Assginment-3.1

AI Assisted Coding

Name : Radha Krishna Jagarlapudi

Batch No : 39

HT No : 2303A52055

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

Write a zero-shot prompt (without providing any examples) to generate

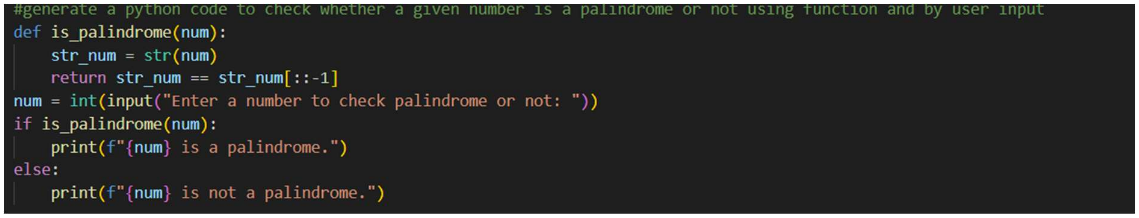
a Python function that checks whether a given number is a palindrome.

Task:

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

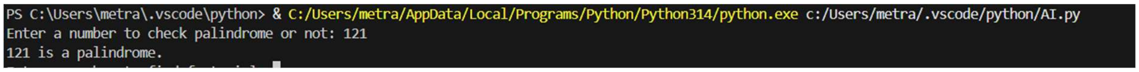
Prompt : #“Write a Python program that takes a number as user input and checks whether it is a palindrome number.”

Code:



```python
#generate a python code to check whether a given number is a palindrome or not using function and by user input
def is_palindrome(num):
    str_num = str(num)
    return str_num == str_num[::-1]
num = int(input("Enter a number to check palindrome or not: "))
if is_palindrome(num):
    print(f"{num} is a palindrome.")
else:
    print(f"{num} is not a palindrome.")
```

Output:



```
PS C:\Users\metra\.vscode\python> & C:/Users/metra/AppData/Local/Programs/Python/Python314/python.exe c:/Users/metra/.vscode/python/AI.py
Enter a number to check palindrome or not: 121
121 is a palindrome.
```

Analysis:

1. **The zero-shot prompt generated correct basic logic for palindrome checking.**
2. **The initial solution lacked proper handling for negative and invalid inputs.**
3. **Additional validation was required to make the program more reliable.**

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

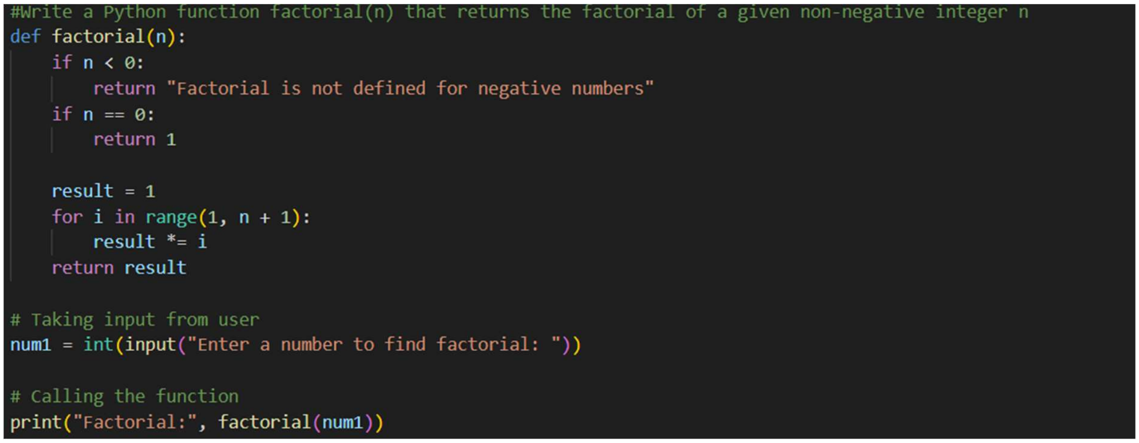
Task:

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

Prompt :

#“Write a Python program to compute the factorial of a number taken as user input.

Code :



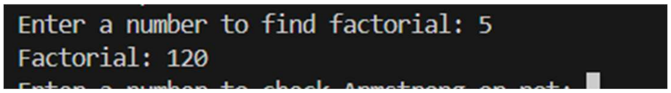
```python
#Write a Python function factorial(n) that returns the factorial of a given non-negative integer n
def factorial(n):
    if n < 0:
        return "Factorial is not defined for negative numbers"
    if n == 0:
        return 1

    result = 1
    for i in range(1, n + 1):
        result *= i
    return result

# Taking input from user
num1 = int(input("Enter a number to find factorial: "))

# Calling the function
print("Factorial:", factorial(num1))
```

OUPUT :



```
Enter a number to find factorial: 5
Factorial: 120
```

Analysis :

