AI ASSISTED CODING

LAB ASSIGNMENT – 3.1

NAME: CH. VISHWANATH RAO

BATCH:02

HTNO:-2303A51095

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 analyse the impact of context and examples on the accuracy and the efficiency of AI-generated code.

3)To develop and refine real-world Python applications through iterative prompt improvement.

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**.

```python
#write a python program to generate the python function that checks whether the given number is palindrome
def is_palindrome(num):
    return str(num) == str(num)[::-1]
number = int(input("Enter a number: "))
if is_palindrome(number):
    print(number, "is a palindrome number")
else:
    print(number, "is not a palindrome number")
```

Code

```
def is_palindrome(num):
    return str(num) == str(num)[::-1]
number = int(input("Enter a number: "))
if is_palindrome(number):
    print(number, "is a palindrome number")
else:
    print(number, "is not a palindrome number")
```

OUTPUT:

```
PS C:\Users\chvis\OneDrive\Desktop\AI ASSIST CODING> & C:/Users/chvis/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/chvis/OneDrive/Desktop/AI ASSIST CODING/ASSIGNM
ENT 3.1/1.py"
Enter a number: 121
121 is a palindrome number
PS C:\Users\chvis\OneDrive\Desktop\AI ASSIST CODING> & C:/Users/chvis/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/chvis/OneDrive/Desktop/AI ASSIST CODING/ASSIGNM
ENT 3.1/1.py"
Enter a number: 141
141 is a palindrome number
PS C:\Users\chvis\OneDrive\Desktop\AI ASSIST CODING> & C:/Users/chvis/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/chvis/OneDrive/Desktop/AI ASSIST CODING/ASSIGNM
ENT 3.1/1.py"
Enter a number: 111
111 is a palindrome number
PS C:\Users\chvis\OneDrive\Desktop\AI ASSIST CODING> & C:/Users/chvis/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/chvis/OneDrive/Desktop/AI ASSIST CODING/ASSIGNM
ENT 3.1/1.py"
Enter a number: 191
191 is a palindrome number
```

```
Enter a number: 11
11 is a palindrome number
PS C:\Users\chvis\OneDrive\Desktop\AI ASSIST CODING> & C:/Users/chvis/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/chvis/OneDrive/Desktop/AI ASSIST CODING/ASSIGNM
ENT 3.1/1.py"
Enter a number: 9
9 is a palindrome number
PS C:\Users\chvis\OneDrive\Desktop\AI ASSIST CODING> & C:/Users/chvis/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/chvis/OneDrive/Desktop/AI ASSIST CODING/ASSIGNM
ENT 3.1/1.py"
Enter a number: 7
7 is a palindrome number
PS C:\Users\chvis\OneDrive\Desktop\AI ASSIST CODING> & C:/Users/chvis/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/chvis/OneDrive/Desktop/AI ASSIST CODING/ASSIGNM
ENT 3.1/1.py"
Enter a number: 21
21 is not a palindrome number
PS C:\Users\chvis\OneDrive\Desktop\AI ASSIST CODING>
```

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.**

ZERO SHOT GENERATED CODE:

```python
#write a python program to generate the python function that checks whether the given number is palindrome
def is_palindrome(num):
    return str(num) == str(num)[::-1]
number = int(input("Enter a number: "))
if is_palindrome(number):
    print(number, "is a palindrome number")
else:
    print(number, "is not a palindrome number")
```

ONE SHOT GENERATED CODE:

```
'''a=n
the output should be 120
The factorial of the given num and that shouldn't accept the negative values & never accepts other than integer values from the user and display invalid input'''
```

```python
def factorial(value):
    if value == 0 or value == 1:
        return 1
    else:
        return value * factorial(value - 1)
```

```
try:
    user_input = int(input("Enter a positive integer to compute its factorial: "))
    if user_input < 0:
        print("Invalid input. Please enter a positive integer.")
    else:
        factorial_result = factorial(user_input)
        print(f"The factorial of {user_input} is {factorial_result}.")
except ValueError:
    print("Invalid input. Please enter a valid integer.")
```

```
'''a=n
 the output should be 120
The factorial of the given num and that shouldn't accept the negative values & never accepts other than integer values from the user and display invalid input'''


def factorial(value):
    if value == 0 or value == 1:
        return 1
    else:
        return value * factorial(value - 1)
try:
    user_input = int(input("Enter a positive integer to compute its factorial: "))
    if user_input < 0:
        print("Invalid input. Please enter a positive integer.")
    else:
        factorial_result = factorial(user_input)
        print(f"The factorial of {user_input} is {factorial_result}.")
except ValueError:
    print("Invalid input. Please enter a valid integer.")
```

