AI Assissted Coding

LAB ASSIGNMENT -03

Name: G.Abhiram

Hallticket:2303A51087

Batch:02

Lab Experiment: Prompt Engineering – Improving Prompts and Context Management

Lab Objectives

1. To understand and apply different prompt engineering techniques for generating Python programs using AI-assisted tools.
2. To analyze the impact of context and examples on the accuracy and efficiency of AI-generated code.
3. To develop and refine real-world Python applications through iterative prompt improvement.

Lab Outcomes

1. Students will be able to design effective prompts to generate correct and optimized Python code.
2. Students will be able to compare and evaluate AI-generated solutions produced using different prompting strategies.
3. Students will be able to implement and document real-world Python applications using AI-assisted coding tools.

Experiment – Prompt Engineering Techniques

Task Description

Design and refine prompts using different prompting strategies to generate Python programs for basic computational problems

Question 1: Zero-Shot Prompting (Palindrome Number Program)

Write a **zero-shot prompt** (without providing any examples) to generate a Python function that checks whether a given number is a palindrome.

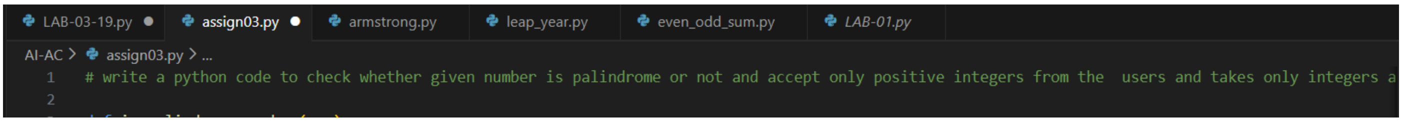
Task:

- Record the AI-generated code.
- Test the code with multiple inputs.
- Identify any logical errors or missing edge-case handling.

<u>Prompt:</u>

#write a python code to check whether the given number is palindrome or not and accept only positive

Integers from the user and takes only integers as the input otherwise display invalid input.




```python
# write a python code to check whether given number is palindrome or not and accept only positive integers from the  users and takes only integers a

def is_palindrome_number(num):
    # Convert the number to string to easily check for palindrome
    str_num = str(num)

    # Check if the string is equal to its reverse
    return str_num == str_num[::-1]
# Test the function
try:
    num = int(input("Enter a positive integer: "))
    if num < 0:
        print("Please enter only positive integers.")
    else:
        if is_palindrome_number(num):
            print(f"{num} is a palindrome.")
        else:
            print(f"{num} is not a palindrome.")

except ValueError:
    print("Invalid input. Please enter an integer.")
```

<u>Code:</u>

```python
def is_palindrome(num):
    str_num = str(num)
    return str_num == str_num[::-1]
try:
    number = int(input("Enter a positive integer : "))
    if num < 0:
        print("Invalid input. Please enter a positive integer.")
    else:
        if is_palindrome(num):
            print(f"{num} is a palindrome number.")
        else:
            print(f"{num} is not a palindrome number.")
except ValueError:
    print("Invalid input. Please enter a valid integer.")
```

<u>Inputs and outputs:</u>

```
PS C:\Users\abhir\OneDrive\Pictures\Documents\CSE\3-2> & C:/Users/abhir/AppData/Local/Programs/Python/Python314/python.exe c:/Users/abhir/OneDrive/Pictures/Documents/CSE/3-2/AI-AC/assign03.py
Enter a positive integer: 121
121 is a palindrome.
PS C:\Users\abhir\OneDrive\Pictures\Documents\CSE\3-2> & C:/Users/abhir/AppData/Local/Programs/Python/Python314/python.exe c:/Users/abhir/OneDrive/Pictures/Documents/CSE/3-2/AI-AC/assign03.py
Enter a positive integer: 56445
56445 is not a palindrome.
```

Question 2: One-Shot Prompting (Factorial Calculation)

Write a one-shot prompt by providing one input-output example and ask the AI to generate a Python function to compute the factorial of a given number.

