AI ASSISTED CODING

Sushruth Reddy Chada

2303a51219

BATCH – 04

Lab 3.2: Prompt Engineering – Improving Prompts and Context Management

Lab Title: Prompt Engineering – Improving Prompts and Context Management

Platform Used: Google Colab (Gemini AI)

Lab Objectives

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

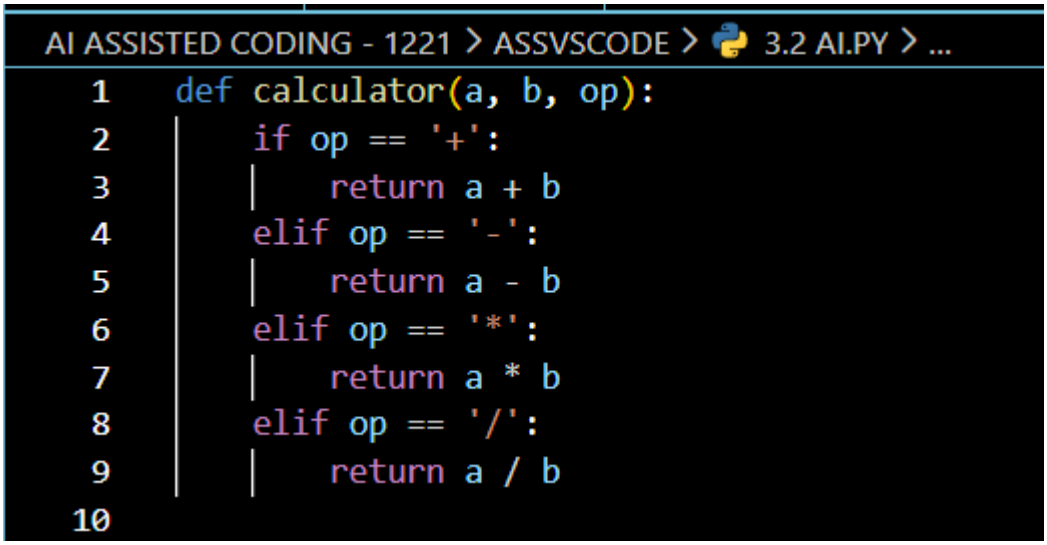
Lab Outcomes (LOs)

After completing this lab, students will be able to:

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

Task Description 1 – Progressive Prompting for Calculator Design

Prompt - Create a calculator in Python.



Issues Identified

- **No user input**
- **No error handling**
- **Function purpose not clear**
- **No comments**

Step 2: Improved Prompt

Create a Python calculator function named simple_calculator that takes two numbers and an operator as input.

Add comments and handle division by zero.

```python
def simple_calculator(a, b, operator):

    if operator == '+':
        return a + b
    elif operator == '-':
        return a - b
    elif operator == '*':
        return a * b
    elif operator == '/':
        if b == 0:
            return "Error: Division by zero"
        return a / b
    else:
        return "Invalid operator"


# Taking input from user
a = int(input("Enter first number: "))
b = int(input("Enter second number: "))
operator = input("Enter operator (+, -, *, /): ")

# Calling function
result = simple_calculator(a, b, operator)
print("Result:", result)
```

Explanation

- Function name clearly explains purpose
- Division by zero handled safely
- Docstring added for clarity

OUTPUT:

```
Enter first number: 12
Enter second number: 2
Enter operator (+, -, *, /): +
Result: 14
PS D:\3-2 SEM>
```

Task Description 2 – Refining Prompts for Sorting Logic

PROMPT

Refining Prompts for Sorting Logic: Start with a vague prompt for sorting student marks, then refine it to clearly specify sorting order and constraints.

