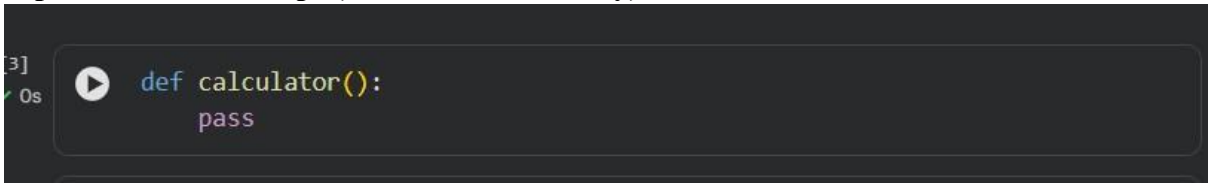
Progressive Prompting for Calculator Design

Design a simple calculator program using **progressive prompt enhancement**:

1. Start with **only the function name**
2. Add **comments describing functionality**
3. Add **usage examples**

Analyze how each step improves the AI-generated code.

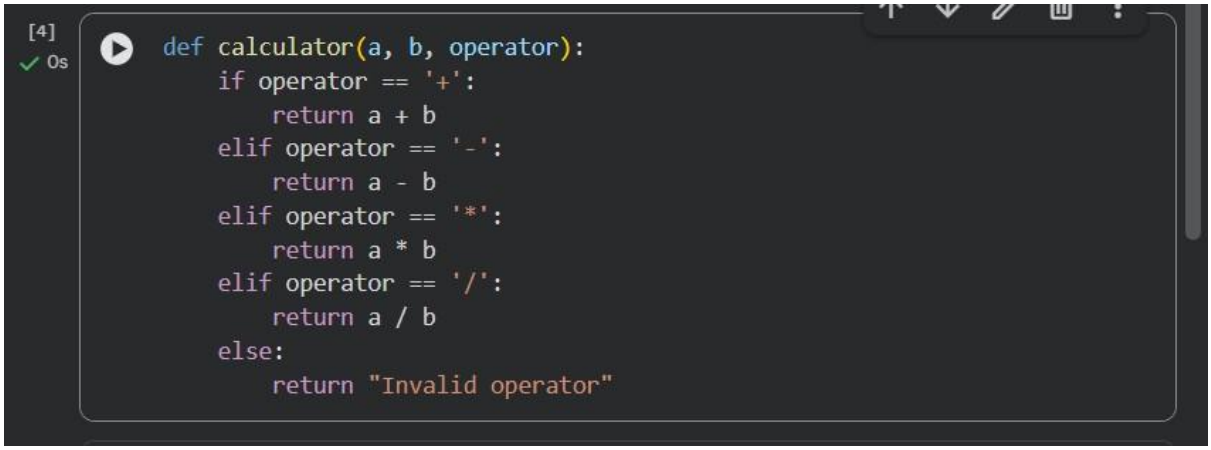
Step 1: Minimal Prompt (Function Name Only):



Observation

- AI lacks context
- No logic, parameters, or functionality
- Output is incomplete and unusable

Step 2: Prompt with Comments (Added Context):

