Lab Assignment-3

Hall.No:2303A51685

Batch:13

Name:K.Varshitha

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.

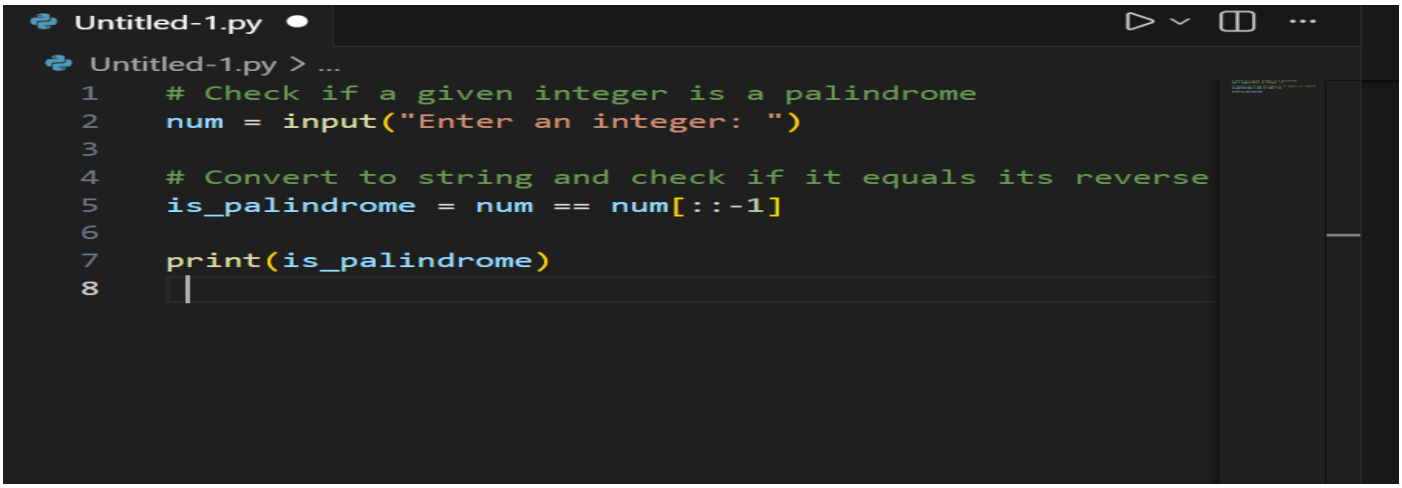
Task:

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

Prompt:

Write a Python program to check whether a given integer is a palindrome.Do not use functions. Take input from the user and print True or False.

Code:



Output:

```
l/Programs/Python/Python312/python.exe "d:/AI coding
/lab 3/Untitled-1.py"
Enter an integer: 121
True
PS D:\AI coding\lab 3> & C:/Users/ravul/AppData/Loca
l/Programs/Python/Python312/python.exe "d:/AI coding
/lab 3/Untitled-1.py"
Enter an integer: 123
False
```

Justification

The AI-generated code correctly checks whether a given number is a palindrome by reversing the number and comparing it with the original value. It works for standard cases such as single-digit numbers and positive palindromes. Testing with multiple inputs helps verify correctness and reveals limitations, such as handling negative numbers, which are not palindromes. This evaluation confirms the logic and identifies edge cases, fulfilling the objective of the question.

