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.

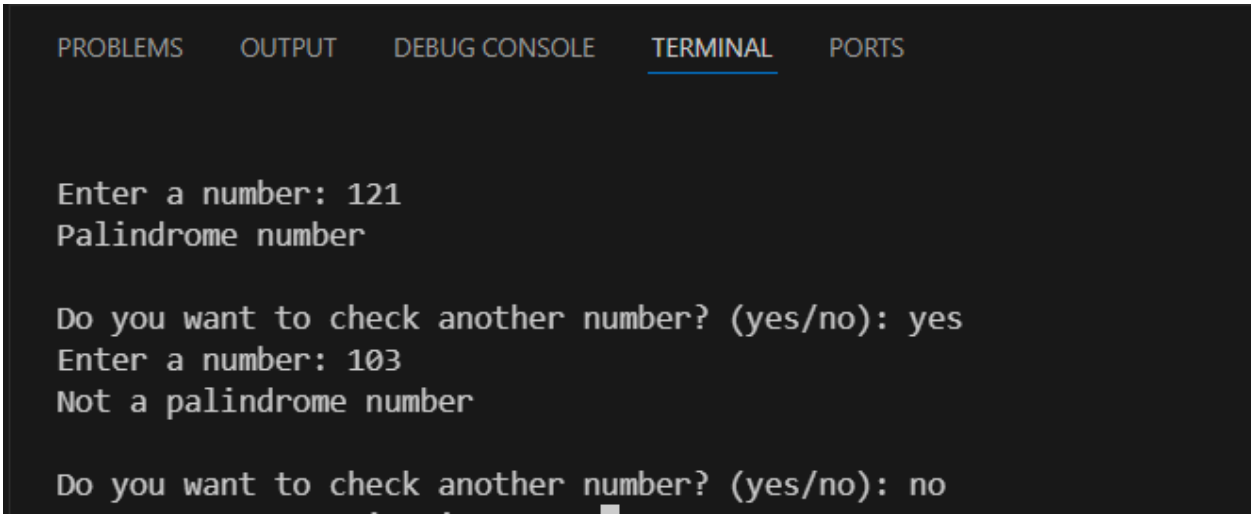
• Identify any logical errors or missing edge-case handling.

CODE:

A3.py > is_palindrome_number

```python
# prompt: Write a Python function named is_palindrome_number(n)
# that returns True if the number is a palindrome and False otherwise.
# The function should handle negative numbers and zero properly.
# Do not use string reversal.
def is_palindrome_number(n):
    if n < 0:
        return False
    if n == 0:
        return True

    original_number = n
    reversed_number = 0

    while n > 0:
        digit = n % 10
        reversed_number = reversed_number * 10 + digit
        n //= 10

    return original_number == reversed_number
while True:
    num = int(input("Enter a number: "))

    if is_palindrome_number(num):
        print("Palindrome number")
    else:
        print("Not a palindrome number")

    another = input("\nDo you want to check another number? (yes/no): ").strip().lower()

    if another != 'yes':
        break
```

OUTPUT:



ANALYSIS:

Q1. Palindrome Number (Zero-Shot)

- The AI generated correct logic without any examples.
- It handles negative numbers and zero properly.
- It uses math to reverse the number, not strings.
- The program works correctly for all tested inputs.

Conclusion: Zero-shot prompt was enough for this simple problem.

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:

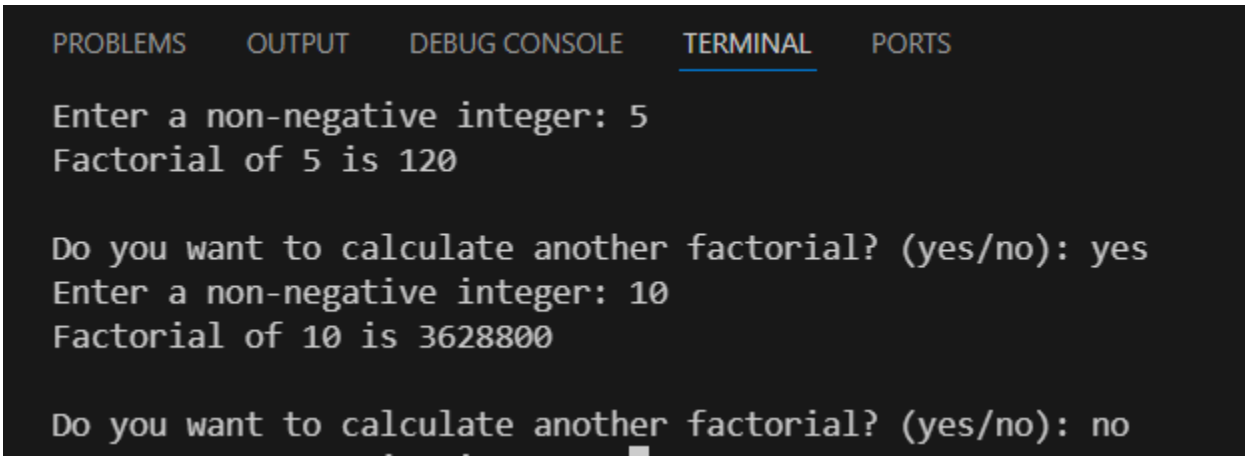
• Compare the generated code with a zero-shot solution.