CODE:

```
def factorial(value):
    if value == 0 or value == 1:
        return 1
    else:
        return value * factorial(value - 1)
try:
    user_input = int(input("Enter a positive integer to compute its factorial: "))
    if user_input < 0:
        print("Invalid input. Please enter a positive integer.")
    else:
        factorial_result = factorial(user_input)
        print(f"The factorial of {user_input} is {factorial_result}.")
```

Comparison with Zero-Shot Solution

The one-shot factorial program is more robust than the zero-shot version because it includes input validation. While the zero-shot solution directly processes user input, the one-shot code checks for invalid and negative values, making it safer and more reliable.

Clarity and Accuracy

Clarity is improved by clearly defining valid input conditions and displaying meaningful error messages. Correctness is enhanced by preventing negative and non-integer inputs, ensuring factorial computation is performed only for valid integers.

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.**

```
'''number = 371
Output: Armstrong number

number = 407
Output: Armstrong number

number = 153
Output: Armstrong number'''

'''write a python program to generate the python function that checks whether the given number is Armstrong number or not'''
```

```
number = 153
Output: Armstrong number'''

'''write a python program to generate the python function that checks whether the given number is Armstrong number or not'''

def is_armstrong(value):
    length = len(str(value))
    total = sum(int(digit) ** length for digit in str(value))
    return total == value
```

```
def is_armstrong(value):
    length = len(str(value))
    total = sum(int(digit) ** length for digit in str(value))
    return total == value
try:
    num = int(input("Enter a positive integer to check if it's an Armstrong number: "))
    if num < 0:
        print("Invalid input. Please enter a positive integer.")
    else:
        if is_armstrong(num):
            print(f"{num} is an Armstrong number.")
        else:
            print(f"{num} is not an Armstrong number.")
except ValueError:
    print("Invalid input. Please enter a valid integer.")
```

CODE:

```
def is_armstrong(value):
    length = len(str(value))
    total = sum(int(digit) ** length for digit in str(value))
    return total == value
try:
    num = int(input("Enter a positive integer to check if it's an Armstrong number: "))
    if num < 0:
        print("Invalid input. Please enter a positive integer.")
    else:
        if is_armstrong(num):
            print(f"{num} is an Armstrong number.")
        else:
            print(f"{num} is not an Armstrong number.")
except ValueError:
    print("Invalid input. Please enter a valid integer.")
```

Effect of Multiple Examples

Using several input–output examples helps the AI identify the Armstrong number pattern correctly. This leads to a generalised solution that calculates digit count dynamically and works for numbers of any length. As a result, accuracy is improved and hardcoded logic is avoided.

Boundary and Invalid Input Testing

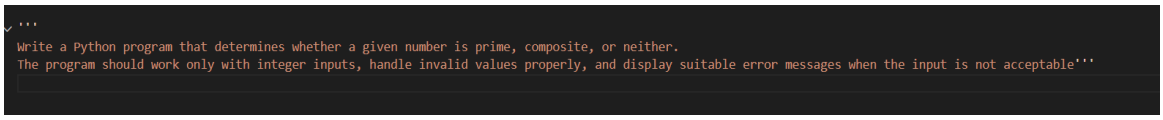
Testing with values like 0 and 1 confirms correct handling of edge cases. Invalid inputs such as negative numbers and non-integers are safely rejected using error handling, making the program more reliable and error-free.

