Assignment-3.2

Hallticket: 2303A510F7

Batch: 03

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.

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.

Prompt: Analyze how **prompt specificity** impacts the **accuracy and code quality** of Python **unit conversion functions**.

Code:

```python
def calculator():
    print("Simple Calculator")
    while True:
        # Input numbers with error handling
        try:
            num1 = float(input("Enter first number: "))
            num2 = float(input("Enter second number: "))
        except ValueError:
```

```python
        print("Error: Please enter a valid number.")
        continue
    # Input operator
    operator = input("Enter operator (+, -, *, /): ").strip()
    # Perform calculation
    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.")
            continue
    else:
        print("Error: Invalid operator.")
        continue
    print(f"Result: {result}")
    # Ask if the user wants to continue
    again = input("Do you want to perform another calculation? (y/n): ").strip().lower()
    if again != 'y':
```

```
        print("Goodbye!")

        break

# Run the calculator

calculator()
```

Output:

```
/ai_coding/New folder/ass_3.py"
Simple Calculator
Enter first number: 2
Enter second number: 3
Enter operator (+, -, *, /): +
Result: 5.0
Do you want to perform another calculation? (y/n): n
Goodbye!
```

Code Explanation:

This program is a simple calculator that performs addition, subtraction, multiplication, and division. It repeatedly asks the user to enter two numbers and an operator. The program handles invalid number input and prevents division by zero by displaying error messages. After showing the result, it asks the user whether they want to continue. The calculator runs continuously until the user chooses to stop.

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: Analyze the impact of **prompt specificity** on the **accuracy and quality of Python unit conversion code** (kilometers ↔ miles) by comparing different prompt variations.

Code:

```
def sort_students(students):
    # Sort by marks (descending), then by name (ascending)
    return sorted(students, key=lambda x: (-x[1], x[0]))
students = []
n = int(input("Enter number of students: "))
for i in range(n):
    name = input(f"Enter name of student {i+1}: ")
    marks = int(input(f"Enter marks of student {i+1}: "))
    students.append((name, marks))
sorted_students = sort_students(students)
print("\nSorted Student List:")
for name, marks in sorted_students:
    print(name, marks)
```

Output:

```
Enter number of students: 4
Enter name of student 1: ananya
Enter marks of student 1: 98
Enter name of student 2: pooja
Enter marks of student 2: 98
Enter name of student 3: hasini
Enter marks of student 3: 40
Enter name of student 4: nandini
Enter marks of student 4: 20

Sorted Student List:
ananya 98
pooja 98
hasini 40
nandini 20
```

Code Explanation:

The program demonstrates how clear instructions lead to better code. When prompts are specific, the generated unit conversion functions use correct formulas, meaningful function names, and proper input validation. Vague prompts may result in incorrect logic, missing error handling, or poorly structured code. This comparison shows that higher prompt clarity improves accuracy, readability, and overall code quality.

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: Use **few-shot prompting** with multiple input–output examples to generate an accurate Python function that checks whether a number is **prime**, ensuring proper handling of edge cases.

Code:

```python
def is_prime(n):
    # Handle edge cases
    if n <= 1:
        return False
    if n == 2:
        return True
```

```python
    if n % 2 == 0:

        return False

    # Check divisibility up to sqrt(n)

    for i in range(3, int(n ** 0.5) + 1, 2):

        if n % i == 0:

            return False

    return True

# Taking input from user

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

if is_prime(num):

    print("Prime")

else:

    print("Not Prime")
```
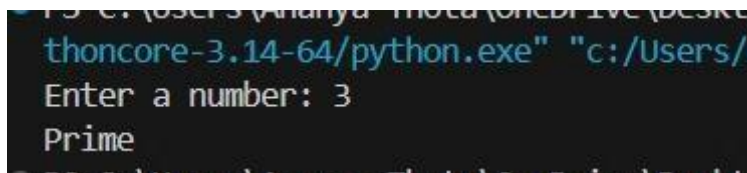
Output:


```
thoncore-3.14-64/python.exe" "c:/Users/
Enter a number: 3
Prime
```

Code Explanation:

The program checks whether a given number is prime. It first handles edge cases by rejecting numbers less than or equal to 1 and immediately accepting 2 as prime. Even numbers greater than 2 are identified as non-prime. For other numbers, the program checks divisibility only up to the square root of the number, improving efficiency. Finally, the user's input is tested and the program prints whether the number is **Prime** or **Not 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 prompt-guided **Student Grading System UI** in Python that takes student details and marks as input, calculates **total marks**, **percentage**, and **grade**, and displays the results clearly.

