LAB ASSIGNMENT-3.2

2303A52020

B.Varun

Batch-38

Question

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 (like 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 explanations and 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 different platforms.

Task Description-1

• Progressive Prompting for Calculator Design: Ask the AI to design a simple calculator program by initially providing only the function name. Gradually enhance the prompt by adding comments and usage examples.

Expected Output-1

• Comparison showing improvement in AI-generated calculator logic and structure.

PROMPT:

design a simple calculator that can perform addition, subtraction, multiplication, and division operations. The calculator should take two numbers and an operator as input from the user and display the result.

CODE:

```python
def calculator():
    num1 = float(input("Enter first number: "))
    operator = input("Enter operator (+, -, *, /): ")
    num2 = float(input("Enter second number: "))

    if operator == "+":
        result = num1 + num2
    elif operator == "-":
        result = num1 - num2
    elif operator == "*":
        result = num1 * num2
    elif operator == "/":
        if num2 != 0:
            result = num1 / num2
        else:
            print("Error: Division by zero is not allowed.")
            return
    else:
        print("Invalid operator.")
        return

    print(f"Result: {result}")


calculator()
```

OUTCOME:

```
PS C:\Users\varun\OneDrive\Desktop\WEB> & "C:/Program F
Enter first number: 5
Enter operator (+, -, *, /): +
Enter second number: 5
Result: 10.0
PS C:\Users\varun\OneDrive\Desktop\WEB>
```

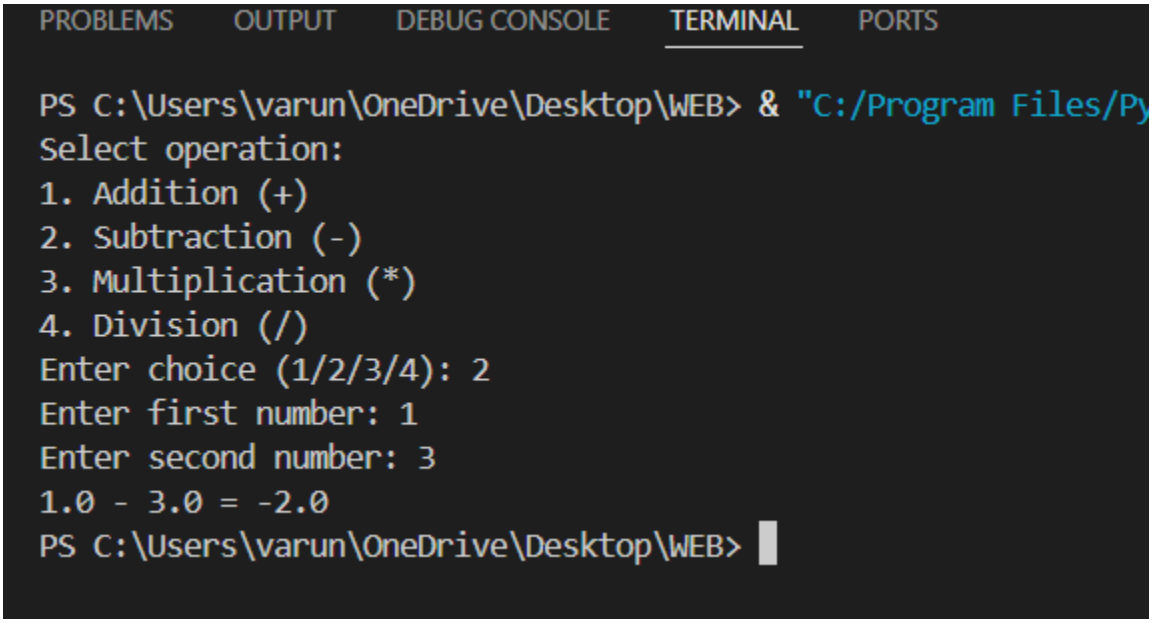
PROMPT:

design a simple calculator that can perform addition, subtraction, multiplication, and division operations. The calculator should take two numbers and an operator as input from the user and display the result and add comment lines and example usages

CODE:

```
def calculator():
    print("2. Subtraction (-)")
    print("3. Multiplication (*)")
    print("4. Division (/)")

    choice = input("Enter choice (1/2/3/4): ")

    try:
        num1 = float(input("Enter first number: "))
        num2 = float(input("Enter second number: "))

        if choice == "1":
            print(f"{num1} + {num2} = {add(num1, num2)}")
        elif choice == "2":
            print(f"{num1} - {num2} = {subtract(num1, num2)}")
        elif choice == "3":
            print(f"{num1} * {num2} = {multiply(num1, num2)}")
        elif choice == "4":
            print(f"{num1} / {num2} = {divide(num1, num2)}")
        else:
            print("Invalid choice")

    except ValueError as e:
        print("Error:", e)


calculator()
```

OUTPUT:



Explanation:

First, a basic prompt was given to create a simple calculator. The code worked but did not have comments or examples, so it was not easy to understand.

