SR UNIVERSITY

AI Assistant Coding Assignment-3.2

Name of Student : G.Sri Sahith

Enrollment No. : 2303A52243

Batch : 37

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.**

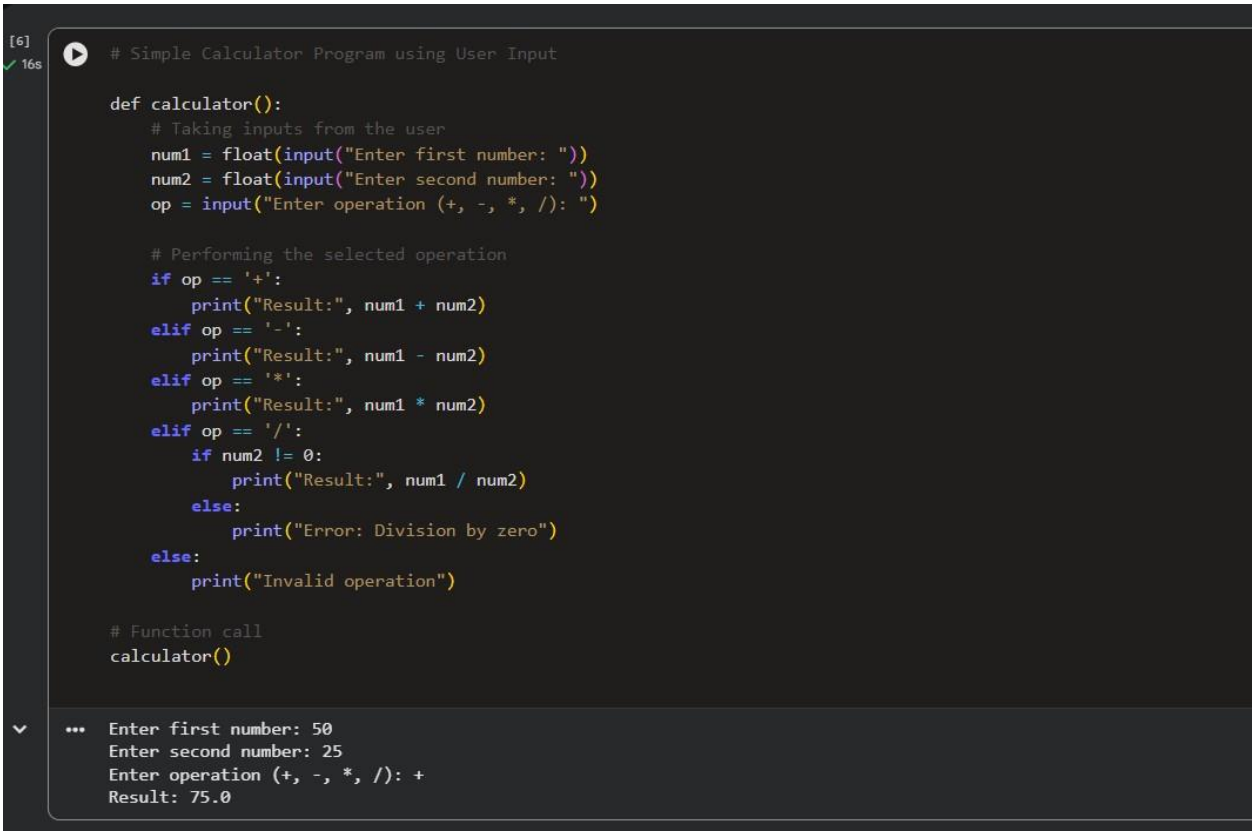
Expected Output-1

- **Comparison showing improvement in AI-generated calculator logic and structure**

PROMPT:

Write a simple calculator program in Python that takes user input for two numbers and an operator (+, −, , /).
The program should perform the selected operation, handle division by zero, and display the result with proper comments

SR UNIVERSITY



```
[6]
✓ 16s
# Simple Calculator Program using User Input

def calculator():
    # Taking inputs from the user
    num1 = float(input("Enter first number: "))
    num2 = float(input("Enter second number: "))
    op = input("Enter operation (+, -, *, /): ")

    # Performing the selected operation
    if op == '+':
        print("Result:", num1 + num2)
    elif op == '-':
        print("Result:", num1 - num2)
    elif op == '*':
        print("Result:", num1 * num2)
    elif op == '/':
        if num2 != 0:
            print("Result:", num1 / num2)
        else:
            print("Error: Division by zero")
    else:
        print("Invalid operation")

# Function call
calculator()

...
Enter first number: 50
Enter second number: 25
Enter operation (+, -, *, /): +
Result: 75.0
```

Task-1 Observation:

Progressive Prompting for Calculator Design

- When only a **function name** is provided, the AI generates **minimal and incomplete logic**.
- Adding **comments and operational details** improves clarity and functionality.
- Including **user input, error handling, and usage context** results in a **fully functional calculator**.
- **Observation:** Progressive prompting significantly enhances code structure, usability, and correctness.

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.

