AI ASSISTED CODING

ASSIGNMENT-3

Name: Harshitha Guda

H.T.No: 2303A51102

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

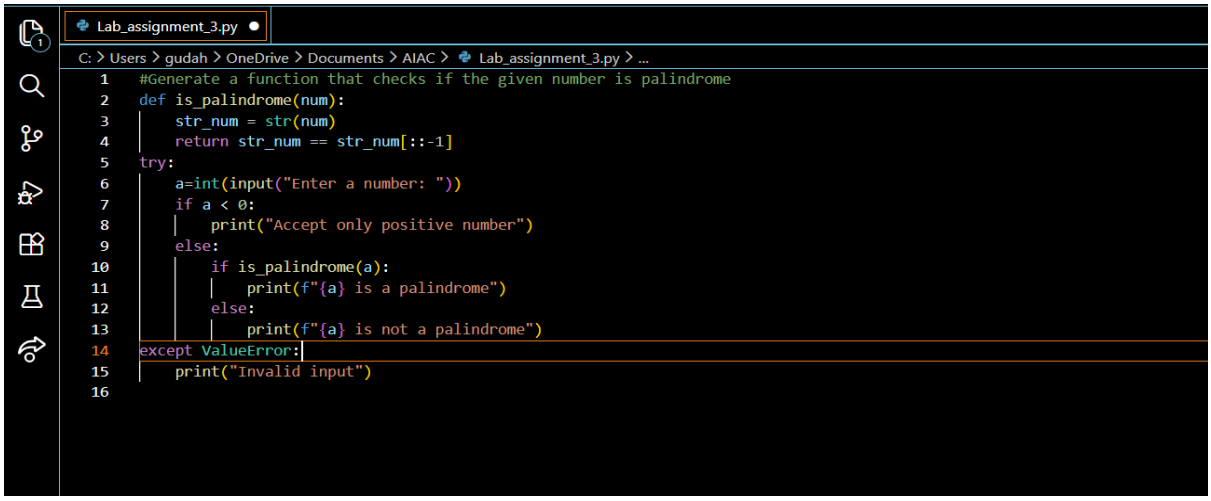
Write a zero-shot prompt (without providing any examples) to generate a Python function that checks whether a given number is a palindrome.

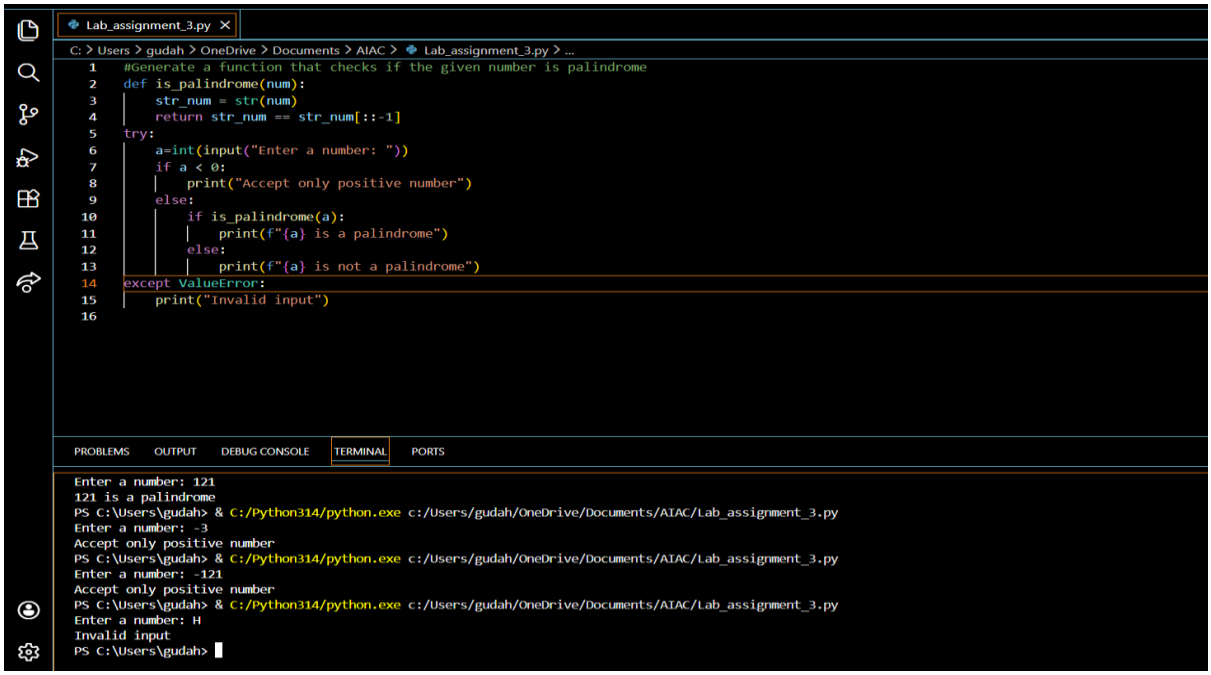
Task:

• Record the AI-generated code.

• Test the code with multiple inputs.

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





There are no logical errors in the code generated. Negative Numbers and invalid input(Strings) are handled using try and except.

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.

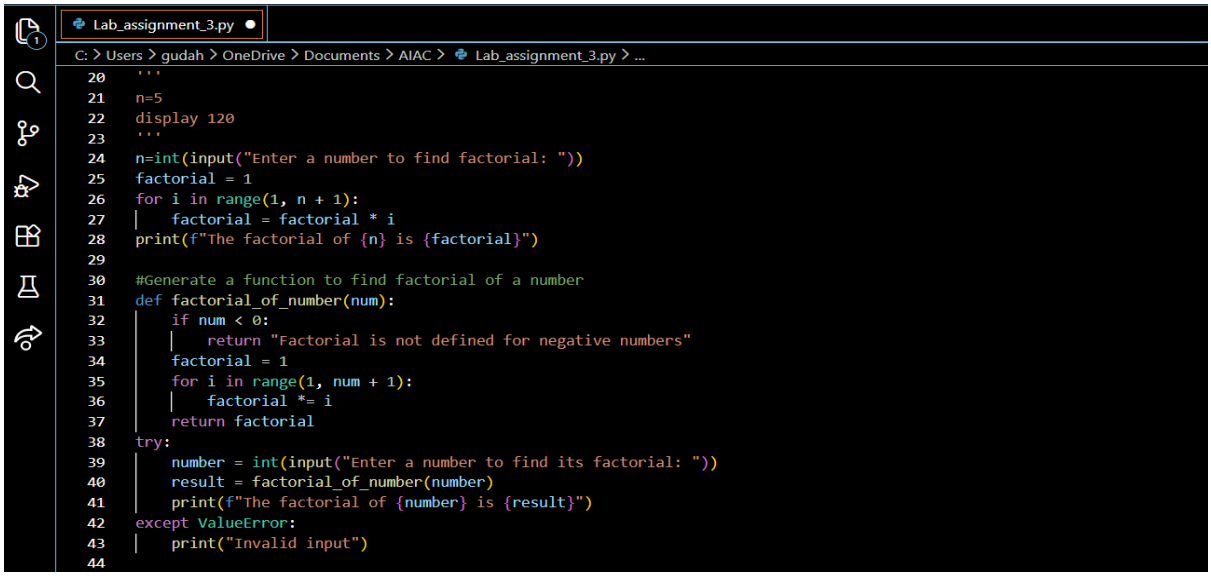
Example:

Input: 5 → Output: 120

Task:

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

• Examine improvements in clarity and correctness.



Comparison of Factorial Programs Using One-Shot and Zero-Shot Prompts

Aspect	Program 1: One-Shot Prompt	Program 2: Zero-Shot Prompt
Definition	Provides a single example and directly computes the factorial	Solves the problem without prior examples using logic and validation
Program Structure	Single block of code	Function-based with main execution logic
Modularity	Not modular	Modular design
Correctness for Positive Integers	Produces correct results	Produces correct results
Handling of Zero (0!)	Correctly returns 1	Correctly returns 1
Handling of Negative Numbers	Not handled; produces incorrect output	Properly handled with validation message
Invalid Input Handling	No handling for invalid input	Uses exception handling (try–except)