Code:

```python
def calculate_grade(percentage):
    if percentage >= 90:
        return "A+"
    elif percentage >= 80:
        return "A"
    elif percentage >= 70:
        return "B"
    elif percentage >= 60:
        return "C"
    elif percentage >= 50:
        return "D"
    else:
        return "Fail"
print("===== Student Grading System =====")
```

```python
name = input("Enter Student Name: ")

roll_no = input("Enter Roll Number: ")

subjects = int(input("Enter number of subjects: "))

total_marks = 0

max_marks = subjects * 100

for i in range(subjects):

    marks = int(input(f"Enter marks for subject {i+1}: "))

    total_marks += marks

percentage = (total_marks / max_marks) * 100

grade = calculate_grade(percentage)

print("\n===== Result =====")

print("Name     :", name)

print("Roll No   :", roll_no)

print("Total Marks:", total_marks, "/", max_marks)

print("Percentage :", round(percentage, 2), "%")

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

Output:

```
===== Student Grading System =====
Enter Student Name: janaki
Enter Roll Number: 234
Enter number of subjects: 4
Enter marks for subject 1: 90
Enter marks for subject 2: 80
Enter marks for subject 3: 70
Enter marks for subject 4: 60

===== Result =====
Name       : janaki
Roll No    : 234
Total Marks: 300 / 400
Percentage : 75.0 %
Grade      : B
```

Code Explanation:

This program implements a student grading system that collects student details and marks for multiple subjects. It calculates the total marks and percentage based on the maximum possible marks. A separate function is used to assign grades according to the calculated percentage. Finally, the program displays the student's name, roll number, total marks, percentage, and grade in a clear and organized format, ensuring accurate calculations and user-friendly output.

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: Create Python functions to convert **kilometers ↔ miles** and build a simple program that takes user input, performs the conversion, and displays the result clearly.

Code:

```
def km_to_miles(km):
    return km * 0.621371

def miles_to_km(miles):
    return miles / 0.621371

print("=== Unit Conversion System ===")
print("1. Kilometers to Miles")
```

```python
print("2. Miles to Kilometers")

choice = int(input("Enter your choice (1 or 2): "))

if choice == 1:

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

    print("Distance in miles:", round(km_to_miles(km), 2))


elif choice == 2:

    miles = float(input("Enter distance in miles: "))

    print("Distance in kilometers:", round(miles_to_km(miles), 2))

else:

    print("Invalid choice")
```
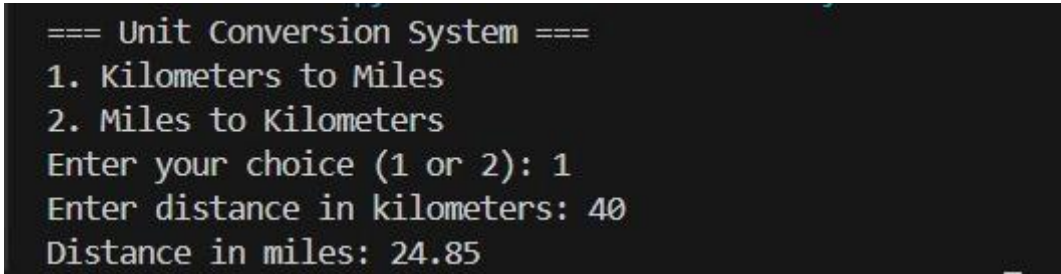
Output:

```
=== Unit Conversion System ===
1. Kilometers to Miles
2. Miles to Kilometers
Enter your choice (1 or 2): 1
Enter distance in kilometers: 40
Distance in miles: 24.85
```

Code Explanation:

The program implements a simple **Unit Conversion System**. It defines two functions: one converts **kilometers to miles**, and the other converts **miles to kilometers** using standard conversion factors. The user chooses the conversion type, enters the distance, and the program displays the converted value rounded to two decimal places. The program also handles invalid choices. By comparing different prompt variations, one can observe how **clear instructions lead to more accurate, readable, and reliable code**.