SR UNIVERSITY

School of Computer Science and Artificial Intelligence

Lab Assignment # 3.2

Program	**: B. Tech (CSE)**
Specialization	**:**
Course Title	**: AI Assisted coding**
Course Code	**:**
Semester	**II**
Academic Session	**: 2025-2026**
Name of Student	**:Ganesh**
Enrollment No.	**: 2403A51L55**
Batch No.	**52**
Date	**:14-01-2026**

Submission Starts here

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.

```
def calculator(a, b, operator):
    """
    Performs basic arithmetic operations on two numbers.
    Supported operators: +, -, *, /
    """
    # Addition
    if operator == '+':
        return a + b
    # Subtraction
    elif operator == '-':
        return a - b
    # Multiplication
    elif operator == '*':
        return a * b
    # Division with zero check
    elif operator == '/':
        if b == 0:
            return "Error: Division by zero"
        return a / b
    # Invalid operator
    else:
        return "Invalid operator"
# Example usage
print(calculator(10, 5, '+'))   # Output: 15
print(calculator(10, 5, '-'))   # Output: 5
print(calculator(10, 5, '*'))   # Output: 50
print(calculator(10, 0, '/'))   # Output: Error: Division by zero
```

```
15
5
50
Error: Division by zero
```

Explanation:

1️⃣ Function Definition

```
def calculator(a, b, operator):
```

- Defines a function named **`calculator`**
- Takes **three parameters**:
  - `a` → first number
  - `b` → second number
  - `operator` → arithmetic operation (`+`, `-`, `*`, `/`)

2️⃣ Function Documentation

```
"""
Performs basic arithmetic operations on two numbers.
Supported operators: +, -, *, /
"""
```

- This is a **docstring**
- It explains **what the function does**
- Helps other programmers understand the function easily

3️⃣ Addition Operation

```
if operator == '+':
    return a + b
```

- Checks if the operator is `+`
- Adds `a` and `b`
- Returns the result

4️⃣ Subtraction Operation

```
elif operator == '-':
    return a - b
```

- Executes when operator is `-`
- Subtracts `b` from `a`

5️⃣ Multiplication Operation

```
elif operator == '*':
    return a * b
```

- Executes when operator is `*`
- Multiplies both numbers

6️⃣ Division Operation with Safety Check

```
elif operator == '/':
    if b == 0:
        return "Error: Division by zero"
    return a / b
```

- Executes when operator is `/`
- First checks if `b` is `0`
- Division by zero is **not allowed**
- Returns an error message if `b` is zero
- Otherwise, performs division

7️⃣ Invalid Operator Handling

```
else:
    return "Invalid operator"
```

- Executes when operator is **not valid**
- Prevents program crash
- Makes code more robust

▶ Example Execution

```
print(calculator(10, 5, '+'))
```

Step-by-step:

- `a = 10`
- `b = 5`
- `operator = '+'`
- Condition `operator == '+'` is true
- Output → **15**

Task2:

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

```
def sort_student_marks(marks):
    """
    Sorts student marks in descending order without modifying the original list.
    """

    # Check if list is empty
    if not marks:
        return []

    # Sort marks in descending order
    sorted_marks = sorted(marks, reverse=True)

    return sorted_marks


# Example usage
marks = [78, 92, 85, 66, 100, 45]
result = sort_student_marks(marks)

print("Original Marks:", marks)
print("Sorted Marks (Descending):", result)
```

```
Original Marks: [78, 92, 85, 66, 100, 45]
Sorted Marks (Descending): [100, 92, 85, 78, 66, 45]
```

Explanation:

1️⃣ Function Definition

```
def sort_student_marks(marks):
```

- Defines a function named `sort_student_marks`
- Accepts one parameter:
  - `marks` → list of student marks (integers)

2️⃣ Function Documentation

```
"""
Sorts student marks in descending order without modifying the
original list.
"""
```

- This docstring explains:
  - Purpose of the function
  - Sorting order (descending)
  - Original list remains unchanged

3️⃣ Empty List Check

```
if not marks:
    return []
```

- Checks whether the list is empty
- Prevents errors during sorting
- Returns an empty list if no marks are provided

4️⃣ Sorting Logic

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

- Uses Python's built-in `sorted()` function
- `reverse=True` → sorts in descending order
- `sorted()` creates a new list, so the original list is safe

5️⃣ Return Statement

```
return sorted_marks
```

- Returns the sorted list to the caller

6️⃣ Example Usage

```
marks = [78, 92, 85, 66, 100, 45]
```

- Sample list of student marks

```
result = sort_student_marks(marks)
```

- Calls the function and stores the result

```
print("Original Marks:", marks)
print("Sorted Marks (Descending):", result)
```

