

N.Sai Akash

2403A51L57

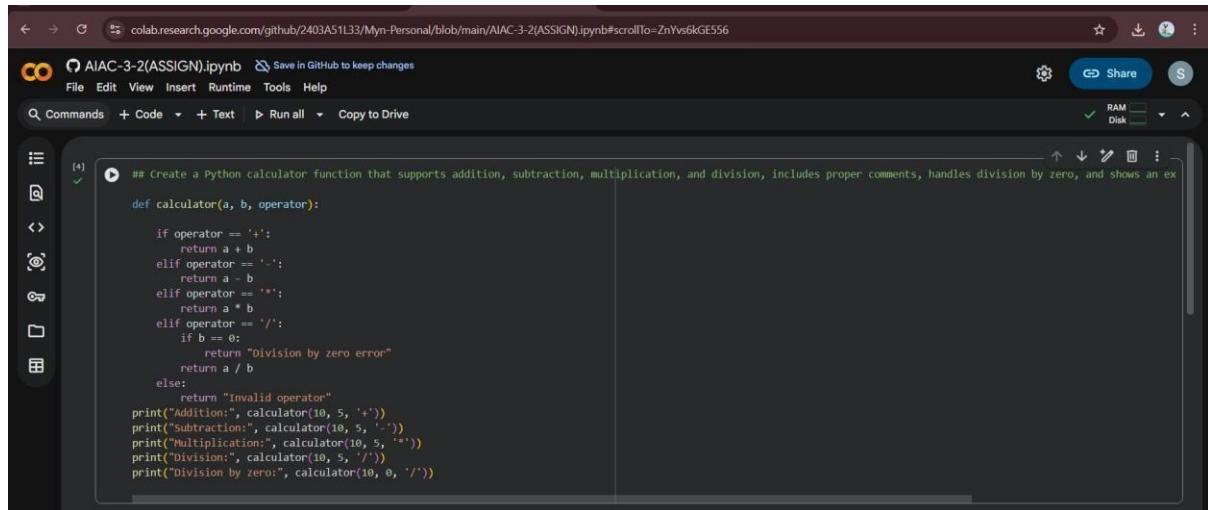
B-52

## ASSIGNMENT – 3.2

### Lab 3: Prompt Engineering – Improving Prompts and Context Management

#### Task– 1: Progressive Prompting for Calculator Design

**Prompt:** Create a Python calculator function that supports addition, subtraction, multiplication, and division, includes proper comments, handles division by zero, and shows an example of how the function is used.



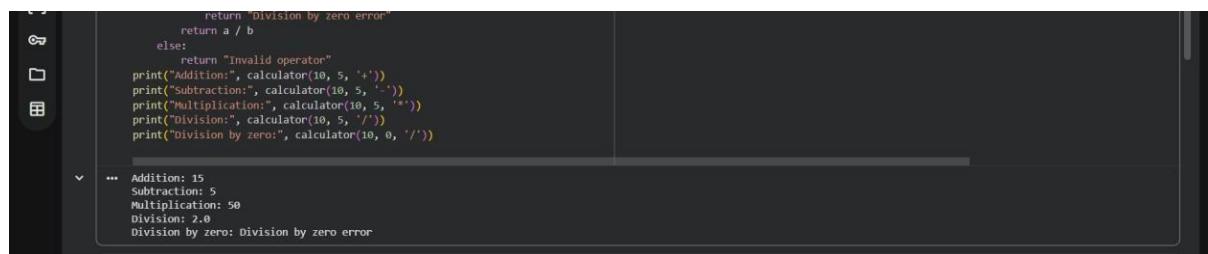
The screenshot shows a Google Colab notebook titled "AIAC-3-2(ASSIGN).ipynb". The code cell contains the following Python code:

```
## Create a Python calculator function that supports addition, subtraction, multiplication, and division, includes proper comments, handles division by zero, and shows an example of how the function is used.

def 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 "Division by zero error"
        return a / b
    else:
        return "Invalid operator"

print("Addition:", calculator(10, 5, '+'))
print("Subtraction:", calculator(10, 5, '-'))
print("Multiplication:", calculator(10, 5, '*'))
print("Division:", calculator(10, 5, '/'))
print("Division by zero:", calculator(10, 0, '/'))
```

#### OUTPUT:



The screenshot shows the output of the code execution in Google Colab. The terminal window displays the results of the print statements:

```
... Addition: 15
Subtraction: 5
Multiplication: 50
Division: 2.0
Division by zero: Division by zero error
```