Aspect	Program 1: One-Shot Prompt	Program 2: Zero-Shot Prompt
Code Readability	Simple and easy to understand	Clear, structured, and professional
Reusability	Cannot be reused	Can be reused multiple times
Maintainability	Difficult to modify or extend	Easy to maintain and extend
Scalability	Limited	Suitable for larger applications
Programming Best Practices	Partially followed	Fully followed
Suitability for Assignments	Basic demonstration	Highly suitable
Overall Robustness	Low	High

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

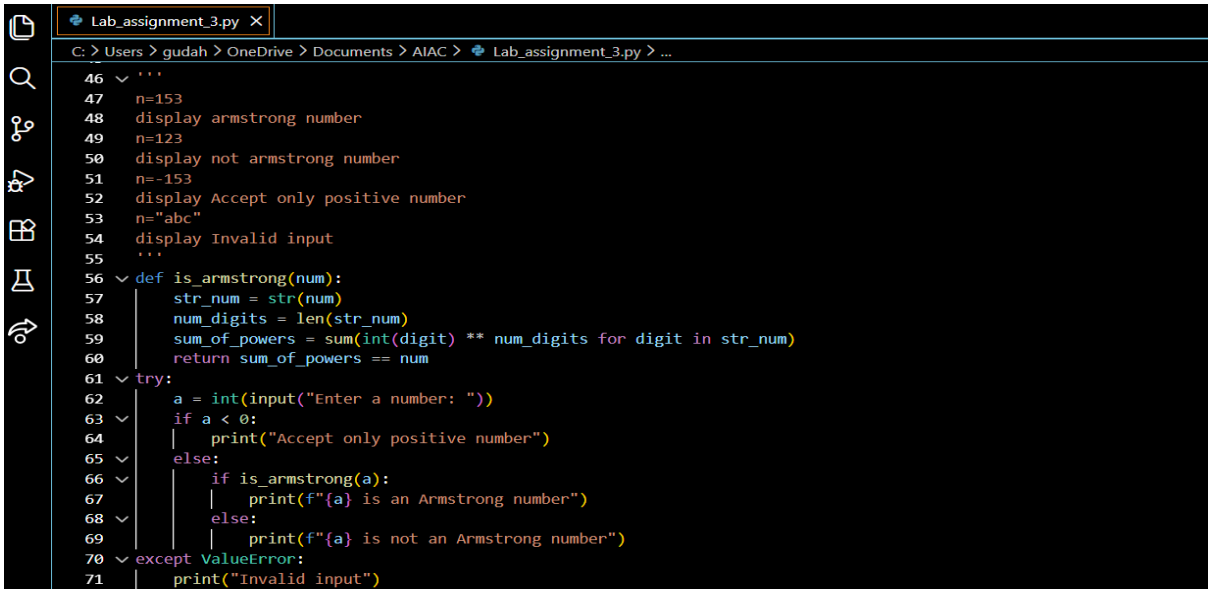
• Input: 123 → Output: Not an Armstrong Number

