

AI ASSISTANT CODING ASSIGNMENT - 3.2

NAME : MOGILI KEERTHI

HT.NO : 2303A51427

BATCH : 21

Lab 3: Prompt Engineering – Improving Prompts and Context Management

Lab Objectives:

- To understand how prompt structure and wording influence AI-generated code.
- To explore how context (like comments and function names) helps AI generate relevant output.
- To evaluate the quality and accuracy of code based on prompt clarity.
- To develop effective prompting strategies for AI-assisted programming.

Lab Outcomes (LOs):

After completing this lab, students will be able to:

- Generate Python code using Google Gemini in Google Colab.
 - Analyze the effectiveness of code explanations and suggestions by Gemini.
 - Set up and use Cursor AI for AI-powered coding assistance.
 - Evaluate and refactor code using Cursor AI features.
 - Compare AI tool behavior and code quality across different platforms.
-

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.
-

CODE :

Prompt: write a python function named calculator

```
lab-3.2.py > calculator
1  #write a python function named calculator
2  def calculator(num1, num2, operation):
3      if operation == 'add':
4          return num1 + num2
5      elif operation == 'subtract':
6          return num1 - num2
7      elif operation == 'multiply':
8          return num1 * num2
9      elif operation == 'divide':
10         if num2 != 0:
11             return num1 / num2
12         else:
13             return "Error: Division by zero"
14     else:
15         return "Error: Invalid operation"
16 # Example usage:
17 print(calculator(10, 5, 'add'))      # Output: 15
18 print(calculator(10, 5, 'subtract')) # Output: 5
19
20
PS C:\Users\keert\OneDrive\Documents\AI Assistance\lab> & "C:/Program Files/Python313/python.exe" "c:/Users/keert/OneDrive/Doc
15
5
PS C:\Users\keert\OneDrive\Documents\AI Assistance\lab>
```

CODE AFTER ENHANCING THE PROMPT :

```

print(calculator(10, 5, 'subtract'))    # Output: 5
#write a python function named calculator
#the function should work as a simple calculator
#it should take two numbers and an operation.
def calculator(num1, num2, operation):
    if operation == 'add':
        return num1 + num2
    elif operation == 'subtract':
        return num1 - num2
    elif operation == 'multiply':
        return num1 * num2
    elif operation == 'divide':
        if num2 != 0:
            return num1 / num2
        else:
            return "Error: Division by zero"
    else:
        return "Error: Invalid operation"
# Example usage:
print(calculator(10, 5, 'add'))          # Output: 15
print(calculator(10, 5, 'subtract'))     # Output: 5
print(calculator(10, 5, 'multiply'))     # Output: 50

```

OUTPUT :

```

PS C:\Users\keert\OneDrive\Documents\AI Assistance\lab> & "C:/Program Files/Python313/python.exe" "c:/Users/keert/OneDrive/
15
5
50
PS C:\Users\keert\OneDrive\Documents\AI Assistance\lab>

```

Final Observation :

At first, when only the function name was given, the AI generated a very basic and incomplete calculator function with little or no logic. After adding comments, the AI started including parameters and arithmetic operations. When usage examples were finally added, the AI produced a complete and well-structured calculator program with proper conditions and error handling. This clearly shows that progressive prompting improves both the logic and structure of the generated code.

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.**
-

STAGE 1 :

Prompt: Generate a code for sorting student marks.