Step 1 – Sort Students marks

```
marks = [50, 90, 70]
marks.sort()
print(marks)
```

Problem

- Sorting order unclear
- No function
- No explanation

Step 2 - Write a Python function that sorts student marks in descending order using efficient logic and comments.

```
def sort_marks_desc(marks):

    #Sorts student marks in descending order.

    return sorted(marks, reverse=True)

marks = [50, 90, 70]
print(sort_marks_desc(marks))
```

Explanation

- sorted() is efficient and readable

- reverse=True ensures descending order

OUTPUT

```
PS D:\3-2 SEM> python -u "d:\3-2 SEM\AI ASSISTED CODING - 1221\ASSVSCODE\3.2 AI.PY"
[90, 70, 50]
PS D:\3-2 SEM>
```

Task Description – 3

Few-Shot Prompting for Prime Number Validation

PROMPT

Example:

Input: 7 → Output: Prime

Input: 4 → Output: Not Prime

Write a Python function to check whether a given number is prime.

```
# Example:
# Input: 7 → Output: Prime
# Input: 4 → Output: Not Prime

# Now write a Python function to check whether a number is prime.
```

CODE

```python
def is_prime(n):
    if n <= 1:
        return False

    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            return False

    return True


num = int(input("Enter a number: "))

if is_prime(num):
    print("Prime Number")
else:
    print("Not a Prime Number")
```

Code Explanation

- The function is_prime() takes an integer as input.
- Numbers less than or equal to 1 are immediately rejected.
- The loop checks divisibility only up to the square root of the number, improving efficiency.
- If no divisor is found, the number is declared prime.
- User input is taken and output is displayed clearly.

OUTPUT:

```
PS D:\3-2 SEM> python
Enter a number: 12
Not a Prime Number
```

Input equals to 12 and it is not prime number

Second input 2 for proving that prime number

```
PS D:\3-2 SEM> python -u
Enter a number: 11
Prime Number
PS D:\3-2 SEM>
```

Task Description – 4

Prompt-Guided User Interface Design for Student Grading System

Prompt Used - Create a Python program that accepts student marks as input, calculates the percentage, assigns grades, and displays the result clearly.

CODE

```
def calculate_grade(marks):
    percentage = marks

    if percentage >= 90:
        grade = "A"
    elif percentage >= 75:
        grade = "B"
    elif percentage >= 60:
        grade = "C"
    else:
        grade = "D"

    return percentage, grade


marks = int(input("Enter student marks: "))
percentage, grade = calculate_grade(marks)

print("Percentage:", percentage)
print("Grade:", grade)
```

OUTPUT:

```
PS D:\3-2 SEM> python -u "d:\3-2 SEM\AI ASSISTED CODING - 1221\ASSVSCODE\3.2 AI.PY"
Enter student marks: 95
Percentage: 95
Grade: A
PS D:\3-2 SEM> python -u "d:\3-2 SEM\AI ASSISTED CODING - 1221\ASSVSCODE\3.2 AI.PY"
Enter student marks: 76
Percentage: 76
Grade: B
PS D:\3-2 SEM> python -u "d:\3-2 SEM\AI ASSISTED CODING - 1221\ASSVSCODE\3.2 AI.PY"
Enter student marks: 64
Percentage: 64
Grade: C
PS D:\3-2 SEM>
```

Code Explanation

- **The function calculate_grade() computes the percentage and grade.**
- **Conditional statements are used to assign grades accurately.**
- **User input is taken for marks.**
- **Output is displayed in a clear and readable format.**

Task Description – 5

Analyzing Prompt Specificity in Unit Conversion

Aim - To understand how **specific and detailed prompts** improve the correctness and clarity of AI-generated unit conversion functions.

PROMPT USED

Write a Python function to convert kilometers to miles.

Include comments and display clear output.

CODE:

```python
def km_to_miles(km):

    return km * 0.621371


km = float(input("Enter distance in kilometers: "))
miles = km_to_miles(km)

print("Distance in miles:", miles)
```

OUTPUT:

```
PS D:\3-2 SEM> python -u "d:\3-2 SEM\AI ASSISTED CODING - 1221\ASSVSCODE\3.2 AI.PY"
Enter distance in kilometers: 3
Distance in miles: 1.8641130000000001
PS D:\3-2 SEM>
```

Conclusion

This lab demonstrates that prompt clarity, examples, and specificity significantly improve AI-generated code quality. Few-shot prompting enhances logical accuracy, while structured prompts lead to better user interaction, readability, and correct program flow.