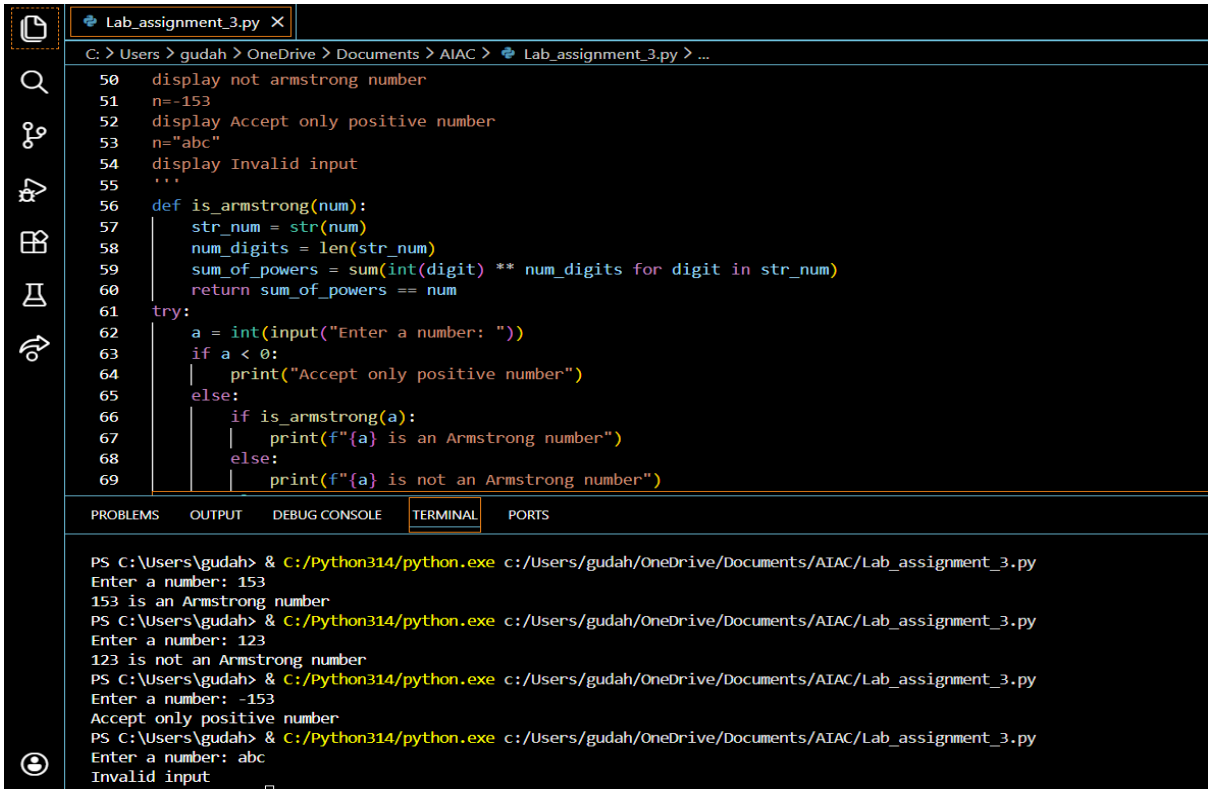
Task:

• Analyze how multiple examples influence code structure and

accuracy.

• Test the function with boundary values and invalid inputs.





Question 4: Context-Managed Prompting (Optimized Number Classification)

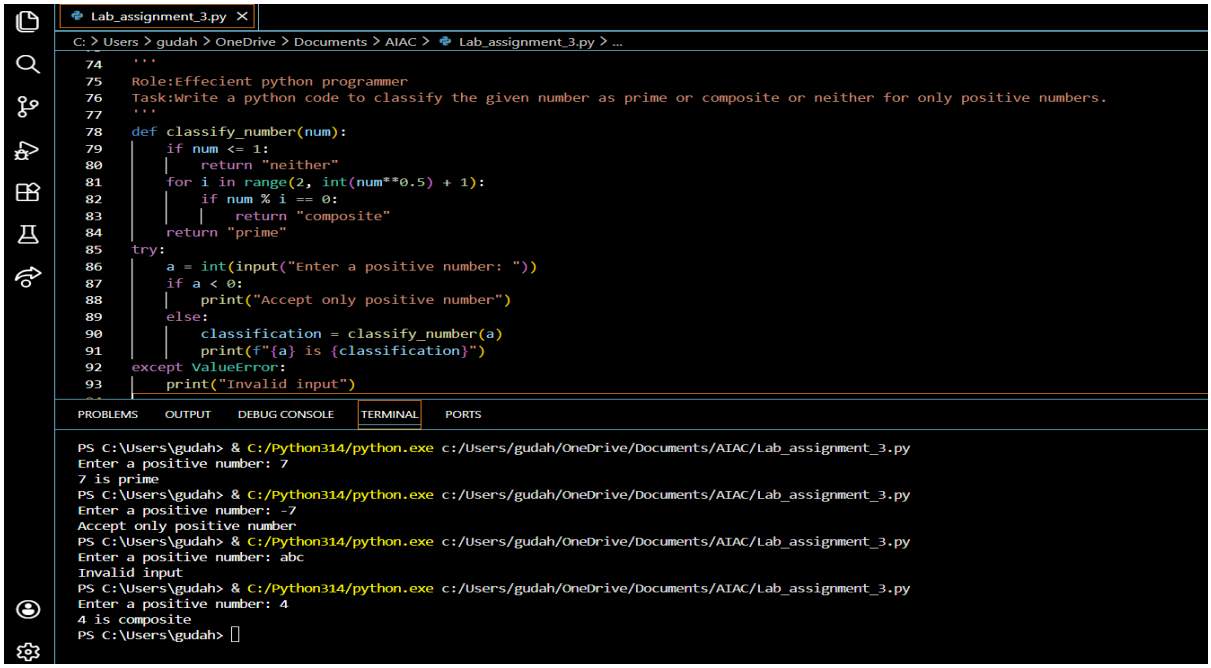
Design a context-managed prompt with clear instructions and constraints to generate an optimized Python program that classifies a number as prime, composite, or neither.