1. Providing one example improved the clarity of the generated code.

2. The program correctly handled negative numbers compared to zero-shot prompting.

3. The logic was more structured and easier to understand.

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

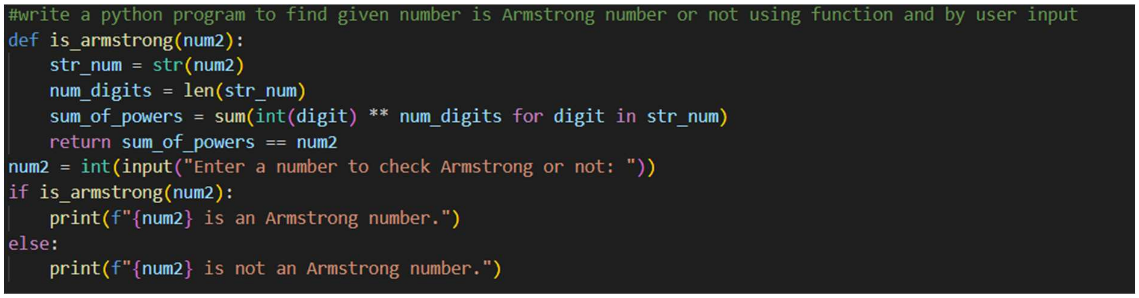
accuracy.

- Test the function with boundary values and invalid inputs.

Prompt :

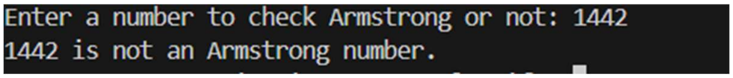
#“Write a Python program to check whether a number is an Armstrong number using user input.

Code :



```python
#write a python program to find given number is Armstrong number or not using function and by user input
def is_armstrong(num2):
    str_num = str(num2)
    num_digits = len(str_num)
    sum_of_powers = sum(int(digit) ** num_digits for digit in str_num)
    return sum_of_powers == num2
num2 = int(input("Enter a number to check Armstrong or not: "))
if is_armstrong(num2):
    print(f"{num2} is an Armstrong number.")
else:
    print(f"{num2} is not an Armstrong number.")
```

OUTPUT :



```
Enter a number to check Armstrong or not: 1442
1442 is not an Armstrong number.
```

Analysis :

1. **Multiple examples helped the AI identify the Armstrong number pattern accurately.**
2. **The generated code was well-structured and produced correct results.**
3. **Few-shot prompting reduced ambiguity compared to zero-shot prompting.**

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.

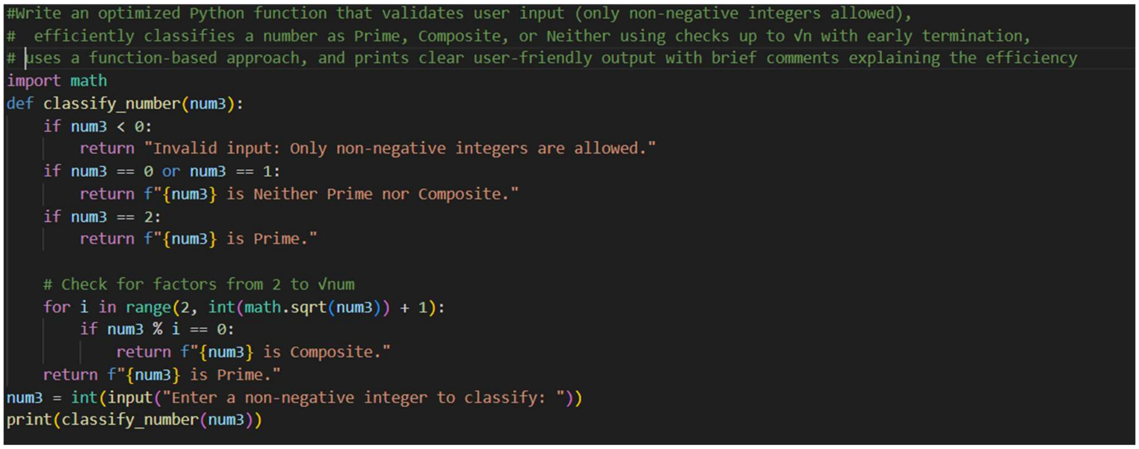
Task:

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

Prompt :

#“Write an optimized Python program that takes a number as user input and classifies it as Prime, Composite, or Neither.

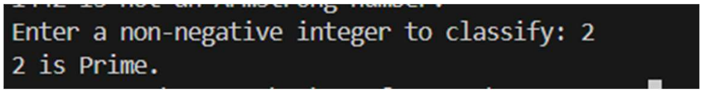
Code :



```
#Write an optimized Python function that validates user input (only non-negative integers allowed),
#  efficiently classifies a number as Prime, Composite, or Neither using checks up to √n with early termination,
# uses a function-based approach, and prints clear user-friendly output with brief comments explaining the efficiency
import math
def classify_number(num3):
    if num3 < 0:
        return "Invalid input: Only non-negative integers are allowed."
    if num3 == 0 or num3 == 1:
        return f"{num3} is Neither Prime nor Composite."
    if num3 == 2:
        return f"{num3} is Prime."

    # Check for factors from 2 to √num
    for i in range(2, int(math.sqrt(num3)) + 1):
        if num3 % i == 0:
            return f"{num3} is Composite."
    return f"{num3} is Prime."
num3 = int(input("Enter a non-negative integer to classify: "))
print(classify_number(num3))
```

OUTPUT :



```
Enter a non-negative integer to classify: 2
2 is Prime.
```

Analysis :

1. **Clear instructions and constraints resulted in an optimized solution.**
2. **The program efficiently classified numbers using minimal iterations.**
3. **Proper input validation improved program reliability.**

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.