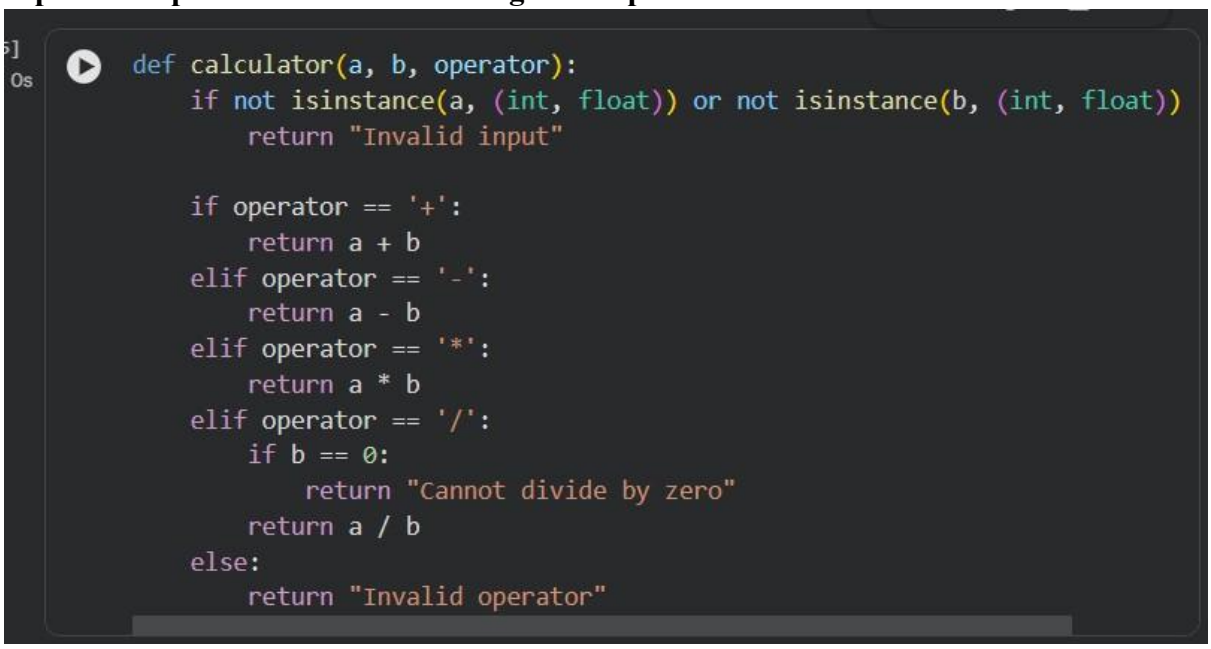


```
[4]
✓ 0s
def calculator(a, b, operator):
    if operator == '+':
        return a + b
    elif operator == '-':
        return a - b
    elif operator == '*':
        return a * b
    elif operator == '/':
        return a / b
    else:
        return "Invalid operator"
```

Observation

- Correct logic implemented
- Parameters inferred correctly
- No input validation (division by zero not handled)

Step 3: Prompt with Comments + Usage Examples:



```
0s
def calculator(a, b, operator):
    if not isinstance(a, (int, float)) or not isinstance(b, (int, float)):
        return "Invalid input"

    if operator == '+':
        return a + b
    elif operator == '-':
        return a - b
    elif operator == '*':
        return a * b
    elif operator == '/':
        if b == 0:
            return "Cannot divide by zero"
        return a / b
    else:
        return "Invalid operator"
```

Observation

- Input validation added
- Division by zero handled
- Code is robust, readable, and user-safe
- Output format matches examples

Expected Output – Comparison Analysis
Comparison of AI-Generated Calculator Code

Prompt Level	Logic Quality	Validation	Robustness	Code Clarity
Function Name Only	Very Poor	None	None	Very Low
With Comments	Good	Partial	Limited	Good
Comments + Examples	**Excellent**	Full	High	**Very High**

Task Description – 2:

Refining Prompts for Sorting Logic Objective:
To observe how refining a vague prompt into a clear, constrained prompt improves the correctness and efficiency of an AI-generated sorting function for student marks.

Step 1: Vague Prompt

```
[6]
✓ 0s
def sort_marks(marks):
    return sorted(marks)
```

Observation

- Sorting order not specified (ascending or descending)
- No validation of input type
- Assumes marks are always valid

Step 2: Refined Prompt with Constraints

```
[7]
✓ 0s
def sort_marks(marks):
    if not all(isinstance(m, (int, float)) for m in marks):
        return "Invalid input"

    return sorted(marks)
```

Observation

- Sorting order clearly defined
- Input validation added
- Logic is more reliable

Step 3: Fully Refined Prompt

```
[8]
✓ 0s
def sort_marks(marks):
    if not isinstance(marks, list) or not all(isinstance(m, (int, float))
        return "Invalid input"

    return sorted(marks, reverse=True)
```