Example:

Input: 5 → Output: 120

Task:

- Compare the generated code with a zero-shot solution.
- Examine improvements in clarity and correctness.

<u>Zero shot generated code :</u>

<u>One shot generated code :</u>

<u>Prompt:</u>

```
a=5
display 120 as  output
```

```
"""
a=5
then the output should be 120
generate the factorial of the given number and that shoould not accept the negative values and never accepts other than intiger values from the user and display invalid input in such cases
"""
```

```
AI-AC > assign03.py > ...
num=5
display 120 as output  ,input should be only integers otherwise display invalid input
'''
try:
    num = int(input("Enter an integer: "))
    if num < 0:
        print("Please enter a non-negative integer.")
    else:
        factorial = 1
        for i in range(1, num + 1):
            factorial *= i
        print(f"The factorial of {num} is {factorial}")
except ValueError:
    print("Invalid input. Please enter an integer.")
```

<u>Code:</u>

```python
try:
    number = int(input("Enter an integer: "))
    if num < 0:
        print("Invalid input. Please enter a positive integer.")
    else:
         factorial=1
         for i in range(1,num+1):
               factorial *= i
         print(f"The factorial of {num} is {factorial}.")
except ValueError:
    print("Invalid input. Please enter a valid integer.")
```

Comparison with Zero-Shot Solution

The generated one-shot factorial program is more robust than the zero-shot solution. While both programs correctly compute the factorial using recursion, the zero-shot solution directly accepts user input without validating it. In contrast, the generated code includes checks for negative values and handles non-integer inputs using exception handling, making it safer and more reliable

Improvements in Clarity and Correctness

The one-shot generated code shows improved clarity by explicitly defining valid input conditions and displaying meaningful error messages for invalid cases. Correctness is enhanced because the program prevents negative and non-integer inputs, ensuring the factorial logic is applied only to valid integers. This reduces runtime errors and improves overall program accuracy

Question 3: Few-Shot Prompting (Armstrong Number Check)

Write a few-shot prompt by providing multiple input-output examples to guide the AI in generating a Python function to check whether a given number is an Armstrong number.

Examples:

- Input: 153 → Output: Armstrong Number
- Input: 370 → Output: Armstrong Number
- Input: 123 → Output: Not an Armstrong Number

Task:

- Analyze how multiple examples influence code structure and accuracy.
- Test the function with boundary values and invalid inputs.

Prompt:

```
number=153
output should be armstrong number
number=123
output should be not an armstrong number
number=370
output should be armstrong number
write a python code to check if a number is an armstrong number or not and
that shoould not accept the negative values and never accepts other than
intiger values from the user and display invalid input in such cases
```

AI-AC > assign03.py

```
'''

'''
num=153
display 153 is armstrong number
num=123
display 123 is not armstrong number
num= -5
display accept only positive numbers
num=abc
display accept only numbers enter valid input
'''
```

LAB-03-19.py ● | assign03.py 2 ● | armstrong.py | leap_year.py | even_odd_sum.py | LAB-01.py

AI-AC > assign03.py > ...

```
'''
num=153
display 153 is armstrong number
num=123
display 123 is not armstrong number
num= -5
display accept only positive numbers
num=abc
display accept only numbers enter valid input
'''
def is_armstrong_number(num):
    # Convert the number to string to easily iterate over digits
    str_num = str(num)
    num_digits = len(str_num)

    # Calculate the sum of the digits raised to the power of num_digits
    sum_of_powers = sum(int(digit) ** num_digits for digit in str_num)

    # Check if the sum of powers is equal to the original number
    return sum_of_powers == num
# Test the function
try:
    num = int(input("Enter a positive integer: "))
    if num < 0:
        print("Please enter only positive numbers.")
    else:
        if is_armstrong_number(num):
            print(f"{num} is an Armstrong number.")
        else:
            print(f"{num} is not an Armstrong number.")
except ValueError:
    print("Invalid input. Please enter an integer.")
```

