AI ASSISTED CODING ASSIGNMENT-3.1

A.Nikhil (2303A51934) BATCH-27

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.

PROMPT:

Write a Python function that checks whether a given integer is a palindrome.

The function should return True if the number is a palindrome and False otherwise.

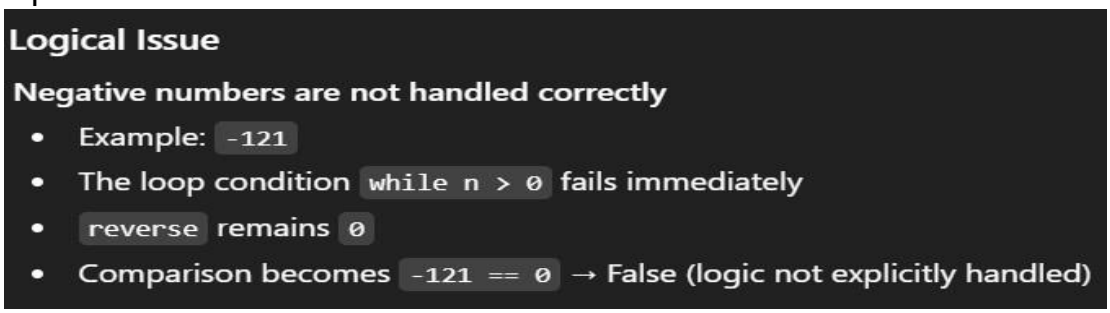
Do not convert the number to a string.

```
def is_palindrome(n):
    original = n
    reverse = 0

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

    return original == reverse


# Take input from the user
n = int(input("Enter a number: "))

# Call the function and print the result
if is_palindrome(n):
    print("The number is a palindrome")
else:
    print("The number is not a palindrome")
```

OUTPUT:

```
Enter a number: 121
The number is a palindrome
```

Inputs:



Input	Expected Output	Actual Output
121	True	True
123	False	False
0	True	True
7	True	True
1221	True	True
1001	True	True
-121	False	Incorrect
10	False	False

Improved Version (Fixing the Issue)

```python
def is_palindrome(n):
    # Convert negative number to positive
    n = abs(n)

    original = n
    reverse = 0

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

    return original == reverse


# Take input from user
n = int(input("Enter a number: "))

# Check palindrome
if is_palindrome(n):
    print("The number is a palindrome")
else:
    print("The number is not a palindrome")
```

OUTPUT:

```
Enter a number: -121
The number is a palindrome
```

EXPLANATION:

This program checks whether a given number is a palindrome by first converting any negative input into a positive number using the absolute value. It then reverses the number using arithmetic operations without converting it into a string. The reversed number is compared with the original value to determine whether they are the same. If both values match, the number is identified as a palindrome. This approach allows the program to handle both positive and negative inputs effectively.

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.

ONE SHOT:

PROMPT:

Example:

Input: 5

Output: 120

Now write a Python function that takes an integer as input and returns the factorial of that number.Handle invalid inputs such as negative numbers appropriately.

```python
def factorial(n):
    if n < 0:
        return "Factorial is not defined for negative numbers"

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

    return result


# Take input from user
n = int(input("Enter a number: "))

# Call the function and print result
result = factorial(n)
print("Factorial:", result)
```

OUTPUT:

```
Enter a number: -5
Factorial: Factorial is not defined for negative numbers
```

ZERO-SHOT:

PROMPT:

Write a Python function to compute the factorial of a given number.

```python
def factorial(n):
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result


# Take input from user
n = int(input("Enter a non-negative number: "))

# Call the function and print the result
print("Factorial:", factorial(n))
```

OUTPUT:

```
Enter a non-negative number: -5
Factorial: 1
```

Comparison:

Aspect	Zero-Shot Solution	One-Shot Solution
Example provided	No	Yes (5 → 120)
Handling negative input	Not handled	Handled
Code clarity	Basic	More explicit
Correctness	Partial	More robust
Output reliability	Lower	Higher

Improvements in clarity and correctness.

One-shot prompting helps the AI generate clearer and more complete logic by providing an explicit input–output example. This guidance improves output format, correctness, and inclusion of edge-case handling such as negative inputs. Overall, one-shot prompting results in more structured and reliable code compared to zero-shot prompting.

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.

