AI ASSISTED CODING – LAB ASSIGNMENT 3.2

NAME: LOKESH
HTNO:2303A51079

Task 1: Progressive Prompting – Calculator Design Aim

To understand how improving the prompt step by step affects the quality and correctness of AI-generated code.

Step-1: Initial Prompt Prompt

Create a calculator function.

AI Generated Code

python Copy code

```python
def calc(a, b):
    return a + b
```

Step-2: Refined Prompt Prompt

Create a calculator program in Python that performs addition, subtraction, multiplication and division.

AI Generated Code

python Copy code

```python
def calculator(a, b, op):
    if op == '+':
        return a + b
    elif op == '-':
        return a - b
    elif op == '*':
        return a * b
    elif op == '/':
        return a / b
```

Step-3: Final Prompt (Context Managed Prompt) Prompt

Write a complete Python calculator program with user input, menu-driven operation selection, and proper error handling.

AI Generated Code

python Copy code

```python
def calculator():
    a = float(input("Enter first number: "))
    b = float(input("Enter second number: "))
    op = input("Choose operation (+, -, *, /): ")

    if op == '+':
        print("Result:", a + b)
    elif op == '-':
        print("Result:", a - b)
    elif op == '*':
        print("Result:", a * b)
    elif op == '/':
        if b != 0:
            print("Result:", a / b)
        else:
            print("Division by zero error")
    else:
        print("Invalid operation")
```

Comparison

Feature	Initial Prompt	Refined Prompt	Final Prompt
Operations	Only addition	All four	All four
User input	No	No	Yes
Error handling	No	No	Yes
Program quality	Poor	Medium	Very Good

Observation

When the prompt was small, the code was also very simple.
As the prompt became clearer and more detailed, the generated code became better and more complete.
The final prompt gave a proper working calculator program.

Conclusion

This task shows that writing good and clear prompts is very important when using AI for coding.

Better prompts produce better programs with fewer mistakes.

Task 2: Refining Prompts for Sorting Student Marks Aim

To study how improving the clarity of a prompt helps in generating better and more accurate sorting logic.

Step-1: Initial (Vague) Prompt Prompt

Sort the student marks.

AI Generated Code

python Copy code

```python
def sort_marks(arr):
    return arr
```

Step-2: Refined Prompt Prompt

Write a Python program to sort student marks in ascending order without using built-in functions.

AI Generated Code

python Copy code

```python
def sort_marks(arr):
    for i in range(len(arr)):
        for j in range(i+1, len(arr)):
            if arr[i] > arr[j]:
                arr[i], arr[j] = arr[j], arr[i]
    return arr
```

Observation

The vague prompt produced almost no useful code.
After improving the prompt with clear instructions, the AI generated a correct and efficient sorting program.

Conclusion

This task shows that when we explain clearly what we want, the AI gives much better results. Clear prompts help avoid mistakes and improve code quality.

Task 3: Few-Shot Prompting for Prime Number Validation Aim

To observe how giving examples in the prompt improves the correctness of the AIgenerated prime number program.

Step-1: Few-Shot Prompt Prompt with Examples

2 → Prime
4 → Not Prime
17 → Prime
1 → Not Prime
Write a Python function to check whether a number is prime.

AI Generated Code

python Copy code

```python
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
```

Observation

Because examples were given, the AI understood the problem clearly.
It handled edge cases like 1 and used an efficient method for checking prime numbers.

Conclusion

Few-shot prompting helps the AI generate more correct and reliable code. Providing examples makes the AI understand the requirement better.

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

To create a simple user interface program for calculating total marks, percentage, and grade using clear prompts.

Prompt Used

Create a Python program that takes student details and marks, calculates total, percentage, and assigns grade, and displays the result clearly.

AI Generated Code

```python
name = input("Enter student name: ")

marks = []
for i in range(5):
    marks.append(int(input(f"Enter marks of subject {i+1}: ")))

total = sum(marks)
percentage = total / 5

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

print("\nStudent Name:", name)
print("Total Marks:", total)
print("Percentage:", percentage)
print("Grade:", grade)
```

Sample Output

Name: Lokesh

Total Marks: 469
Percentage: 89
Grade: B

Observation

Because the prompt was clear, the AI generated a complete program with correct calculations and easy-to-read output.

Conclusion

Clear prompts help in building well-structured programs with fewer errors. The program became simple and useful for real-world use.

Task 5: Analyzing Prompt Specificity in Unit Conversion Functions

Aim

To study how giving clear and specific instructions in a prompt improves the accuracy of AI-generated unit conversion programs.

Prompt Used

Write a Python program to convert kilometers to miles and miles to kilometers using correct formulas and proper function structure.

AI Generated Code

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

def miles_to_km(miles):
    return miles / 0.621371

km = float(input("Enter distance in kilometers: "))
print("Miles:", km_to_miles(km))

miles = float(input("Enter distance in miles: "))
print("Kilometers:", miles_to_km(miles))
```

Sample Output

Enter distance in kilometers: 10 Miles:
6.21371

Enter distance in miles: 5
Kilometers: 8.04672

Observation

Because the prompt was very clear about the task and formulas, the AI produced accurate and well-organized code.

Conclusion

When the prompt is specific and detailed, the generated code becomes more correct and easier to use.