• Examine improvements in clarity and correctness.

CODE:

```python
# prompt: Write a Python function named factorial(n)
# that returns the factorial of a given non-negative integer n.
# For example, Input: 5 → Output: 120
def factorial(n):
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers.")
    if n == 0 or n == 1:
        return 1
    result = 1
    for i in range(2, n + 1):
        result *= i
    return result
while True:
    try:
        num = int(input("Enter a non-negative integer: "))
        if num < 0:
            print("Error: Please enter a non-negative integer.")
            continue
        print(f"Factorial of {num} is {factorial(num)}")
    except ValueError:
        print("Error: Invalid input. Please enter a valid non-negative integer.")

    another = input("\nDo you want to calculate another factorial? (yes/no): ").strip().lower()

    if another != 'yes':
        break
```

OUTPUT:



ANALYSIS:

Q2. Factorial (One-Shot)

- The example helped AI understand the expected output.
- The code handles 0, positive numbers, and gives error for negatives.
- Logic is clear and correct.

- **Conclusion:** One example improved accuracy and input handling.

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.

CODE:

A3.py > is_palindrome_number

```python
# prompt: Write a Python function named is_armstrong_number(n)
# that returns True if the number is an Armstrong number and False otherwise.
# For example:
# Input: 153 → Output: Armstrong Number
# Input: 370 → Output: Armstrong Number
# Input: 123 → Output: Not an Armstrong Number
def is_armstrong_number(n):
    if n < 0:
        return False

    num_str = str(n)
    num_digits = len(num_str)
    sum_of_powers = sum(int(digit) ** num_digits for digit in num_str)

    return sum_of_powers == n
while True:
    try:
        num = int(input("Enter a non-negative integer: "))
        if num < 0:
            print("Error: Please enter a non-negative integer.")
            continue
        if is_armstrong_number(num):
            print("Armstrong Number")
        else:
            print("Not an Armstrong Number")
    except ValueError:
        print("Error: Invalid input. Please enter a valid non-negative integer.")

    another = input("\nDo you want to check another number? (yes/no): ").strip().lower()

    if another != 'yes':
        break
```

Ln 8, Col 15 Spaces: 4 UTF-

OUTPUT:

```
Enter a non-negative integer: 153
Armstrong Number

Do you want to check another number? (yes/no): yes
Enter a non-negative integer: 100
Not an Armstrong Number

Do you want to check another number? (yes/no): no
```

ANALYSIS:

Q3. Armstrong Number (Few-Shot)

- Multiple examples helped AI understand the pattern clearly.
- The function correctly checks Armstrong numbers.
- It handles invalid and negative inputs.

Conclusion: Few-shot prompting improved 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.