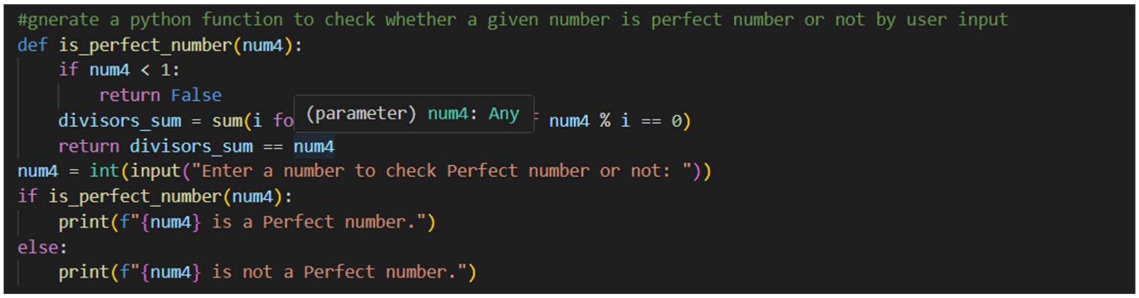
- Test the program with multiple inputs.

- Identify any missing conditions or inefficiencies in the logic.

Prompt :

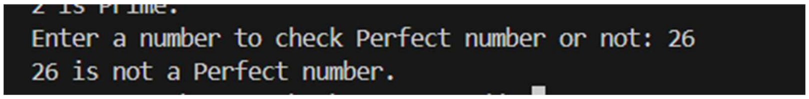
#“Write a Python program that takes a number as user input and checks whether it is a perfect number.”

Code :



```
#gnerate a python function to check whether a given number is perfect number or not by user input
def is_perfect_number(num4):
    if num4 < 1:
        return False
    divisors_sum = sum(i fo (parameter) num4: Any = num4 % i == 0)
    return divisors_sum == num4
num4 = int(input("Enter a number to check Perfect number or not: "))
if is_perfect_number(num4):
    print(f"{num4} is a Perfect number.")
else:
    print(f"{num4} is not a Perfect number.")
```

OUTPUT :



```
2 is Prime.
Enter a number to check Perfect number or not: 26
26 is not a Perfect number.
```

Analysis :

1. The generated code correctly identified perfect numbers.

2. The solution was inefficient for large inputs due to unnecessary iterations.

3. Edge-case handling and optimization were missing initially.

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

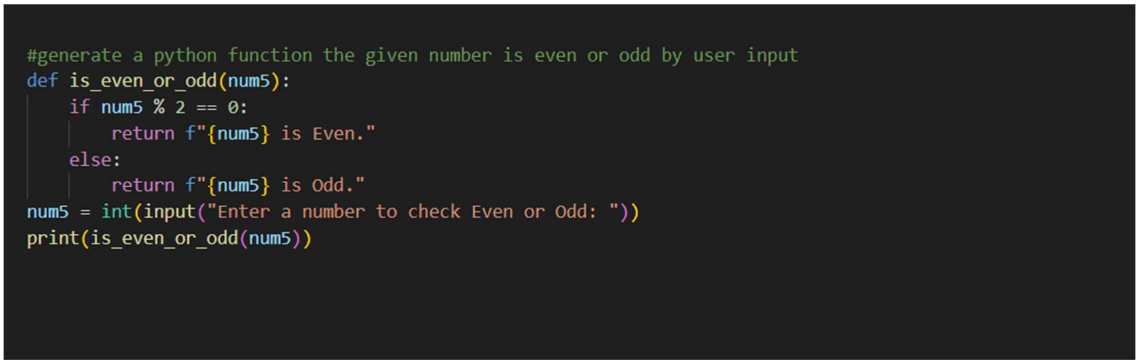
Task:

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

Prompt :

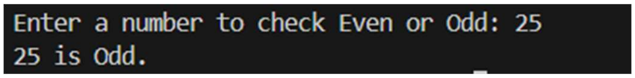
#“Write a Python program that determines whether a number is even or odd using user input with proper validation.

Code :



```
#generate a python function the given number is even or odd by user input
def is_even_or_odd(num5):
    if num5 % 2 == 0:
        return f"{num5} is Even."
    else:
        return f"{num5} is Odd."
num5 = int(input("Enter a number to check Even or Odd: "))
print(is_even_or_odd(num5))
```

OUTPUT :



```
Enter a number to check Even or Odd: 25
25 is Odd.
```

Analysis :

1. Examples improved input handling and output clarity.

2. The program correctly handled negative numbers and zero.

3. Input validation prevented errors for non-integer values.