Code :

```
40
41 #generate a code for sorting student marks.
42 def sort_student_marks(marks):
43     | return sorted(marks)
44 # Example usage:
45 student_marks = [88, 92, 79, 85, 95]
46 sorted_marks = sort_student_marks(student_marks)
47 print("Sorted student marks:", sorted_marks) # Output: [79, 85, 88, 92, 95]
```

Output :

```
PS C:\Users\keert\OneDrive\Documents\AI Assistance\lab> & "C:/Program F
Sorted student marks: [79, 85, 88, 92, 95]
PS C:\Users\keert\OneDrive\Documents\AI Assistance\lab>
```

STAGE 2 :

Code :

```
# write a python function that takes an array of student marks (integers) and sort them in descending order using a suitable sorting technique.
def sort_student_marks(marks):
    return sorted(marks, reverse=True)
#give an user input example for above function
if __name__ == "__main__":
    student_marks = [88, 92, 79, 85, 95]
    sorted_marks = sort_student_marks(student_marks)
    print("Sorted student marks in descending order:", sorted_marks) # Output: [95, 92, 88, 85, 79]
```

Output :

```
PS C:\Users\keert\OneDrive\Documents\AI Assistance\lab> C:\Program Files\Python3.9\python.exe C:\Users\keert\OneDrive\Documents\AI Assistance\lab\lab-3.2.py
Sorted student marks in descending order: [95, 92, 88, 85, 79]
PS C:\Users\keert\OneDrive\Documents\AI Assistance\lab>
```

STAGE 3 :

Code :

```
# lab-3.2.py 7 ...
73 #Write an optimized python function to sort an array of student marks in descending order
74 # Constraints:
75 # Marks range from 0 to 100
76 # Do not use built-in sort functions
77 # Time complexity should be efficient
78 #Handle duplicate marks properly
79 def counting_sort_descending(marks):
80     # Create a count array to store the count of each mark
81     count = [0] * 101 # Marks range from 0 to 100
82
83     # Store the count of each mark
84     for mark in marks:
85         count[mark] += 1
86
87     # Build the sorted array in descending order
88     sorted_marks = []
89     for mark in range(100, -1, -1):
90         sorted_marks.extend([mark] * count[mark])
91
92     return sorted_marks
93 #user input example for the function
94 if __name__ == "__main__":
95     student_marks = [88, 92, 79, 85, 95, 92, 100, 67, 85]
96     sorted_marks = counting_sort_descending(student_marks)
97     print("Sorted student marks in descending order:", sorted_marks) # Output: [100, 95, 92, 92, 88, 85, 85, 79, 67]
```

Output :

```
PS C:\Users\keert\OneDrive\Documents\AI Assistance\lab> C:\Program Files\Python3.9\python.exe C:\Users\keert\OneDrive\Documents\AI Assistance\lab\lab-3.2.py
Sorted student marks in descending order: [100, 95, 92, 92, 88, 85, 85, 79, 67]
PS C:\Users\keert\OneDrive\Documents\AI Assistance\lab>
```

Final Observation :

the AI produced a simple sorting solution without clear direction or constraints. After refining the prompt to specify sorting order, the output became more accurate and meaningful. When clear constraints and examples were added, the AI generated a more

structured and efficient sorting function. This demonstrates that refining prompts helps the AI move from ambiguous logic to a correct and reliable implementation.

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.
-

STAGE 1 :

Code :

Prompt: Write a python code to check if a number is prime or not

```
98
99  #write a python code to check if a number is prime
100 def is_prime(num):
101     if num <= 1:
102         return False
103     for i in range(2, int(num**0.5) + 1):
104         if num % i == 0:
105             return False
106     return True
107 # Example usage
108 number = 29
109 if is_prime(number):
110     print(f"{number} is a prime number.")
111 else:
112     print(f"{number} is not a prime number.")
113
```

Output :

```
PS C:\Users\keert\OneDrive\Documents\AI Assistance\lab> & "C:/Program F
thon313/python.exe" "c:/Users/keert/OneDrive/Documents/AI Assistance/la
.2.py"
29 is a prime number.
PS C:\Users\keert\OneDrive\Documents\AI Assistance\lab>
```