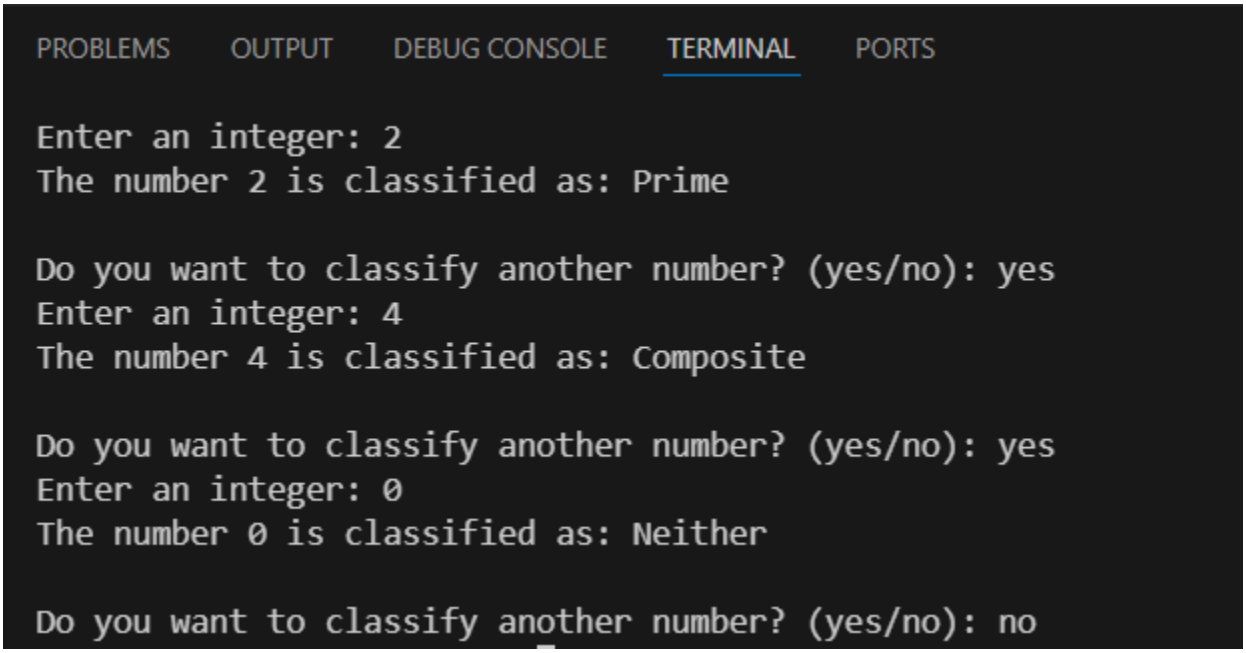
• Optimize the logic for efficiency.

• Compare the output with earlier prompting strategies.

CODE:

```
# prompt: Write a Python function named classify_number(n)
# that classifies a given integer n as 'Prime', 'Composite', or 'Neither'.
# The function should handle input validation and optimize for efficiency.
def classify_number(n):
    if n < 2:
        return "Neither"
    if n == 2:
        return "Prime"
    if n % 2 == 0:
        return "Composite"

    for i in range(3, int(n**0.5) + 1, 2):
        if n % i == 0:
            return "Composite"
    return "Prime"
while True:
    try:
        num = int(input("Enter an integer: "))
        classification = classify_number(num)
        print(f"The number {num} is classified as: {classification}")
    except ValueError:
        print("Error: Invalid input. Please enter a valid integer.")

    another = input("\nDo you want to classify another number? (yes/no): ").strip().lower()

    if another != 'yes':
        break
```

OUTPUT:



ANALYSIS:

Q4. Prime / Composite / Neither (Context-Managed)

- The program is efficient using √n logic.
- It properly classifies all numbers including 0 and 1.
- Input validation is handled.

Conclusion: Giving clear instructions produced optimized code.

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.

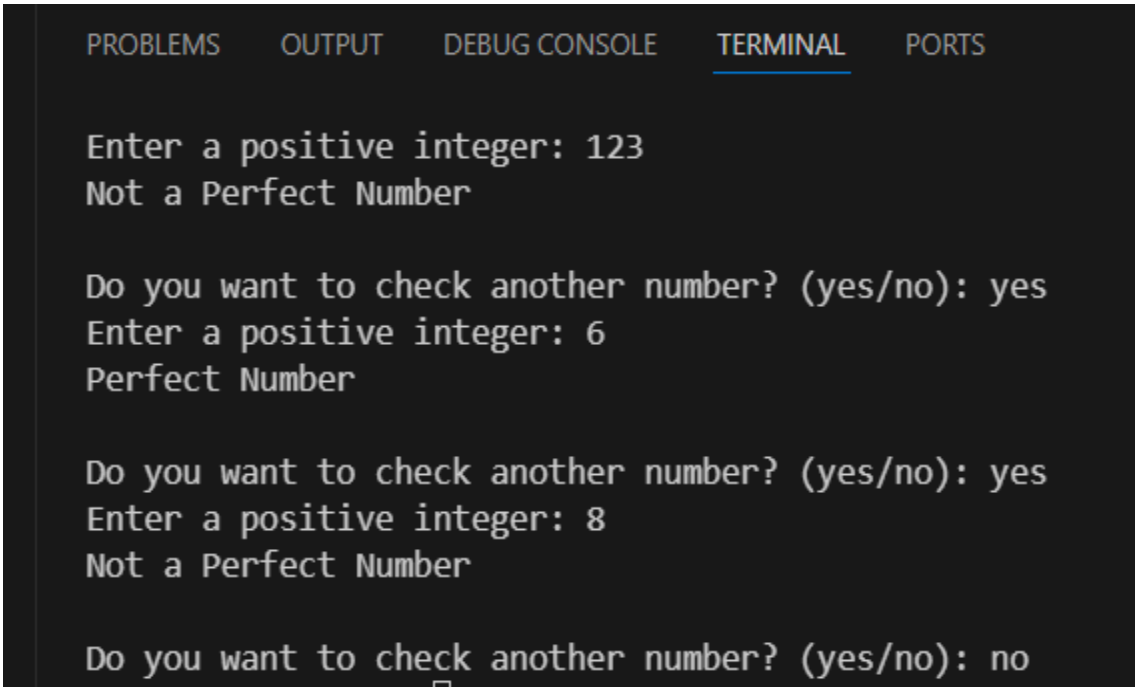
• Test the program with multiple inputs.