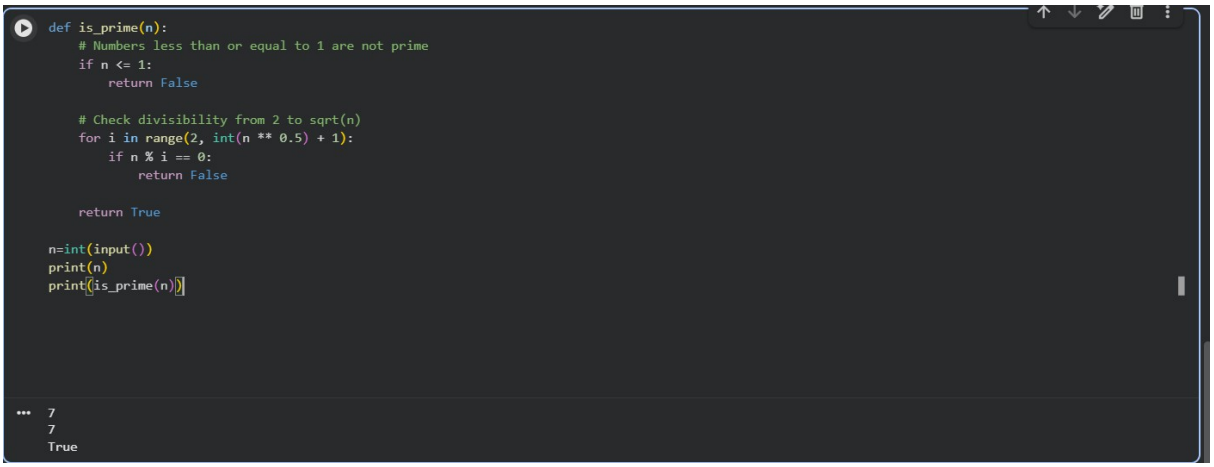
- Shows:
  - Original list (unchanged)
  - Sorted list (descending order)

▶ Sample Output

```
Original Marks: [78, 92, 85, 66, 100, 45]
Sorted Marks (Descending): [100, 92, 85, 78, 66, 45]
```

Task3:

- 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.



```
def is_prime(n):
    # Numbers less than or equal to 1 are not prime
    if n <= 1:
        return False

    # Check divisibility from 2 to sqrt(n)
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            return False

    return True

n=int(input())
print(n)
print(is_prime(n))
```

```
7
7
True
```

Explanation:

1 Handle Edge Cases

```
if n <= 1:
    return False
```

- Prime numbers must be greater than 1
- Eliminates wrong results for 0 and 1

---

2 Efficient Loop

```
for i in range(2, int(n ** 0.5) + 1):
```

- Checks factors only up to √n
- Improves performance

3 Divisibility Check

```
if n % i == 0:
    return False
```

- If divisible, number is not prime

---

4 Final Return

```
return True
```

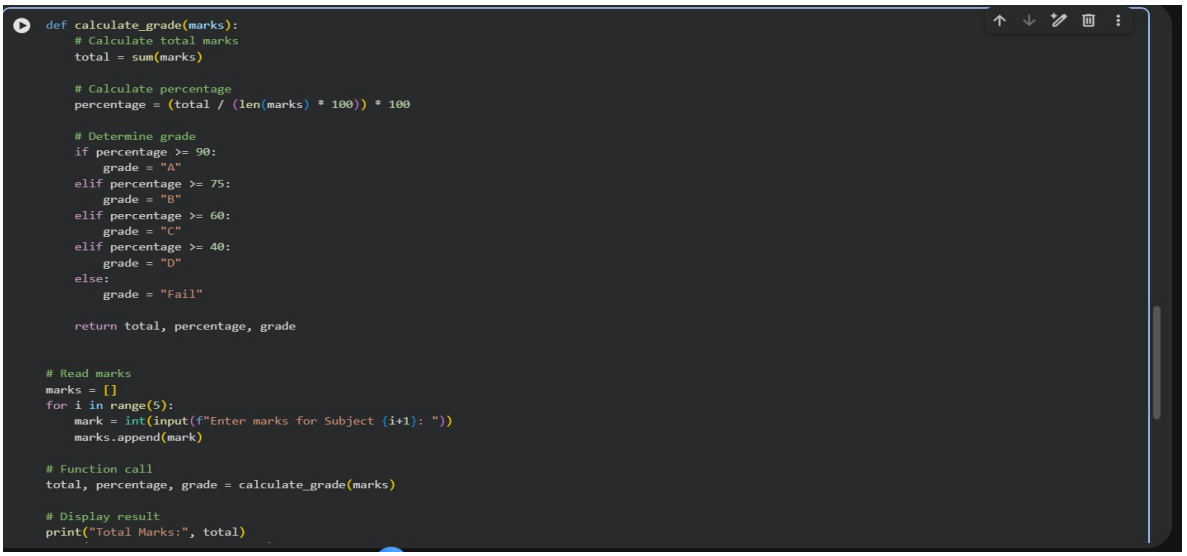
- If no divisors found → number is prime