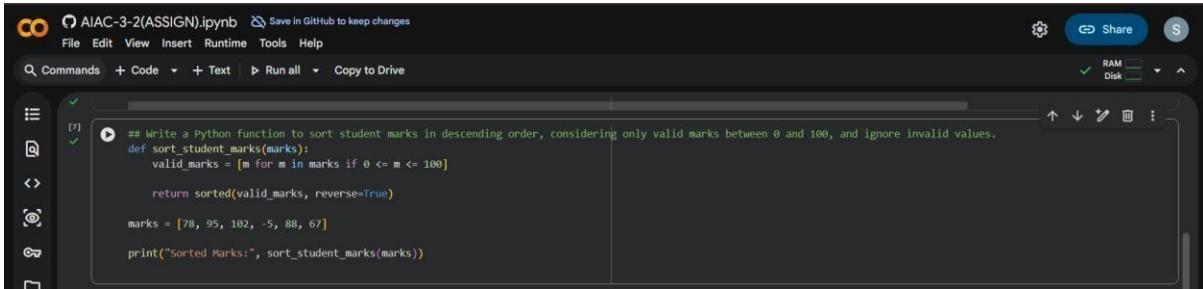
#### Explanation:

Initially, limited prompt information leads to a simple implementation. Adding comments, examples, and constraints helps the AI generate structured logic with proper error handling.

This shows how prompt refinement improves code quality.

## Task – 2: Refining Prompts for Sorting Logic

**Prompt:** Write a Python function to sort student marks in descending order, considering only valid marks between 0 and 100, and ignore invalid values.

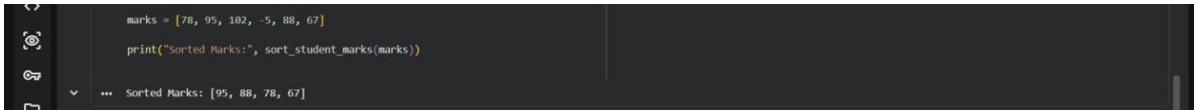


The screenshot shows a Jupyter Notebook interface with a dark theme. The top bar includes the file name "AIAC-3-2(ASSIGN).ipynb", a "Save in GitHub" button, and various menu options like File, Edit, View, Insert, Runtime, Tools, and Help. Below the menu is a toolbar with icons for Commands, Code, Text, Run all, and Copy to Drive. The main area contains a code cell with the following Python code:

```
# Write a Python function to sort student marks in descending order, considering only valid marks between 0 and 100, and ignore invalid values.
def sort_student_marks(marks):
    valid_marks = [m for m in marks if 0 <= m <= 100]
    return sorted(valid_marks, reverse=True)

marks = [78, 95, 102, -5, 88, 67]
print("Sorted Marks:", sort_student_marks(marks))
```

### OUTPUT:



The screenshot shows the output of the code execution. The code cell from the previous image is run, and the resulting output is displayed in the bottom pane:

```
marks = [78, 95, 102, -5, 88, 67]
print("Sorted Marks:", sort_student_marks(marks))
... Sorted Marks: [95, 88, 78, 67]
```

### Explanation:

A vague prompt results in generic sorting without validation.

Providing clear constraints such as order and valid range enables the AI to produce accurate and meaningful logic.

Prompt clarity removes ambiguity in implementation.

## Task– 3: Few-Shot Prompting for Prime Number Validation

**Prompt:** Using the examples (2 → True, 4 → False, 1 → False), write a Python function that checks whether a given number is prime and correctly handles edge cases.

A screenshot of a Google Colab notebook titled "AIAC-3-2(ASSIGN).ipynb". The code cell contains the following Python code:

```
print("Sorted Marks:", sort_student_marks(marks))

## Using the examples (2 → True, 4 → False, 1 → False), write a Python function that checks whether a given number is prime and correctly handles edge cases.

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
    print("Is 2 prime?", is_prime(2))
    print("Is 4 prime?", is_prime(4))
    print("Is 1 prime?", is_prime(1))
    print("Is 13 prime?", is_prime(13))
```

## OUTPUT:

The output of the code cell shows the results of the prime check function:

```
Is 2 prime? True
Is 4 prime? False
Is 1 prime? False
Is 13 prime? True
```

## Explanation:

Few-shot prompting provides clear expectations through examples. This helps the AI understand edge cases and generate correct primechecking logic.

Accuracy improves compared to zero-example prompts.