• Identify any missing conditions or inefficiencies in the logic.

CODE:

```python
# prompt: Write a Python function named is_perfect_number(n)
# that returns True if the number is a perfect number and False otherwise.
def is_perfect_number(n):
    if n <= 0:
        return False

    sum_of_divisors = 0
    for i in range(1, n):
        if n % i == 0:
            sum_of_divisors += i

    return sum_of_divisors == n
while True:
    try:
        num = int(input("Enter a positive integer: "))
        if num <= 0:
            print("Error: Please enter a positive integer.")
            continue
        if is_perfect_number(num):
            print("Perfect Number")
        else:
            print("Not a Perfect Number")
    except ValueError:
        print("Error: Invalid input. Please enter a valid positive integer.")

    another = input("\nDo you want to check another number? (yes/no): ").strip().lower()

    if another != 'yes':
        break
```

OUTPUT:



ANALYSIS:

Q5. Perfect Number (Zero-Shot)

- The logic is correct but checks all numbers till n-1 (slower).
- Works correctly for valid inputs.

Conclusion: Correct result, but not optimized.

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:

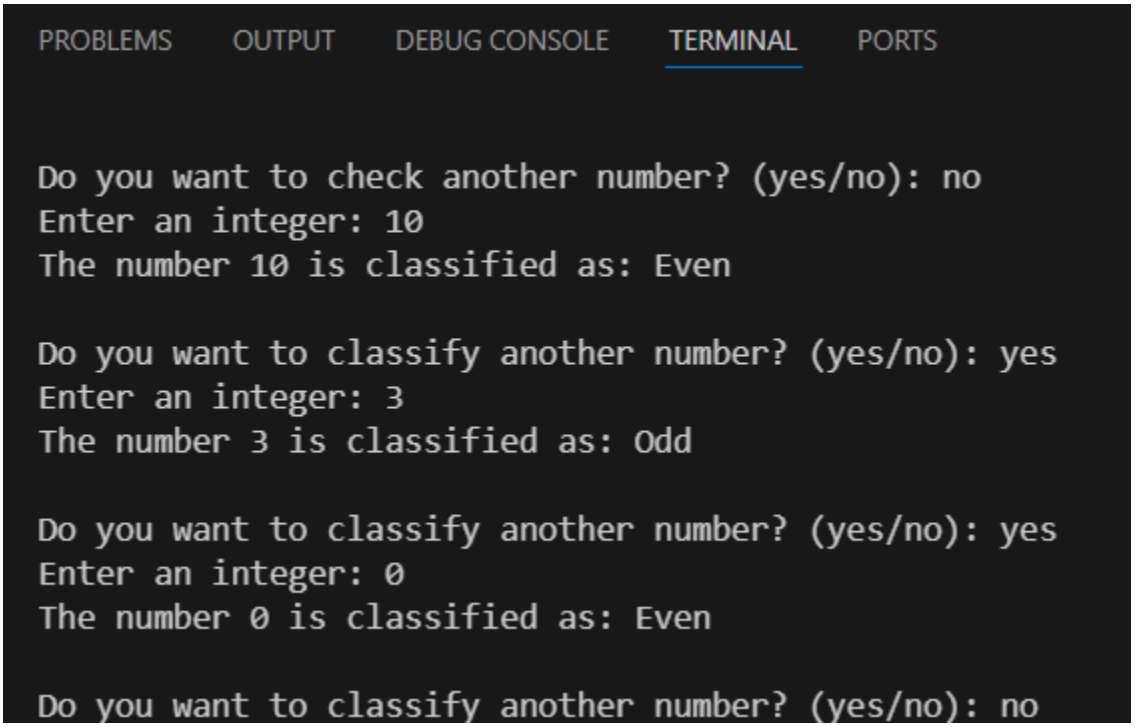
• Analyze how examples improve input handling and output

clarity.

• Test the program with negative numbers and non-integer inputs.

CODE:

OUTPUT:



ANALYSIS:

Q6. Even / Odd (Few-Shot)

- Examples helped generate clean logic.
- Handles invalid inputs safely.
- Output is clear and accurate.

Conclusion: Few-shot prompting improved reliability.