STAGE 2

Code :

```
prime.py > ...
1  # Write a Python function is_prime(n) that returns True if n is prime, else False.
2  # Examples:
3  # is_prime(2) → True
4  # is_prime(3) → True
5  # is_prime(4) → False
6  # is_prime(9) → False
7  # is_prime(11) → True
8  def is_prime(n):
9      """
10     Check if a number is prime.
11     A prime number is greater than 1 and has no divisors other than 1 and itself.
12     """
13     if n <= 1:
14         return False
15     if n <= 3:
16         return True
17     if n % 2 == 0 or n % 3 == 0:
18         return False
19
20     i = 5
21     while i * i <= n:
22         if n % i == 0 or n % (i + 2) == 0:
23             return False
24         i += 6
25     return True
26 # example usage
27 if __name__ == "__main__":
28     test_numbers = [2, 3, 4, 9, 11, 15, 17, 18, 19, 20]
```

```
23         return False
24         i += 6
25     return True
26 # example usage
27 if __name__ == "__main__":
28     test_numbers = [2, 3, 4, 9, 11, 15, 17, 18, 19, 20]
29     for number in test_numbers:
30         result = is_prime(number)
31         print(f"is_prime({number}) → {result}")
```

Output :

```
(.venv) PS D:\AIASSCoding> & D:/AIASSCoding/.venv/Scripts/python.exe d:/AIASSCoding/prime.py
is_prime(11) → True
is_prime(15) → False
is_prime(17) → True
is_prime(18) → False
is_prime(19) → True
is_prime(20) → False
(.venv) PS D:\AIASSCoding>
```

STAGE 3 :

Code :

```
115 # write a Python function is_prime(n) that returns True if n is prime, else False.
116 # Examples:
117 #is_prime(2) → True
118 # is_prime(3) → True
119 #is_prime(4) → False
120 # is_prime(9) → False
121 #is_prime(11) → True
122 def is_prime(n):
123     if n <= 1:
124         return False
125     if n <= 3:
126         return True
127     if n % 2 == 0 or n % 3 == 0:
128         return False
129     i = 5
130     while i * i <= n:
131         if n % i == 0 or n % (i + 2) == 0:
132             return False
133         i += 6
134     return True
135 # Example usage
136 number = 11
137 if is_prime(number):
138     print(f"{number} is a prime number.")
139 else:
140     print(f"{number} is not a prime number.")
```

Output :

```
PS C:\Users\keert\OneDrive\Documents\AI Assistance\lab> &
11 is a prime number.
PS C:\Users\keert\OneDrive\Documents\AI Assistance\lab>
```

Final Observation :

In the first stage, the AI generated a basic prime-checking function due to an unclear prompt and missed edge cases. Providing examples in the second stage improved the correctness of the logic. In the third stage, few-shot prompting with edge-case examples resulted in a more accurate and efficient solution. The generated code correctly handles negative numbers, 0, and 1. Overall, few-shot prompting significantly improves the reliability and quality of AI-generated code.

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.**
-

STAGE 1:

Code :

```

141
142 #create a user interface to grade students based on their marks
143 def grade_student(marks):
144     if marks >= 90:
145         return 'A'
146     elif marks >= 80:
147         return 'B'
148     elif marks >= 70:
149         return 'C'
150     elif marks >= 60:
151         return 'D'
152     else:
153         return 'F'
154 #user input example for the function
155 if __name__ == "__main__":
156     user_input = int(input("Enter student marks (0-100): "))
157     try:
158         marks = float(user_input)
159         if 0 <= marks <= 100:
160             grade = grade_student(marks)
161             print(f"The grade for marks {marks} is: {grade}")
162         else:
163             print("Please enter marks between 0 and 100.")
164     except ValueError:
165         print("Invalid input. Please enter a numeric value for marks.")