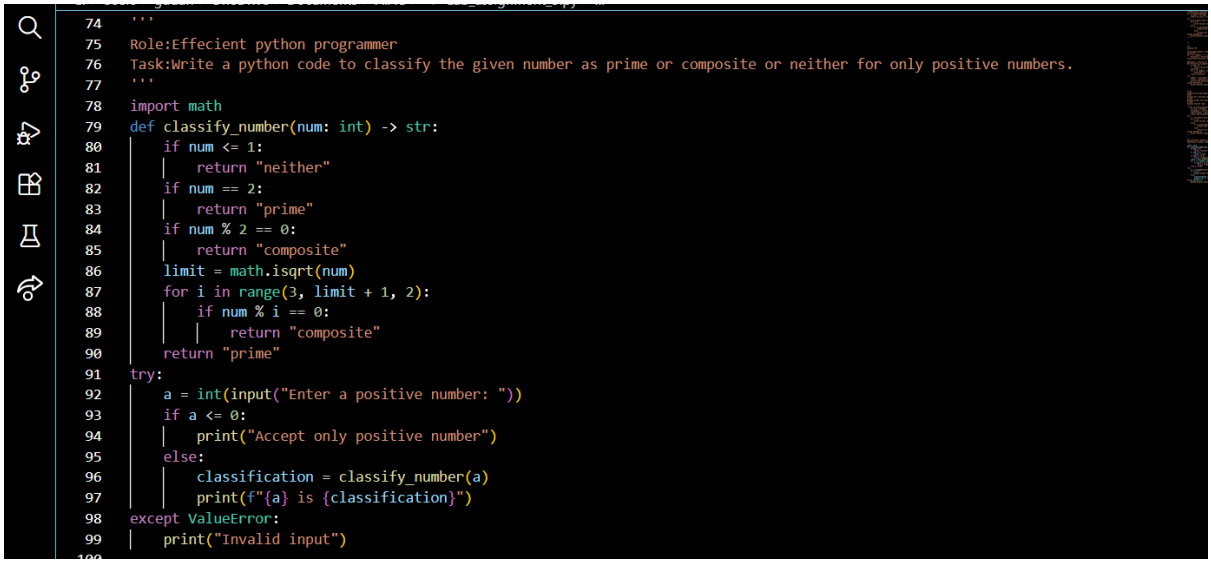
Task:

• Ensure proper input validation.

• Optimize the logic for efficiency.

• Compare the output with earlier prompting strategies.





```python
'''
Role:Effecient python programmer
Task:Write a python code to classify the given number as prime or composite or neither for only positive numbers.
'''
import math
def classify_number(num: int) -> str:
    if num <= 1:
        return "neither"
    if num == 2:
        return "prime"
    if num % 2 == 0:
        return "composite"
    limit = math.isqrt(num)
    for i in range(3, limit + 1, 2):
        if num % i == 0:
            return "composite"
    return "prime"
try:
    a = int(input("Enter a positive number: "))
    if a <= 0:
        print("Accept only positive number")
    else:
        classification = classify_number(a)
        print(f"{a} is {classification}")
except ValueError:
    print("Invalid input")
```

Comparison (optimized vs earlier prompting strategy)