## Task– 4: Prompt-Guided UI Design for Student Grading System

**Prompt:** Create a Python program that takes student marks as input, calculates total marks, percentage, and grade, and displays the results clearly to the user.

```

## Create a Python program that takes student marks as input, calculates total marks, percentage, and grade, and displays the results clearly to the user.
marks1 = float(input("Enter marks for Subject 1: "))
marks2 = float(input("Enter marks for Subject 2: "))
marks3 = float(input("Enter marks for Subject 3: "))
marks4 = float(input("Enter marks for Subject 4: "))
marks5 = float(input("Enter marks for Subject 5: "))

total_marks = marks1 + marks2 + marks3 + marks4 + marks5

percentage = (total_marks / 500) * 100

if percentage >= 90:
    grade = "A+"
elif percentage >= 80:
    grade = "A"
elif percentage >= 70:
    grade = "B"
elif percentage >= 60:
    grade = "C"
elif percentage >= 50:
    grade = "D"
else:
    grade = "Fail"

print("\n--- Student Result ---")
print("Total Marks:", total_marks)
print("Percentage:", percentage, "%")
print("Grade:", grade)

```

Variables Terminal 12:24PM Python 3

## OUTPUT:

```

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

...
Enter marks for Subject 1: 24
Enter marks for Subject 2: 50
Enter marks for Subject 3: 48
Enter marks for Subject 4: 60
Enter marks for Subject 5: 78

--- Student Result ---
Total Marks: 260.0
Percentage: 52.0 %
Grade: D

```

## Explanation:

Clear prompt instructions guide the AI to generate a structured and interactive program.

The code correctly handles user input, calculations, and result display. Prompt guidance improves usability and readability.

## Task– 5: Analysing Prompt Specificity in Unit Conversion Functions

**Prompt:** Write two Python functions to accurately convert kilometers to miles and miles to kilo-meters using standard conversion values and clear function names.

```

File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all Copy to Drive RAM Disk
Percentage: 52.0 %
Grade: D

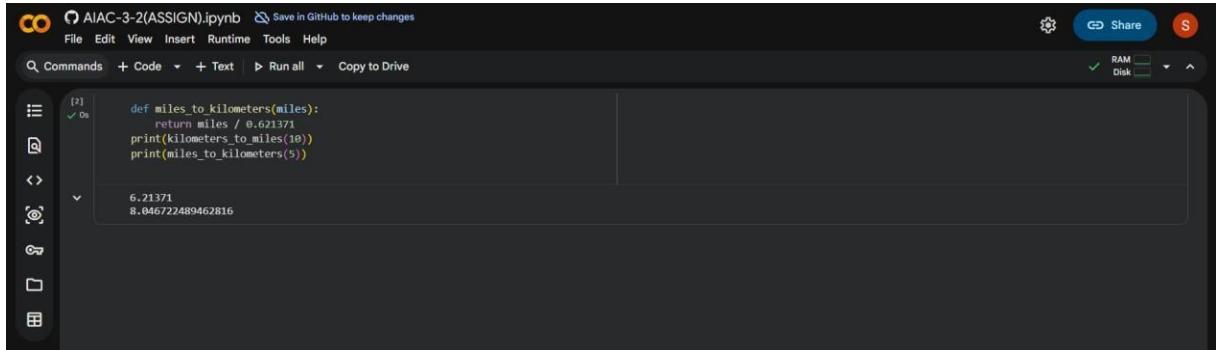
...
def kilometers_to_miles(kilometers):
    return kilometers * 0.621371

def miles_to_kilometers(miles):
    return miles / 0.621371

print(kilometers_to_miles(10))
print(miles_to_kilometers(5))

```

## OUTPUT:



The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** AIAC-3-2(ASSIGN).ipynb, Save in GitHub to keep changes
- Toolbar:** File, Edit, View, Insert, Runtime, Tools, Help
- Code Cell:** Contains the following Python code:

```
def miles_to_kilometers(miles):
    return miles / 0.621371
print(kilometers_to_miles(10))
print(miles_to_kilometers(5))
```
- Output Cell:** Displays the results of the code execution:

```
6.21371
8.046722489462816
```
- Left Sidebar:** Includes icons for file operations like New, Open, Save, and Run.
- Right Sidebar:** Shows RAM and Disk usage.

## Explanation:

Specific prompts lead to accurate and well-defined conversion functions. Clear instructions ensure correct formulas and readable function names. This demonstrates how prompt specificity improves program correctness.