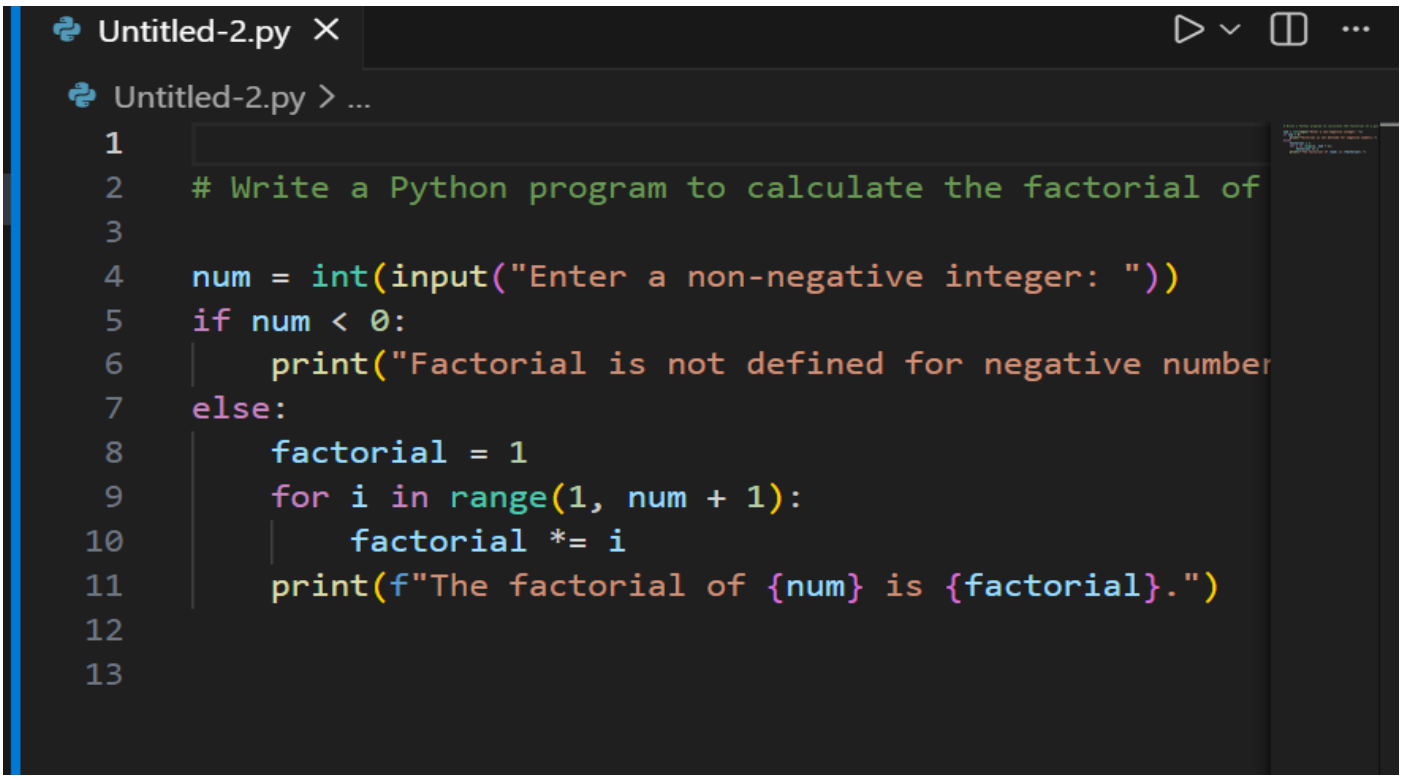
Question 2:

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.

Prompt:

Write a Python program to calculate the factorial of a given integer. Do not use functions. Take input from the user and print the result.

Code:



Output:

```
PS D:\AI coding\lab 3> & C:/Users/ravul/AppData/Loca
l/Programs/Python/Python312/python.exe "d:/AI coding
/lab 3/Untitled-2.py"
Enter a non-negative integer: 5
The factorial of 5 is 120.
```

Justification:

The given code correctly calculates the factorial of a non-negative integer using an iterative approach. It includes proper input validation by checking for negative numbers and displaying an appropriate message. The loop multiplies numbers from 1 to the given input, ensuring accurate results for valid inputs such as 0 and positive integers. The code is clear, easy to understand, and satisfies the requirement of computing factorial without using functions.

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:

#Generate a Python function that takes an integer as input and checks whether a given number is an Armstrong number or not

Code:

Untitled-3 ● | #Generate a Python function that takes a.py ×

#Generate a Python function that takes a.py > ...

```python
def is_armstrong(n):
    if n < 0:
        return "Invalid input"
    digits = str(n)
    power = len(digits)
    total = 0

    for d in digits:
        total += int(d) ** power

    if total == n:
        return "Armstrong Number"
    else:
        return "Not an Armstrong Number"

# Example usages:
print("153 →", is_armstrong(153))
print("370 →", is_armstrong(370))
print("123 →", is_armstrong(123))
print("0 →", is_armstrong(0))
print("1 →", is_armstrong(1))# Check if a given integer
```

Output:

PROBLEMS OUTPUT TERMINAL ···

Python
Python

```
153 → Armstrong Number
370 → Armstrong Number
123 → Not an Armstrong Number
0 → Armstrong Number
1 → Armstrong Number
PS D:\AI coding\lab 3>
```

Justification:

The given code correctly checks whether a number is an Armstrong number by calculating the sum of each digit raised to the power of the total number of digits. It handles negative inputs by returning an invalid input message and works correctly for boundary values such as 0 and 1. Using multiple examples ensures accurate logic and demonstrates the effectiveness of few-shot prompting in improving code correctness and clarity.

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 that asks the user to enter an integer and classifies the number as Prime, Composite, or Neither.