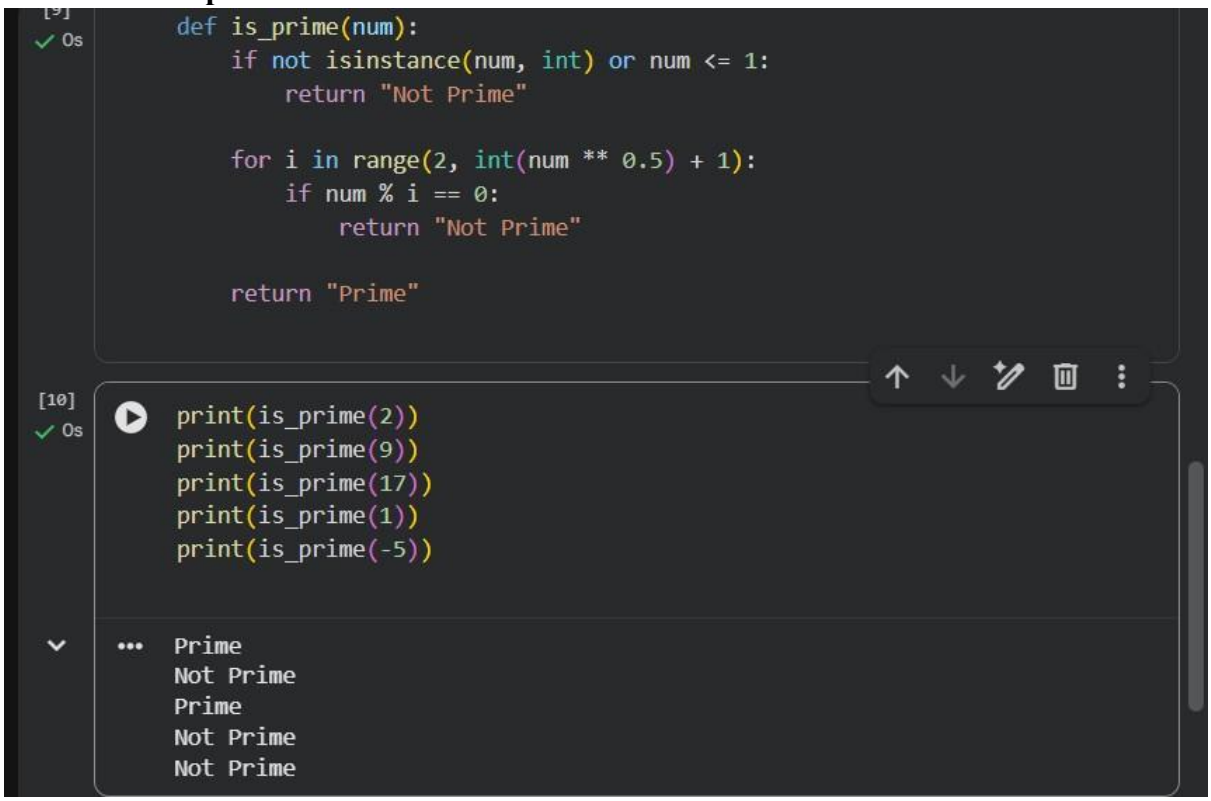
Expected Output – 2 (Analysis)

Prompt Clarity	Sorting Accuracy	Validation	Efficiency
Vague	Low	None	Medium
Refined	High	Partial	High
Fully Refined	**Very High**	**Full**	**High**

Task Description – 3

Few-Shot Prompting for Prime Number Validation Objective:
To analyze how providing multiple input-output examples improves correctness and edgecase handling.

Few-Shot Prompt:



Expected Output – 3

- Correct handling of 0, 1, and negative numbers
- Efficient √n optimization
- Clear output format