▶ Example Test Cases

```
print(is_prime(7))    # True
print(is_prime(4))    # False
print(is_prime(1))    # False
print(is_prime(17))   # True
print(is_prime(20))   # False
```

Task4:

- 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.



```
def calculate_grade(marks):
    # Calculate total marks
    total = sum(marks)

    # Calculate percentage
    percentage = (total / (len(marks) * 100)) * 100

    # Determine grade
    if percentage >= 90:
        grade = "A"
    elif percentage >= 75:
        grade = "B"
    elif percentage >= 60:
        grade = "C"
    elif percentage >= 40:
        grade = "D"
    else:
        grade = "Fail"

    return total, percentage, grade


# Read marks
marks = []
for i in range(5):
    mark = int(input(f"Enter marks for Subject {i+1}: "))
    marks.append(mark)

# Function call
total, percentage, grade = calculate_grade(marks)

# Display result
print("Total Marks:", total)
```

```
# Read marks
marks = []
for i in range(5):
    mark = int(input(f"Enter marks for Subject {i+1}: "))
    marks.append(mark)

# Function call
total, percentage, grade = calculate_grade(marks)

# Display result
print("Total Marks:", total)
print("Percentage:", percentage)
print("Grade:", grade)
```

```
Enter marks for Subject 1: 75
Enter marks for Subject 2: 98
Enter marks for Subject 3: 89
Enter marks for Subject 4: 97
Enter marks for Subject 5: 93
Total Marks: 452
Percentage: 90.4
Grade: A
```

Explanation:

1 Function Definition

```
def calculate_grade(marks):
```

- Accepts a list of subject marks

---

2 Total Calculation

```
total = sum(marks)
```

- Adds all subject marks

---

3 Percentage Calculation

```
percentage = (total / (len(marks) * 100)) * 100
```

- Assumes each subject is out of 100

---

4 Grade Assignment

- Uses `if-elif-else` conditions
- Assigns grade based on percentage

---

5 Function Return

```
return total, percentage, grade
```

- Returns all results together

---

6 Function Call

```
total, percentage, grade = calculate_grade(marks)
```

- Calls the function and stores results

Task5:

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

```
def convert_distance(value, unit):
    """
    Converts distance between kilometers and miles.
    unit = 'km'   → kilometers to miles
    unit = 'mile' → miles to kilometers
    """

    if value < 0:
        return "Distance cannot be negative"

    if unit == "km":
        return value * 0.621371
    elif unit == "mile":
        return value * 1.60934
    else:
        return "Invalid unit"
print(convert_distance(10, "km"))     # 6.21371 miles
print(convert_distance(5, "mile"))   # 8.0467 km
print(convert_distance(-3, "km"))    # Error
print(convert_distance(7, "meter"))  # Invalid unit
```

```
6.21371
8.0467
Distance cannot be negative
Invalid unit
```

Explanation:

1 Function Definition

```
def convert_distance(value, unit):
```

- Defines a function named **`convert_distance`**
- Takes two parameters:
  - `value` → numerical distance
  - `unit` → type of input unit (`"km"` or `"mile"`)

2️⃣ Function Description (Docstring)

```
"""
Converts distance between kilometers and miles.
"""
```

- Explains the purpose of the function
- Helps users understand expected input and output

3️⃣ Negative Value Check

```
if value < 0:
    return "Distance cannot be negative"
```

- Distance cannot be negative in real life
- This check avoids incorrect results
- Shows how **constraints improve correctness**

4️⃣ Kilometers to Miles Conversion

```
if unit == "km":
    return value * 0.621371
```

- Uses the standard formula:
    - **1 km = 0.621371 miles**
- Executes when input unit is `"km"`

5️⃣ Miles to Kilometers Conversion

```
elif unit == "mile":
    return value * 1.60934
```

- Uses the formula:

- **1 mile = 1.60934 km**

- Executes when input unit is `"mile"`

6️⃣ Invalid Unit Handling

```
else:
    return "Invalid unit"
```

- Runs if unit is not `"km"` or `"mile"`
- Prevents incorrect conversions
- Makes the function more reliable

▶ Example Execution

```
print(convert_distance(10, "km"))
```

Execution Flow:

- `value = 10`
- `unit = "km"`
- Negative check → passed
- `"km"` condition is true
- Output → **6.21371 miles**