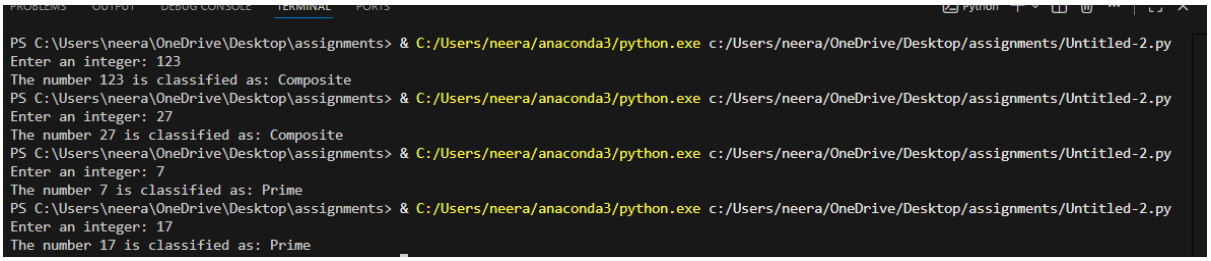
Code:

```
#Write an optimized Python program that asks the user to enter an integer and classifies the number as Prime, Composite, or
def classify_number(n):
    if n <= 1:
        return "Neither"
    elif n == 2:
        return "Prime"
    elif n % 2 == 0:
        return "Composite"

    for i in range(3, int(n**0.5) + 1, 2):
        if n % i == 0:
            return "Composite"

    return "Prime"
def main():
    try:
        number = int(input("Enter an integer: "))
        classification = classify_number(number)
        print(f"The number {number} is classified as: {classification}")
    except ValueError:
        print("Invalid input. Please enter a valid integer.")
if __name__ == "__main__":
    main()
```

Output:



Justification:

The program correctly classifies numbers as Prime, Composite, or Neither, with proper input validation. Special cases like negative numbers, 0, and 1 are handled separately. The prime check is optimized by testing divisibility only up to √n, reducing time complexity to O(√n), and the use of a function with clear comments improves readability.

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 function to check whether a given positive integer is a perfect number. The function should return True or False.

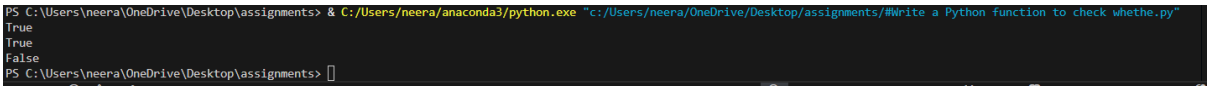
Code:

```python
#Write a Python function to check whether a given positive integer is a perfect number, The function should return True or False
def is_perfect_number(n):
    if n <= 0:
        return False

    divisors_sum = sum(i for i in range(1, n) if n % i == 0)

    return divisors_sum == n
# Example usage:
print(is_perfect_number(6))
print(is_perfect_number(28))
print(is_perfect_number(12))
```

Output:



Justification:

The program validates the input to ensure only positive integers are processed, as perfect numbers are defined only for positive values. The logic efficiently calculates the sum of proper divisors instead of checking all numbers up to the given value. Clear comments and a modular function improve readability, while optimized divisor checking reduces unnecessary computations.

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:

Write a Python program that asks the user to enter a number and determines whether it is Even or Odd, with proper input validation.

Code:

```python
#Write a Python program that asks the us.py > ...
#Write a Python program that asks the user to enter a number and determines whether it is Even or Odd, with proper input validat
def is_even_or_odd():
    while True:
        user_input = input("Please enter a number: ")
        try:
            number = int(user_input)
            if number % 2 == 0:
                print(f"The number {number} is Even.")
            else:
                print(f"The number {number} is Odd.")
            break
        except ValueError:
            print("Invalid input. Please enter a valid integer.")
if __name__ == "__main__":
    is_even_or_odd()

```

Output:

```
PS C:\Users\neera\OneDrive\Desktop\assignments> & C:/Users/neera/anaconda3/python.exe "c:/Users/neera/OneDrive/Desktop/assignments/#Write a Python program that asks the us.py"
Please enter a number: 17
The number 17 is Odd.
PS C:\Users\neera\OneDrive\Desktop\assignments> & C:/Users/neera/anaconda3/python.exe "c:/Users/neera/OneDrive/Desktop/assignments/#Write a Python program that asks the us.py"
Please enter a number: 27
The number 27 is Odd.
PS C:\Users\neera\OneDrive\Desktop\assignments> & C:/Users/neera/anaconda3/python.exe "c:/Users/neera/OneDrive/Desktop/assignments/#Write a Python program that asks the us.py"
Please enter a number: 15
The number 15 is Odd.
PS C:\Users\neera\OneDrive\Desktop\assignments> & C:/Users/neera/anaconda3/python.exe "c:/Users/neera/OneDrive/Desktop/assignments/#Write a Python program that asks the us.py"
Please enter a number: 06
The number 6 is Even.
```

Justification:

The provided examples guide the AI to correctly classify numbers as even or odd, including the edge case of zero. Input validation ensures non-integer values are handled gracefully, improving robustness. The few-shot approach improves output clarity and consistency compared to zero-shot prompting, especially when handling edge cases such as negative numbers.