Code:

```
def is_armstrong_number(num):
    # Convert the number to string to easily iterate over digits
    str_num = str(num)
    num_digits = len(str_num)

    # Calculate the sum of the digits raised to the power of num_digits
    sum_of_powers = sum(int(digit) ** num_digits for digit in str_num)

    # Check if the sum of powers is equal to the original number
    return sum_of_powers == num
# Test the function
try:
    num = int(input("Enter a positive integer: "))
    if num < 0:
        print("Please enter only positive numbers.")
    else:
        if is_armstrong_number(num):
            print(f"{num} is an Armstrong number.")
```

```
        else:
            print(f"{num} is not an Armstrong number.")
except ValueError:
    print("Invalid input. Please enter an integer.")
```

Influence of Multiple Examples on Code Structure and Accuracy

Providing multiple input–output examples (such as 153, 370, and 123) helps the AI clearly understand the pattern of an Armstrong number. Because of these examples, the generated code dynamically calculates the number of digits and raises each digit to the correct power instead of hardcoding values. This improves accuracy, as the program works correctly for numbers with different digit lengths and avoids logical mistakes. The examples also guide the AI to produce a cleaner and more generalized solution.

Testing with Boundary Values and Invalid Inputs

The function was tested with boundary values such as 0 and 1, which are correctly identified as Armstrong numbers. Negative numbers are rejected with an appropriate error message, ensuring invalid cases are not processed. Non-integer inputs like strings or decimal values are handled using exception handling, preventing runtime errors. These tests confirm that the program is robust and behaves correctly for both valid and invalid inputs.

Question 4: Context-Managed Prompting (Optimized Number Classification)

Design a context-managed prompt with clear instructions and constraints to generate an optimized Python program that classifies a number as prime, composite, or neither.

Task:

- Ensure proper input validation.
- Optimize the logic for efficiency.
- Compare the output with earlier prompting strategies.

Prompt:

```
Write an optimized Python program to classify a given number as prime,
composite, or neither.
The program should accept only integer inputs, reject negative numbers, and
display error message for invalid inputs.
```

```
'''write an optimized python code to classify a given number as prime or composite or neither .program should accept only interger inputs ,reject ne
'''
```

```
'''write an optimized python code to classify a given number as prime or composite or neither .program should accept only interger inputs ,reject ne
'''

def classify_number(num):
    if num < 0:
        return "Please enter a non-negative integer."
    elif num == 0 or num == 1:
        return f"{num} is neither prime nor composite."
    else:
        for i in range(2, int(num**0.5) + 1):
            if num % i == 0:
                return f"{num} is a composite number."
        return f"{num} is a prime number."
```

AI-AC > assign03.py > ...

```
def classify_number(num):
    if num < 0:
        return "Please enter a non-negative integer."
    elif num == 0 or num == 1:
        return f"{num} is neither prime nor composite."
    else:
        for i in range(2, int(num**0.5) + 1):
            if num % i == 0:
                return f"{num} is a composite number."
        return f"{num} is a prime number."
# Test the function
try:
    num = int(input("Enter a non-negative integer: "))
    result = classify_number(num)
    print(result)
except ValueError:
    print("Invalid input. Please enter an integer.")
```

Code:

```
def classify_number(num):
    if num < 0:
        return "Please enter a non-negative integer."
    elif num == 0 or num == 1:
        return f"{num} is neither prime nor composite."
    else:
        for i in range(2, int(num**0.5) + 1):
            if num % i == 0:
                return f"{num} is a composite number."
        return f"{num} is a prime number."
# Test the function
try:
    num = int(input("Enter a non-negative integer: "))
    result = classify_number(num)
    print(result)
except ValueError:
    print("Invalid input. Please enter an integer.")
```