Then, the prompt was improved by asking for comments and example usage. Because of this, the generated code became clearer, well-documented, and easier to use.

This shows that improving the prompt helps in getting better and more understandable code.

Task Description-2

• Refining Prompts for Sorting Logic: Start with a vague prompt for sorting student marks,

then refine it to clearly specify sorting order and constraints.

Expected Output-2

• AI-generated sorting function evolves from ambiguous logic to an accurate and efficient implementation.

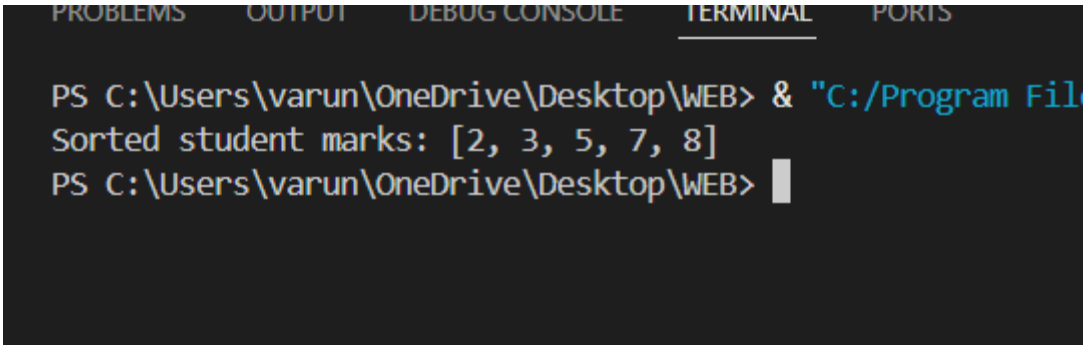
PROMPT:

write a program to sort the student marks

CODE:

```
def sort_student_marks(marks):
    return sorted(marks)


# Example usage
student_marks = [3, 2, 7, 8, 5]
sorted_marks = sort_student_marks(student_marks)

print("Sorted student marks:", sorted_marks)
```

OUTPUT:


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
PS C:\Users\varun\OneDrive\Desktop\WEB> & "C:/Program Fil
Sorted student marks: [2, 3, 5, 7, 8]
PS C:\Users\varun\OneDrive\Desktop\WEB>

PROMPT:

write a program to sort the student marks in ascending order and by using constraints by user input

CODE:

```python
def sort_student_marks(marks):
    # Sort the marks in ascending order
    sorted_marks = sorted(marks)
    return sorted_marks


if __name__ == "__main__":
    # Get number of students from user
    num_students = int(input("Enter the number of students: "))

    # Initialize an empty list to store marks
    student_marks = []

    # Get marks for each student
    for i in range(num_students):
        mark = float(input(f"Enter marks for student {i + 1}: "))
        student_marks.append(mark)

    # Sort the student marks
    sorted_marks = sort_student_marks(student_marks)

    # Display the sorted marks
    print("Sorted student marks in ascending order:")
    for mark in sorted_marks:
        print(mark)
```

OUTPUT:

```
PS C:\Users\varun\OneDrive\Desktop\WEB> & "C:/Program Files/
Enter the number of students: 3
Enter marks for student 1: 20
Enter marks for student 2: 50
Enter marks for student 3: 70
Sorted student marks in ascending order:
20.0
50.0
70.0
PS C:\Users\varun\OneDrive\Desktop\WEB>
```

EXPLANATION:

First, a general prompt was given to sort student marks. Since it did not mention the sorting order or any conditions, the output was basic and unclear.

Then, the prompt was refined by specifying ascending order and user input constraints. This helped

the AI generate a more accurate and well-defined sorting program.

Task Description-3

• Few-Shot Prompting for Prime Number Validation: Provide multiple input-output

examples for a function that checks whether a number is prime. Observe how few-shot prompting improves correctness.

Expected Output-3

• Improved prime-checking function with better edge-case handling.