Prompt:

Example 1:

Input: 153

Output: Armstrong Number Example

3:

Input: 123

Output: Not an Armstrong Number

Now write a Python function that takes an integer as input and checks whether it is an Armstrong number. The function should print "Armstrong Number" if it is, otherwise "Not an Armstrong Number".

```python
def is_armstrong(n):
    # Handle negative numbers
    if n < 0:
        print("Not an Armstrong Number")
        return

    # Count number of digits
    num_digits = len(str(n))
    total = 0
    temp = n

    # Calculate sum of powers of digits
    while temp > 0:
        digit = temp % 10
        total += digit ** num_digits
        temp //= 10

    # Check if total equals original number
    if total == n:
        print("Armstrong Number")
    else:
        print("Not an Armstrong Number")


# Take input from user
n = int(input("Enter a number: "))

# Call the function
is_armstrong(n)
```

OUTPUT:

```
Enter a number: 153
Armstrong Number
```

Input	Output
153	Armstrong Number
370	Armstrong Number
123	Not an Armstrong Number

9474 Armstrong Number

9475 Not an Armstrong Number

-153 Not an Armstrong Number

0 Armstrong Number

Analysis: examples clarify the output format (“Armstrong Number” vs “Not an Armstrong Number”)

The AI includes edge-case handling (negative numbers, zero)

Code is more structured, readable, and accurate compared to zero-shot or one-shot prompting

Accuracy improves because AI can generalize from several examples rather than just one

EXPLANATION:

Few-shot prompting improves clarity, correctness, and robustness of the generated code. By providing multiple examples, the AI better understands the task requirements, handles edge cases, and produces code that matches the expected output format reliably.

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.

```python
import math

def classify_number(n):
    # Validate negative numbers and 0,1
    if n <= 1:
        return "Neither"

    # Check for prime using efficient method
    if n == 2:
        return "Prime"
    if n % 2 == 0:
        return "Composite"

    sqrt_n = int(math.sqrt(n)) + 1
    for i in range(3, sqrt_n, 2):
        if n % i == 0:
            return "Composite"
    return "Prime"


# Input validation
try:
    num = int(input("Enter an integer: "))
except ValueError:
    print("Invalid input! Please enter an integer.")
else:
    result = classify_number(num)
    print(f"The number {num} is {result}.")
```

OUTPUT:

```
Enter an integer: 20
The number 20 is Composite.
```

Strategy	Accuracy	Clarity	Optimization	Edge-Case Handling
Zero-Shot	Moderate	Basic	Not optimized	Limited

Strategy	Accuracy	Clarity	Optimization	Edge-Case Handling
One-Shot	Better	Some	Clearer optimization	Limited

Few-Shot	High	Structured	Moderate	Handles examples provided
Context-Managed	Highest	Very Clear	Optimized (sqrt(n))	Handles all edge cases & input validation

EXPLANATION:

Context-managed prompting provides precise instructions and constraints, leading to highly optimized, robust, and readable code. It outperforms zeroshot, one-shot, and few-shot strategies by explicitly guiding the AI on efficiency, input validation, and output formatting.

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.

PROMPT:

Write a Python function that checks whether a given number is a perfect number.

The function should return True if the number is perfect, otherwise False.

Do not provide any examples.

```python
def is_perfect(n):
    if n <= 0:
        return False

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

    return sum_divisors == n


# Take input from user
num = int(input("Enter a number: "))

# Call the function and print result
if is_perfect(num):
    print(f"{num} is a Perfect Number")
else:
    print(f"{num} is Not a Perfect Number")
```

OUTPUT:

```
Enter a number: 6
6 is a Perfect Number
```

Input	Expected Output	Actual Output
6	True	True
28	True	True
12	False	False
1	False	False
0	False	False
-6	False	False

Missing Conditions / Inefficiencies

Inefficient: Checks all numbers from 1 to n-1 → Can be optimized by checking up to n//2 (since no proper divisor is greater than n/2)

Could include input validation to ensure integer input

Could be optimized further using square root method for large numbers

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

PROMPT:

Example 1:

Input: 8

Output: Even

Example 2:

Input: 15

Output: Odd

Example 3:

Input: 0

Output: Even

Now write a Python program that:

1. Takes user input.

2. Validates that the input is an integer.

3. Prints "Even" if the number is even and "Odd" if it is odd.

4. Handles negative numbers correctly.

5. Handles invalid (non-integer) inputs gracefully.

```
# Even or Odd Classification with Input Validation

try:
    num = int(input("Enter a number: "))
except ValueError:
    print("Invalid input! Please enter an integer.")
else:
    if num % 2 == 0:
        print("Even")
    else:
        print("Odd")
```

OUTPUT:

```
Enter a number: 2
Even
```

```
Enter a number: -2
Even
```

ANALYSIS:

Providing multiple input-output examples helps the AI understand the expected behavior and output format more clearly. For instance, showing examples with positive numbers, zero, and odd numbers guides the model to classify numbers correctly in all these cases. It also signals that the program should handle edge cases, such as negative numbers or zero. Furthermore, including examples implicitly emphasizes the importance of consistent output formatting (“Even” vs “Odd”) and encourages input validation to manage noninteger inputs. Overall, examples make the generated code more robust, userfriendly, and aligned with expected results compared to zero-shot prompting.

EXPLANATION:

Few-shot prompting provides multiple input-output examples, which helps the AI understand the task requirements clearly. It guides the program to handle

edge cases like zero and negative numbers correctly. Examples also ensure consistent output formatting, such as printing “Even” or “Odd” as expected. Including these examples encourages input validation, so non-integer inputs are handled gracefully. Overall, few-shot prompting produces code that is robust, accurate, and user-friendly compared to zero-shot prompting.