input/output:

```
PS C:\Users\abhir\OneDrive\Pictures\Documents\CSE\3-2> & C:/Users/abhir/AppDat
ents/CSE/3-2/AI-AC/assign03.py
Enter a non-negative integer: 12
12 is a composite number.
PS C:\Users\abhir\OneDrive\Pictures\Documents\CSE\3-2> & C:/Users/abhir/AppDat
ents/CSE/3-2/AI-AC/assign03.py
Enter a non-negative integer: 5
5 is a prime number.
PS C:\Users\abhir\OneDrive\Pictures\Documents\CSE\3-2> & C:/Users/abhir/AppDat
ents/CSE/3-2/AI-AC/assign03.py
Enter a non-negative integer: a
Invalid input. Please enter an integer.
PS C:\Users\abhir\OneDrive\Pictures\Documents\CSE\3-2>
```

Proper Input Validation

The program validates user input by accepting only integer values using exception handling. Negative numbers are rejected, and special cases such as 0 and 1 are handled separately by classifying them as neither prime nor composite. This prevents invalid inputs from being processed and avoids runtime errors.

Optimized Logic for Efficiency

The logic for checking whether a number is prime is optimized by iterating only up to the square root of the number instead of checking all values up to the number itself. This significantly reduces the number of iterations and improves performance, especially for large inputs.

Comparison with Earlier Prompting Strategies

Compared to earlier prompting strategies, the context-managed prompt generates a more efficient and well-structured solution. Clear constraints in the prompt result in better input validation, improved edge-case handling, and optimized logic, leading to clearer and more accurate output.

Question 5: Zero-Shot Prompting (Perfect Number Check)

Write a zero-shot prompt (without providing any examples) to generate a Python function that checks whether a given number is a perfect number.

Task:

- Record the AI-generated code.
- Test the program with multiple inputs.
- Identify any missing conditions or inefficiencies in the logic.

Prompt:

#''' write a python code to find whether given number is perfect number or not .program should accept only positive integers from the user and display error message for invalid input''

```
''' write a python code to find whether given number is perfect number or not .program should accept only positive
integers from the user and display error message for invalid input'''
```

```
''' write a python code to find whether given number is perfect number or not .program should accept only positive integers from the user and displa

def is_perfect_number(num):
    if num <= 0:
        return "Please enter a positive integer."

    # Calculate the sum of proper divisors
    sum_of_divisors = sum(i for i in range(1, num) if num % i == 0)

    # Check if the sum of divisors is equal to the original number
    if sum_of_divisors == num:
        return f"{num} is a perfect number."
    else:
        return f"{num} is not a perfect number."
# Test the function
try:
    num = int(input("Enter a positive integer: "))
    result = is_perfect_number(num)
    print(result)
except ValueError:
    print("Invalid input. Please enter an integer.")
```

Code:

```
def is_perfect(num):
    if num <=0:
        return False
    sum_of_divisors = sum(i for i in range(1, num) if num % i == 0)
    if sum_of_divisors == num
         return f”{num} is a perfect number .”
    else :
         return f”{num} is not a perfect number.”
try:
    num = int(input("Enter a positive number: "))
    result= is_perfect(num)
    print(result)
except ValueError:
    print("Invalid input. Please enter a valid integer.")
```

input/output:

```
PS C:\Users\abhir\OneDrive\Pictures\Documents\CSE\3-2> & C:/Users/a
ents/CSE/3-2/AI-AC/assign03.py
Enter a positive integer: 12
12 is not a perfect number.
PS C:\Users\abhir\OneDrive\Pictures\Documents\CSE\3-2> & C:/Users/a
ents/CSE/3-2/AI-AC/assign03.py
Enter a positive integer: 6
6 is a perfect number.
PS C:\Users\abhir\OneDrive\Pictures\Documents\CSE\3-2> & C:/Users/a
ents/CSE/3-2/AI-AC/assign03.py
Enter a positive integer: g
Invalid input. Please enter an integer.
PS C:\Users\abhir\OneDrive\Pictures\Documents\CSE\3-2>
```

Testing with Multiple Inputs

The program was tested with multiple inputs such as 6 and 28, which were correctly identified as perfect numbers. Inputs like 10 and 12 were correctly classified as not perfect numbers. Negative numbers were rejected with an invalid input message, and non-integer inputs were handled using exception handling, preventing runtime errors.