```

Output :

```

11 IS a prime number.
PS C:\Users\keert\OneDrive\Documents\AI Assistance\lab> & "C:/Program Files/Python39/python.exe" C:\Users\keert\OneDrive\Documents\AI Assistance\lab\grade_student.py
Enter student marks (0-100): 56
The grade for marks 56.0 is: F
PS C:\Users\keert\OneDrive\Documents\AI Assistance\lab>

```

STAGE 2 :

Code :

```

167 #create a student grading system UI that accepts marks from the user
168 #calculates total marks and percentage, and displays the result.
169 def calculate_grade(marks):
170     if marks >= 90:
171         return 'A'
172     elif marks >= 80:
173         return 'B'
174     elif marks >= 70:
175         return 'C'
176     elif marks >= 60:
177         return 'D'
178     else:
179         return 'F'
180 #example usage
181 if __name__ == "__main__":
182     user_input = input("Enter student marks separated by spaces: ")
183     marks_list = list(map(float, user_input.split()))
184     total_marks = sum(marks_list)
185     print(f"Total Marks: {total_marks}")
186     print(f"Percentage: {total_marks / len(marks_list):.2f}%")
187     print(f"Grade: {calculate_grade(total_marks / len(marks_list))}")
188

```

Output :

```

thon313/python.exe" "c:/Users/keert/OneDrive/Documents/AI Ass
.2.py"
Enter student marks separated by spaces: 23 24 19 36 89
Total Marks: 191.0
Percentage: 38.20%
Grade: F
PS C:\Users\keert\OneDrive\Documents\AI Assistance\lab>

```

STAGE 3 :

Code :

```

89 # Create a well-structured UI for a student grading system.
90 #Allow users to enter marks for 5 subjects.
91 # Calculate total marks, percentage, and assign grades
92 #(A: 290, B: 275, C: 250, Fail: <50).
93 #Display all results clearly.
94 def calculate_grade(total_marks):
95     if total_marks >= 290:
96         return 'A'
97     elif total_marks >= 275:
98         return 'B'
99     elif total_marks >= 250:
100         return 'C'
101     else:
102         return 'Fail'
103     return total_marks,percentage , calculate_grade
104 # user input and display results
105 if __name__ == "__main__":
106     print("Student Grading System")
107     marks_list = []
108     for i in range(1, 6):
109         while True:
110             try:
111                 mark = float(input(f"Enter marks for subject {i} (0-100): "))
112                 if 0 <= mark <= 100:
113                     marks_list.append(mark)

```

```

        while True:
            try:
                mark = float(input(f"Enter marks for subject {i} (0-100): "))
                if 0 <= mark <= 100:
                    marks_list.append(mark)
                    break
            except ValueError:
                print("Please enter marks between 0 and 100.")
            print("Invalid input. Please enter a numeric value for marks.")
total_marks = sum(marks_list)
percentage = (total_marks / 500) * 100
grade = calculate_grade(total_marks)

print("\n----- Result -----")
print(f"Total Marks: {total_marks}/500")
print(f"Percentage: {percentage:.2f}%")
print(f"Grade: {grade}")

```

Output :

```
PS C:\Users\keert\OneDrive\Documents\AI Assistance\lab> & "C:/Program File
Student Grading System
Enter marks for subject 1 (0-100): 53
Enter marks for subject 2 (0-100): 38
Enter marks for subject 3 (0-100): 47
Enter marks for subject 4 (0-100): 87
Enter marks for subject 5 (0-100): 56

----- Result -----
Total Marks: 281.0/500
Percentage: 56.20%
Grade: B
PS C:\Users\keert\OneDrive\Documents\AI Assistance\lab> █
```

Final Observation :

In the first stage, the AI generated a basic UI due to a vague prompt with limited functionality. In the second stage, clearer instructions improved the structure and calculation logic of the grading system. In the third stage, a detailed prompt resulted in a well-organized UI with accurate total, percentage, and grade calculation.

The outputs were clearly displayed and easy to understand.