PROMPT:

Write a program to check whether a number is prime or not

CODE:

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

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

    return True


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

if is_prime(number):
    print(f"{number} is a prime number.")
else:
    print(f"{number} is not a prime number.")
```

OUTPUT:

```
PS C:\Users\varun\OneDrive\Desktop\WEB> & "C:/Program Files/Python312/pyt
Enter a number: 10
10 is not a prime number.
PS C:\Users\varun\OneDrive\Desktop\WEB>
```

PROMPT:

Write a program to check whether a number is prime.

Input: 2

Output: Prime

CODE:

```python
def is_prime(n):
    if n <= 1:
        return "Not Prime"

    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            return "Not Prime"

    return "Prime"


# Example usage
number = 15
result = is_prime(number)
print(result)
```

OUTPUT:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\varun\OneDrive\Desktop\WEB> & "C:/Program Files/Pyth
Not Prime
PS C:\Users\varun\OneDrive\Desktop\WEB>
```

EXPLANATION:

The program checks whether a given number is a prime number.
It takes a number as input and tests if it can be divided by any number other than 1 and itself.

For the input 2, it has no other divisors, so the program prints Prime.

Task Description-4

• Prompt-Guided UI Design for Student Grading System: Create a user interface for a

student grading system that calculates total marks, percentage, and grade based on user

input.

Expected Output-4

• Well-structured UI code with accurate calculations and clear output display.

PROMPT:

Create a user interface for a student grading system that calculates total marks, percentage, and grade based on user input.

CODE:

```python
def calculate_grade(marks_obtained, total_marks):
    percentage = (marks_obtained / total_marks) * 100

    if percentage >= 90:
        grade = 'A'
    elif percentage >= 80:
        grade = 'B'
    elif percentage >= 70:
        grade = 'C'
    elif percentage >= 60:
        grade = 'D'
    else:
        grade = 'F'

    return percentage, grade


def main():
    print("Student Grading System")

    name = input("Enter student name: ")
    total_marks = float(input("Enter total marks: "))
    marks_obtained = float(input("Enter marks obtained: "))

    percentage, grade = calculate_grade(marks_obtained, total_marks)

    print("\nStudent Name:", name)
    print("Total Marks:", total_marks)
    print("Marks Obtained:", marks_obtained)
    print(f"Percentage: {percentage:.2f}%")
    print("Grade:", grade)


if __name__ == "__main__":
    main()
```

OUTPUT:

```
Student Grading System
Enter student name: VARUN
Enter total marks: 95
Enter marks obtained: 90

Student Name: VARUN
Total Marks: 95.0
Marks Obtained: 90.0
Percentage: 94.74%
Grade: A
PS C:\Users\varun\OneDrive\Desktop\WEB>
```

EXPLANATION:

The prompt generated a student grading system that takes user input, calculates percentage, assigns a

grade, and displays the result clearly.

Task Description-5

• Analyzing Prompt Specificity in Unit Conversion Functions: Improving a Unit Conversion Function (Kilometers to Miles and Miles to Kilometers) Using Clear Instructions.

Expected Output-5

• Analysis of code quality and accuracy differences across multiple prompt variations.

PROMPT:

Write a program to convert kilometers to miles and miles to kilometers.

CODE:

```python
def miles_to_km(miles):
    return miles * 1.60934


print("Kilometers to Miles and Miles to Kilometers Converter")

while True:
    print("\n1. Kilometers to Miles")
    print("2. Miles to Kilometers")
    print("3. Exit")

    choice = input("Enter your choice (1/2/3): ")

    if choice == '1':
        km = float(input("Enter distance in kilometers: "))
        miles = km_to_miles(km)
        print(f"{km} kilometers is equal to {miles:.2f} miles.")

    elif choice == '2':
        miles = float(input("Enter distance in miles: "))
        km = miles_to_km(miles)
        print(f"{miles} miles is equal to {km:.2f} kilometers.")

    elif choice == '3':
        print("Exiting the program.")
        break

    else:
        print("Invalid choice. Please enter a valid option (1/2/3).")
```

OUTPUT:

```
PS C:\Users\varun\OneDrive\Desktop\WEB> & "C:/Program Files/P
Kilometers to Miles and Miles to Kilometers Converter

1. Kilometers to Miles
2. Miles to Kilometers
3. Exit
Enter your choice (1/2/3): 2
Enter distance in miles: 10
10.0 miles is equal to 16.09 kilometers.

1. Kilometers to Miles
2. Miles to Kilometers
3. Exit
Enter your choice (1/2/3):
```

EXPLANATION:

The program converts kilometers to miles and miles to kilometers using user input and displays accurate results through a menu-driven interface.