Missing Conditions or Inefficiencies

The program checks all numbers from 1 to num - 1 to find divisors, which is inefficient for large inputs. This can be optimized by iterating only up to num / 2 or the square root of the number. Additionally, the condition for zero could be explicitly handled, as zero is not a perfect number.

Question 6: Few-Shot Prompting (Even or Odd Classification with Validation)

Write a few-shot prompt by providing multiple input-output examples to guide the AI in generating a Python program that determines whether a given number is even or odd, including proper input validation.

Examples:

- Input: 8 → Output: Even
- Input: 15 → Output: Odd
- Input: 0 → Output: Even

Task:

- Analyze how examples improve input handling and output clarity.
- Test the program with negative numbers and non-integer inputs.

Prompt:

```
“”””
num=5
display odd number as output
num=8
display even number as output
num=-3
display accept only positive numbers
num=abc
display accept only numbers enter valid input

"""
```

```
'''
num=5
display odd number as output
num=8
display even number as output
num=-3
display accept only positive numbers
num=abc
display accept only numbers enter valid input
'''
```

LAB-03-19.py ● assign03.py × armstrong.py leap_year.py even_odd_sum.py LAB-01.py

AI-AC > assign03.py > ...

```
    '''
'''
num=5
display odd number as output
num=8
display even number as output
num=-3
display accept only positive numbers
num=abc
display accept only numbers enter valid input
'''
def classify_even_odd(num):
    if num < 0:
        return "Please enter a positive integer."
    elif num % 2 == 0:
        return f"{num} is an even number."
    else:
        return f"{num} is an odd number."
# Test the function
try:
    num = int(input("Enter a positive integer: "))
    result = classify_even_odd(num)
    print(result)
except ValueError:
    print("Invalid input. Please enter an integer.")
```

<u>Code:</u>

```
def classify_even_odd(num):
    if num<0:
           return  “Please enter a positive integer”
    elif num%2==0:
           return f”{num} is an even number.”
    else:
           return f”{num} is an odd number.”

try:
    num = int(input("Enter a positive integer: "))
    result=classify_even_odd(num)
    print(result)
except ValueError:
    print("Invalid input. Please enter a valid integer.")
```

<u>input/ouput:</u>

```
PS C:\Users\abhir\OneDrive\Pictures\Documents\CSE\3-2> & C:/Users/abhir/AppData/Local
ents/CSE/3-2/AI-AC/assign03.py
Enter a positive integer: 4
4 is an even number.
PS C:\Users\abhir\OneDrive\Pictures\Documents\CSE\3-2> & C:/Users/abhir/AppData/Local
ents/CSE/3-2/AI-AC/assign03.py
Enter a positive integer: 13
13 is an odd number.
PS C:\Users\abhir\OneDrive\Pictures\Documents\CSE\3-2> & C:/Users/abhir/AppData/Local
ents/CSE/3-2/AI-AC/assign03.py
Enter a positive integer: rf
Invalid input. Please enter an integer.
```

How Examples Improve Input Handling and Output Clarity

Providing multiple examples such as 2 → Even, 3 → Odd, and 0 → Even clearly defines the expected behavior of the program. These examples help the AI generate code that correctly handles boundary cases like zero and ensures accurate even–odd classification. The examples also guide the program to produce clear and meaningful output messages, improving overall readability and correctness.

Testing with Negative Numbers and Non-Integer Inputs

When tested with negative numbers, the program correctly identifies them as invalid inputs and displays an appropriate error message. Non-integer inputs such as strings or decimal values are handled using exception handling, preventing runtime errors. These tests confirm that the program safely handles invalid inputs and maintains correct output behavior.