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.
-

STAGE 1:

Code :

```
27
28 #write a python function to convert distance units.
29 def convert_distance(value, from_unit, to_unit):
30     # Conversion factors to meters
31     conversion_factors = {
32         'meters': 1,
33         'kilometers': 1000,
34         'miles': 1609.34,
35         'feet': 0.3048,
36         'inches': 0.0254
37     }
38     if from_unit not in conversion_factors or to_unit not in conversion_factors:
39         return "Error: Invalid unit"
40
41     # Convert from the original unit to meters
42     value_in_meters = value * conversion_factors[from_unit]
43
44     # Convert from meters to the target unit
45     converted_value = value_in_meters / conversion_factors[to_unit]
46     return converted_value
47
48 # Example usage
49 if __name__ == "__main__":
50     distance = 5
51     from_unit = 'kilometers'
52     to_unit = 'miles'
53     result = convert_distance(distance, from_unit, to_unit)
54     print(f"{distance} {from_unit} is equal to {result:.2f} {to_unit}")
```

Output :

```
PS C:\Users\keert\OneDrive\Documents\AI Assistance\lab> & "C:/Prog
5 kilometers is equal to 3.11 miles
PS C:\Users\keert\OneDrive\Documents\AI Assistance\lab> █

0 ^ 0
```

STAGE 2:

Code :

```

5 #write a python function that converts kilometers to miles and miles to kilometers.
6 def convert_distance(value, from_unit, to_unit):
7     conversion_factors = {
8         'kilometers_to_miles': 0.621371,
9         'miles_to_kilometers': 1.60934
10    }
11    key = f"{from_unit}_to_{to_unit}"
12    if key not in conversion_factors:
13        return "Error: Invalid conversion units"
14
15    converted_value = value * conversion_factors[key]
16    return converted_value
17
18 # Example usage
19 if __name__ == "__main__":
20     distance = 10
21     from_unit = 'kilometers'
22     to_unit = 'miles'
23     result = convert_distance(distance, from_unit, to_unit)
24     print(f"{distance} {from_unit} is equal to {result:.2f} {to_unit}")
25
26     distance = 6.21
27     from_unit = 'miles'
28     to_unit = 'kilometers'
29     result = convert_distance(distance, from_unit, to_unit)
30     print(f"{distance} {from_unit} is equal to {result:.2f} {to_unit}")

```

Output :

```

PS C:\Users\keert\OneDrive\Documents\AI Assistance\lab> & "C:/Program Files/Python37/python.exe" C:\Users\keert\OneDrive\Documents\AI Assistance\lab\convert_distance.py
10 kilometers is equal to 6.21 miles
6.21 miles is equal to 9.99 kilometers
PS C:\Users\keert\OneDrive\Documents\AI Assistance\lab>

```

STAGE 3 :

Code :

```

5 # Write a Python function convert_distance(value, unit).
6 # If unit is "km", convert kilometers to miles.
7 # If unit is "miles", convert miles to kilometers.
8 # Use correct formulas (1 km = 0.621371 miles).
9 # Handle invalid units and negative values properly.
10 def convert_distance(value, unit):
11     if value < 0:
12         return "Error: Distance cannot be negative"
13     if unit == "km":
14         return value * 0.621371 # Convert km to miles
15     elif unit == "miles":
16         return value / 0.621371 # Convert miles to km
17     else:
18         return "Error: Invalid unit"
19 # Example usage
20 if __name__ == "__main__":
21     distance = 10
22     unit = 'km'
23     result = convert_distance(distance, unit)
24     print(f"{distance} {unit} is equal to {result:.2f} {'miles' if unit == 'km' else 'km'}")

```

Output :

```

PS C:\Users\keert\OneDrive\Documents\AI Assistance\lab> ^C
PS C:\Users\keert\OneDrive\Documents\AI Assistance\lab> & "C:/Pr
10 km is equal to 6.21 miles
PS C:\Users\keert\OneDrive\Documents\AI Assistance\lab> 

```

Final Observation :

In the first stage, the vague prompt resulted in a basic unit conversion function with unclear logic and limited accuracy. In the second stage, specifying the conversion directions improved correctness and readability of the code. In the third stage, detailed instructions and constraints led to an accurate and well-structured conversion function.

The final code correctly handled both kilometers-to-miles and miles-to-kilometers conversions. It also included proper validation for invalid units and negative values.