Task Description – 4

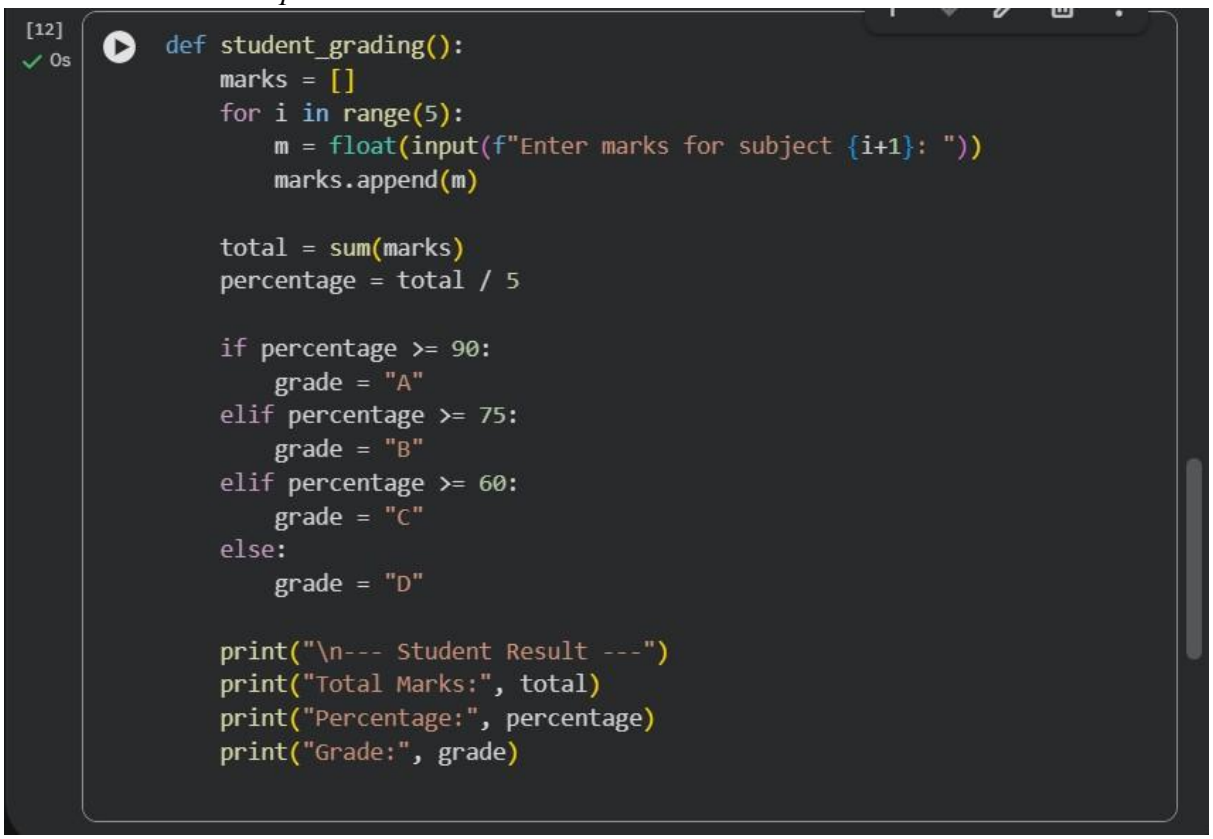
Prompt-Guided UI Design for Student Grading System Objective:

To generate a simple UI-based program that calculates **total marks, percentage, and grade** using prompt guidance.

Prompt

"Design a Python UI program for a student grading system.
The program should:

- *Accept marks of 5 subjects*
- *Calculate total and percentage*
- *Display grade based on percentage*
- *Show clear output"*



Expected Output – 4

- Clean UI flow
- Accurate calculations
- Easy-to-read result display
- Logic directly guided by prompt clarity

Task Description – 5

Analyzing Prompt Specificity in Unit Conversion Functions Objective:
To analyze how clearer instructions improve accuracy in unit conversion functions.

Prompt 1: Vague Prompt

"Write a function to convert distance."

```
[13]
✓ 0s
def convert(distance):
    return distance * 1.6
```

Issue:

- Conversion direction unclear
- No flexibility

Prompt 2: Refined Prompt

"Write a function to convert kilometers to miles."

```
[14]
✓ 0s
def convert(distance):
    return distance * 1.6
```

Prompt 3: Fully Specific Prompt

```
[15]
✓ 0s
def convert_distance(value, unit):
    if not isinstance(value, (int, float)):
        return "Invalid input"

    if unit == "km":
        return value * 0.621371
    elif unit == "miles":
        return value / 0.621371
    else:
        return "Invalid unit"
```

Expected Output – 5 (Analysis)

Prompt Type	Accuracy	Flexibility	Validation
Vague	Low	None	None
Refined	High	Partial	None
Specific	**Very High**	**High**	**Yes**

Final Result :

- Prompt refinement leads to:
  - Better logic
  - Improved validation
  - More efficient algorithms
- Few-shot prompting significantly improves **edge-case handling**
- UI-based programs benefit greatly from **clear task decomposition**

Conclusion

This lab demonstrates that **prompt specificity and context management are critical** in AIassisted programming. As prompts evolve from vague to well-defined, the generated code becomes more accurate, efficient, and usable. Few-shot prompting and UI-guided prompts further enhance reliability and structure.