Question 4: Context-Managed Prompting (Optimised 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.
- Optimise the logic for efficiency.
- Compare the output with earlier prompting strategies.



```python
def check_number_type(value):
    if value < 2:
        return "neither prime nor composite"
    for divisor in range(2, int(value**0.5) + 1):
        if value % divisor == 0:
            return "composite"
    return "prime"

try:
    input_value = int(input("Enter a positive integer to classify: "))
    if input_value < 0:
        print("Invalid input. Please enter a positive integer.")
    else:
        result = check_number_type(input_value)
        print(f"{input_value} is {result}.")
except ValueError:
    print("Invalid input. Please enter a valid integer.")
```

```python
Write a Python program that determines whether a given number is prime, composite, or neither.
The program should work only with integer inputs, handle invalid values properly, and display suitable error messages when the input is not acceptable'''



def check_number_type(value):
    if value < 2:
        return "neither prime nor composite"
    for divisor in range(2, int(value**0.5) + 1):
        if value % divisor == 0:
            return "composite"
    return "prime"
```

```python
Write a Python program that determines whether a given number is prime, composite, or neither.
The program should work only with integer inputs, handle invalid values properly, and display suitable err



def check_number_type(value):
    if value < 2:
        return "neither prime nor composite"
    for divisor in range(2, int(value**0.5) + 1):
        if value % divisor == 0:
            return "composite"
    return "prime"

try:
    input_value = int(input("Enter a positive integer to classify: "))
    if input_value < 0:
        print("Invalid input. Please enter a positive integer.")
    else:
        result = check_number_type(input_value)
        print(f"{input_value} is {result}.")
except ValueError:
    print("Invalid input. Please enter a valid integer.")
```

CODE:

```
def check_number_type(value):
    if value < 2:
        return "neither prime nor composite"
    for divisor in range(2, int(value**0.5) + 1):
        if value % divisor == 0:
            return "composite"
    return "prime"


try:
    input_value = int(input("Enter a positive integer to classify: "))
    if input_value < 0:
        print("Invalid input. Please enter a positive integer.")
    else:
        result = check_number_type(input_value)
        print(f"{input_value} is {result}.")
except ValueError:
    print("Invalid input. Please enter a valid integer.")
```

OUTPUT:

Input Handling and Validation

The program ensures only valid integer inputs are processed by using exception handling. Negative values are rejected, and edge cases like 0 and 1 are correctly identified as neither prime nor composite, preventing incorrect execution.

Efficient Prime-Checking Logic

To improve efficiency, the program checks divisibility only up to the square root of the given number. This approach minimises unnecessary iterations, making the program faster for larger numbers.

Improvement Over Basic Prompting

Compared to simpler prompting methods, the refined prompt produces a cleaner and more reliable solution. Clear requirements lead to better error handling, proper edge-case coverage, and optimised logic, resulting in accurate and readable 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.**

```
#Write a Python program using a function to determine whether a given number is a perfect number or not.
# The program should accept only positive integer inputs from the user and display an error message for any invalid input
```

```python
#Write a Python program using a function to determine whether a given number is a perfect number or not.
# The program should accept only positive integer inputs from the user and display an error message for any invalid input

def check_perfect(num):
    if num <= 0:
        return False

    total = sum(div for div in range(1, num) if num % div == 0)
    return total == num
```

```python
#Write a Python program using a function to determine whether a given number is a perfect number or not.
# The program should accept only positive integer inputs from the user and display an error message for any invalid input

def check_perfect(num):
    if num <= 0:
        return False

    total = sum(div for div in range(1, num) if num % div == 0)
    return total == num


try:
    number = int(input("Enter a positive integer to check if it's a perfect number: "))

    if number <= 0:
        print("Invalid input. Please enter a positive integer.")
    else:
        if check_perfect(number):
            print(f"{number} is a perfect number.")
        else:
            print(f"{number} is not a perfect number.")

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

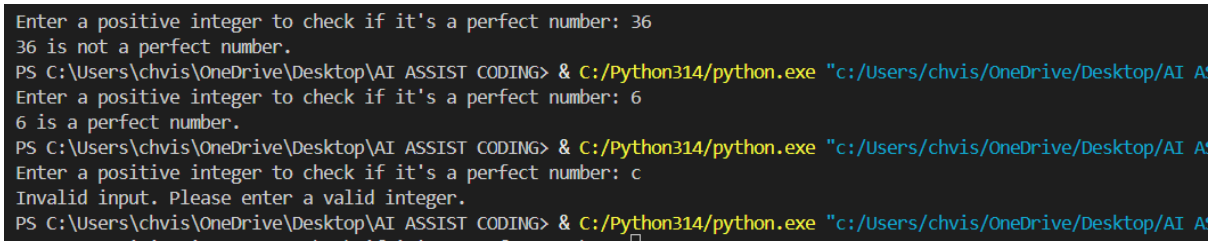
CODE:

```
def check_perfect(num):
    if num <= 0:
        return False
    total = sum(div for div in range(1, num) if num % div == 0)
    return total == num
try:
    number = int(input("Enter a positive integer to check if it's a perfect number: "))

    if number <= 0:
        print("Invalid input. Please enter a positive integer.")
    else:
        if check_perfect(number):
            print(f"{number} is a perfect number.")
        else:
            print(f"{number} is not a perfect number.")

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

output:



Validation Through Sample Inputs

The program was verified using several test values. Numbers like 6 and 28 were correctly detected as perfect numbers, while values such as 10 and 12 were identified as non-perfect. Invalid cases, including negative and non-integer inputs, were safely handled using error messages and exception handling.

Limitations and Possible Optimisations

The current approach checks all divisors from 1 up to the number itself, which is inefficient for large values. Performance can be improved by limiting the loop to half of the number or up to its square root. Also, explicitly handling the case of zero would make the logic more complete, since 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.**

```
'''
number = 8
Output: Even number

number = 15
Output: Odd number

number = 4
Output: Even number'''
```

```
'''write a Python program to determine whether a given number is even or odd.
The program should accept only positive integer inputs, reject negative numbers and non-integer values, and display an  error for invalid input'


'''
try:
    user_input = int(input("Enter a positive integer: "))
    if user_input < 0:
        print("Invalid input. Please enter a positive integer.")
    else:
        if user_input % 2 == 0:
            print(f"{user_input} is an even number.")
        else:
            print(f"{user_input} is an odd number.")
except ValueError:
    print("Invalid input. Please enter a valid integer.")'''
```

```
try:
    user_input = int(input("Enter a positive integer: "))
    if user_input < 0:
        print("Invalid input. Please enter a positive integer.")
    else:
        if user_input % 2 == 0:
            print(f"{user_input} is an even number.")
        else:
            print(f"{user_input} is an odd number.")
except ValueError:
    print("Invalid input. Please enter a valid integer.")
```

OUTPUT:

```
PS C:\Users\chvis\OneDrive\Desktop\AI ASSIST CODING> & C:/Python314/python.exe "c:/Users/chvis/OneDrive/Desktop/AI ASSIST CODING/ASSIGNMENT 3.1/7.py"
Enter a positive integer: 8
8 is an even number.
PS C:\Users\chvis\OneDrive\Desktop\AI ASSIST CODING> & C:/Python314/python.exe "c:/Users/chvis/OneDrive/Desktop/AI ASSIST CODING/ASSIGNMENT 3.1/7.py"
Enter a positive integer: 15
15 is an odd number.
PS C:\Users\chvis\OneDrive\Desktop\AI ASSIST CODING> & C:/Python314/python.exe "c:/Users/chvis/OneDrive/Desktop/AI ASSIST CODING/ASSIGNMENT 3.1/7.py"
Enter a positive integer: 4
4 is an even number.
PS C:\Users\chvis\OneDrive\Desktop\AI ASSIST CODING> & C:/Python314/python.exe "c:/Users/chvis/OneDrive/Desktop/AI ASSIST CODING/ASSIGNMENT 3.1/7.py"
Enter a positive integer: D
Invalid input. Please enter a valid integer.
```

Code:

```
try:
    user_input = int(input("Enter a positive integer: "))
    if user_input < 0:
        print("Invalid input. Please enter a positive integer.")
    else:
        if user_input % 2 == 0:
            print(f"{user_input} is an even number.")
        else:
            print(f"{user_input} is an odd number.")
except ValueError:
    print("Invalid input. Please enter a valid integer.")
```

Improving Program Behaviour

Including clear input–output examples helps define how the program should behave in different situations. They guide the logic to correctly classify numbers as even or odd, handle edge cases like zero, and generate clear, user-friendly output messages.

Handling Invalid and Edge Inputs

The program correctly rejects negative values by displaying an error message. It also uses exception handling to manage non-integer inputs such as text or decimal values, ensuring stable execution without runtime errors.