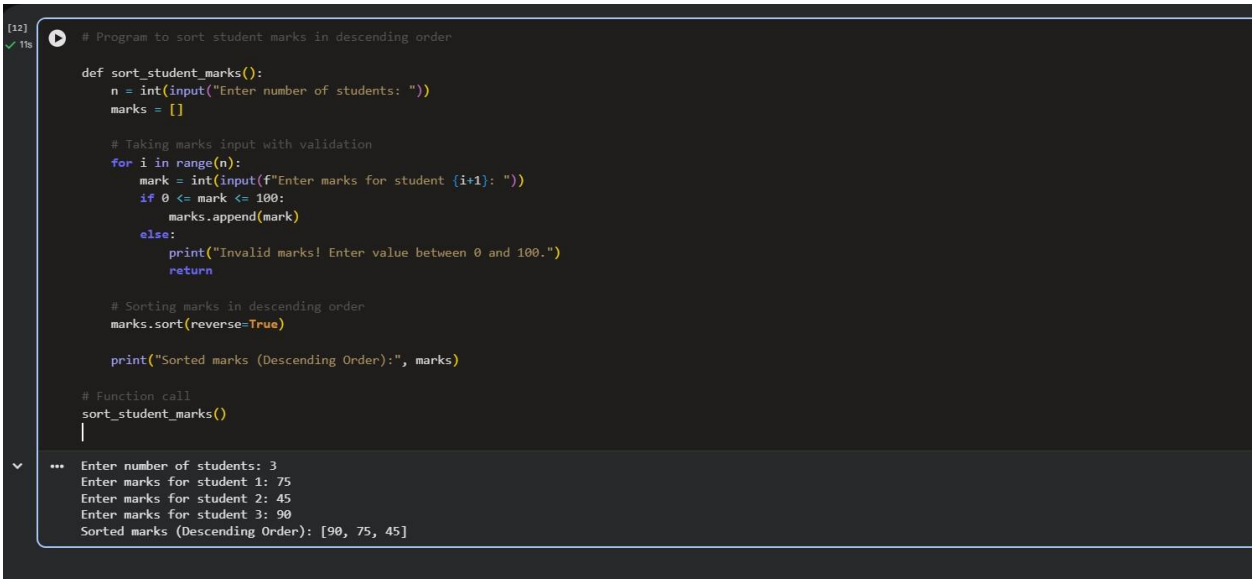
Expected Output-2

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

SR UNIVERSITY

PROMPT:

Create a Python function that accepts student marks as user input, validates that each mark is between 0 and 100, sorts the marks in a specified order (ascending or descending), and prints the sorted result with clear comments.



```
[12]
✓ 11s
# Program to sort student marks in descending order

def sort_student_marks():
    n = int(input("Enter number of students: "))
    marks = []

    # Taking marks input with validation
    for i in range(n):
        mark = int(input(f"Enter marks for student {i+1}: "))
        if 0 <= mark <= 100:
            marks.append(mark)
        else:
            print("Invalid marks! Enter value between 0 and 100.")
            return

    # Sorting marks in descending order
    marks.sort(reverse=True)

    print("Sorted marks (Descending Order):", marks)

# Function call
sort_student_marks()

...
Enter number of students: 3
Enter marks for student 1: 75
Enter marks for student 2: 45
Enter marks for student 3: 90
Sorted marks (Descending Order): [90, 75, 45]
```

Task-2 Observation

Refining Prompts for Sorting Student Marks

- A **vague prompt** produces ambiguous sorting logic with missing requirements.
- Specifying **sorting order (ascending/descending)** improves correctness.
- Adding **constraints (valid marks range) and input method** leads to accurate and reliable output.
- **Observation:** Clear constraints and explicit instructions guide the AI toward efficient and correct sorting logic.

SR UNIVERSITY

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:

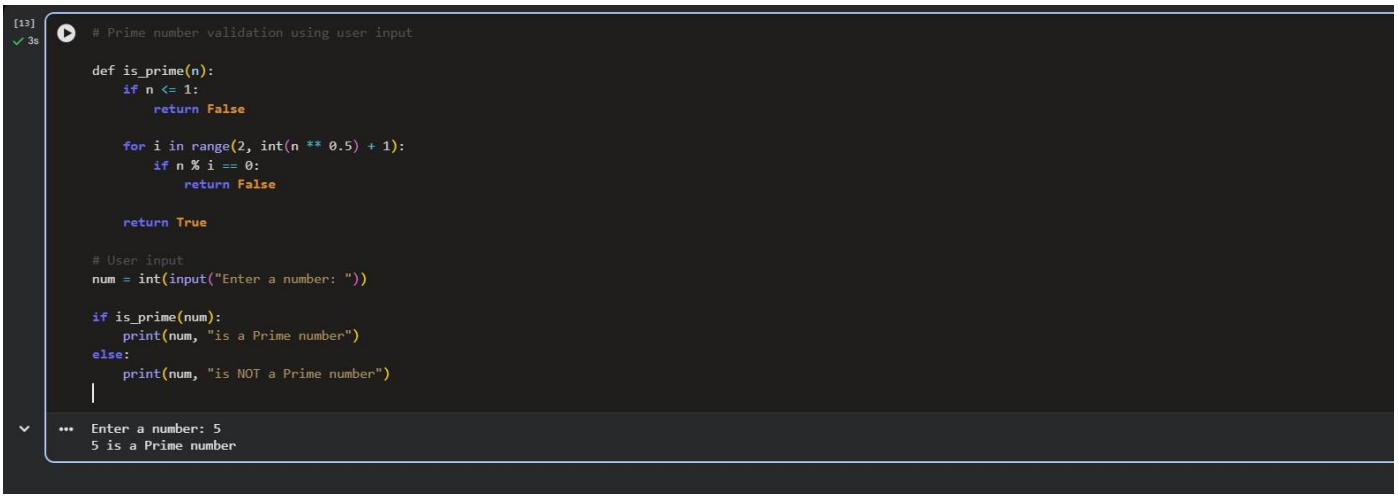
Write a Python function is_prime(n) to check whether a number is prime. Use the following examples to guide your logic and handle edge cases correctly:

is_prime(2) → True is_prime(3) → True

is_prime(4) → False is_prime(1) → False

is_prime(0) → False is_prime(17) → True

The function should be efficient and handle all edge cases



SR UNIVERSITY

Task-3 Observation

Few-Shot Prompting for Prime Number Validation

- Without examples, the AI fails to handle **edge cases** such as 0 and 1.
- Providing **input–output examples** helps the AI infer correct behavior.
- Few-shot prompting improves both **accuracy and efficiency** (Vn optimization).
- **Observation:** Few-shot examples significantly improve correctness and edge-case handling.

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 Python GUI application using Tkinter for a student grading system.
The UI should take marks for three subjects as user input, calculate total marks, percentage, and assign a grade based on the percentage.
Include input validation, clear output display, and proper comments.

Task-4 Observation

Prompt-Guided UI Design for Student Grading System

- Generic UI prompts result in incomplete or poorly structured interfaces.
- Clearly defining **UI elements, calculations, validation, and output format** produces a well-designed application.
- Prompt guidance ensures accurate grade calculation and better user experience.
- **Observation:** Detailed prompts enable AI to generate structured, user-friendly, and functional UI applications.

SR UNIVERSITY

```
[20]
✓ 33s
def calculator():
    # Taking initial input from the user
    try:
        current_result = float(input("Enter initial number: "))
    except ValueError:
        print("Invalid input. Please enter a valid number.")
        return

    while True:
        op = input("Enter operation (+, -, *, /, or 'q' to quit): ")

        if op == 'q':
            print("Final Result:", current_result)
            print("Exiting calculator.")
            break

        if op not in ['+', '-', '*', '/']:
            print("Invalid operation. Please enter +, -, *, /, or q.")
            continue

        try:
            next_number = float(input("Enter next number: "))
        except ValueError:
            print("Invalid input. Please enter a valid number.")
            continue

        if op == '+':
            current_result += next_number
        elif op == '-':
            current_result -= next_number
        elif op == '*':
            current_result *= next_number
        elif op == '/':
            if next_number != 0:
                current_result /= next_number
            else:
                print("Error: Division by zero. Current result remains unchanged.")
                continue # Skip printing result for this invalid operation

        print("Current Result:", current_result)

# Function call
calculator()
```

OUTPUT:

```
Enter initial number: 564
Enter operation (+, -, *, /, or 'q' to quit): *
Enter next number: 565
Current Result: 318660.0
Enter operation (+, -, *, /, or 'q' to quit): +
Enter next number: 56
Current Result: 318716.0
Enter operation (+, -, *, /, or 'q' to quit): q
Final Result: 318716.0
Exiting calculator.
```

SR UNIVERSITY

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.

PRMOPT:

Develop a Python program that allows users to convert distances between kilometers and miles. The program should take user input for the value and conversion direction, use accurate standard conversion factors, handle invalid input, and display the converted result with proper comments.

```python
# Unit Conversion Program: Kilometers ↔ Miles

def convert_units():
    choice = input("Enter conversion type (km_to_miles / miles_to_km): ")
    value = float(input("Enter distance value: "))

    if choice == "km_to_miles":
        result = value * 0.621371
        print("Miles:", result)

    elif choice == "miles_to_km":
        result = value * 1.60934
        print("Kilometers:", result)

    else:
        print("Invalid conversion type")

# Function call
convert_units()
```

OUTPUT:

```
Enter conversion type (km_to_miles / miles_to_km): 56
Enter distance value: 452
Invalid conversion type
```

SR UNIVERSITY

Task-5 Observation

Analyzing Prompt Specificity in Unit Conversion Functions

- Vague prompts lead to **ambiguous or incorrect conversion logic**.
- Moderately specific prompts improve formulas but lack usability.
- Highly specific prompts produce **accurate, user-interactive, and maintainable code**.
- **Observation:** Increasing prompt specificity directly improves accuracy, clarity, and real-world usability.