- **Original prompt:** Role:Effecient python programmer Task:Write a python code to classify the given number as prime or composite or neither for only positive numbers.
- **Original outcome:** classify_number used vanilla trial division from 2 to int(num**0.5)+1; accepted zero as positive (if a < 0), no type hints, no stdlib helpers.
- **Optimized changes:**
    - **Algorithm:** early-case for 2, early even-number rejection, loop only odd divisors: for i in range(3, math.isqrt(num)+1, 2).
    - **Implementation:** uses math.isqrt, adds type hints def classify_number(num: int) -> str.
    - **Validation:** input check tightened to reject a <= 0 (zero treated as non-positive).
- **Performance impact:** same asymptotic complexity O(sqrt(n)) but halves divisor checks (skip evens) and uses integer sqrt for a small constant-speed gain; noticeably faster for large odd numbers.
- **Readability / correctness:** clearer intent, fewer iterations, safer integer sqrt, and explicit types improve maintainability.
- **Tradeoffs / next steps:** still brute-force; for very large inputs consider Miller–Rabin (probabilistic, much faster) or optimized sieves if classifying many numbers.

Question 5: Zero-Shot Prompting (Perfect Number Check)

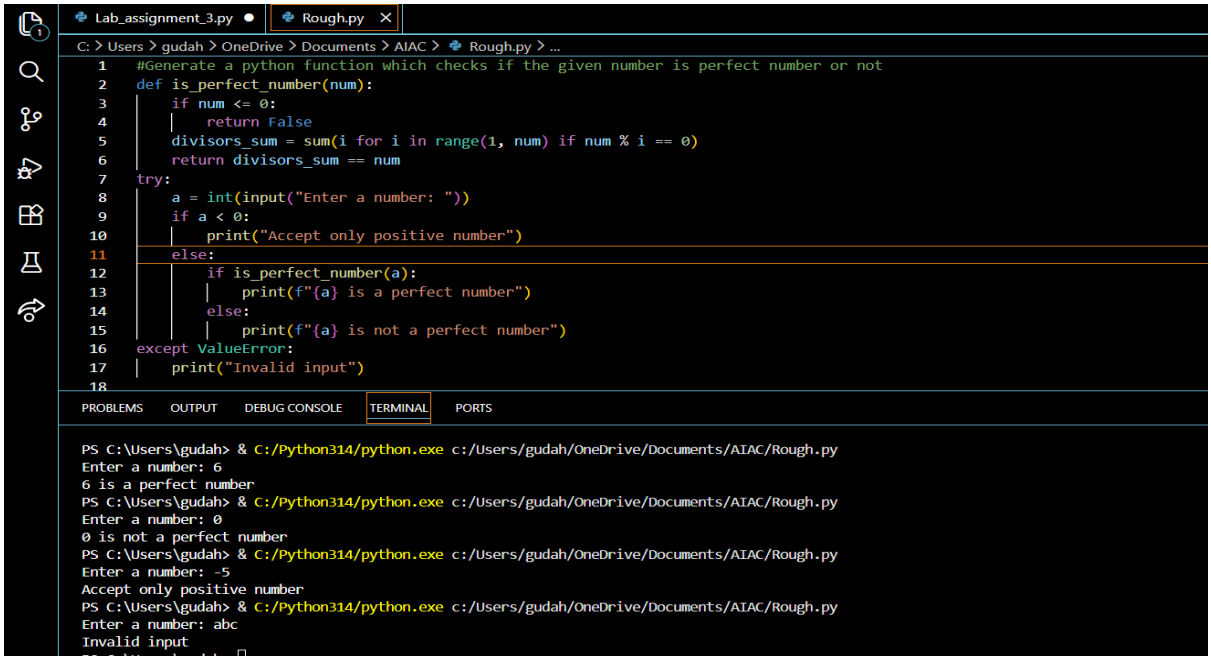
Write a zero-shot prompt (without providing any examples) to generate a Python function that checks whether a given number is a perfect number.

Task:

• Record the AI-generated code.

• Test the program with multiple inputs.

• Identify any missing conditions or inefficiencies in the logic.



1.The program is **logically correct**, but it redundantly checks for negative numbers both inside the function and in the main code.

2.It does not explicitly handle or explain why **0 and 1 are not perfect numbers**.

3.The divisor calculation is **inefficient (O(n))** as it checks all numbers from 1 to num−1.

4.The logic does not use the **square-root optimization**, causing unnecessary iterations.

5.For large inputs, the program may be **slow and inefficient**, even though the result is correct.

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

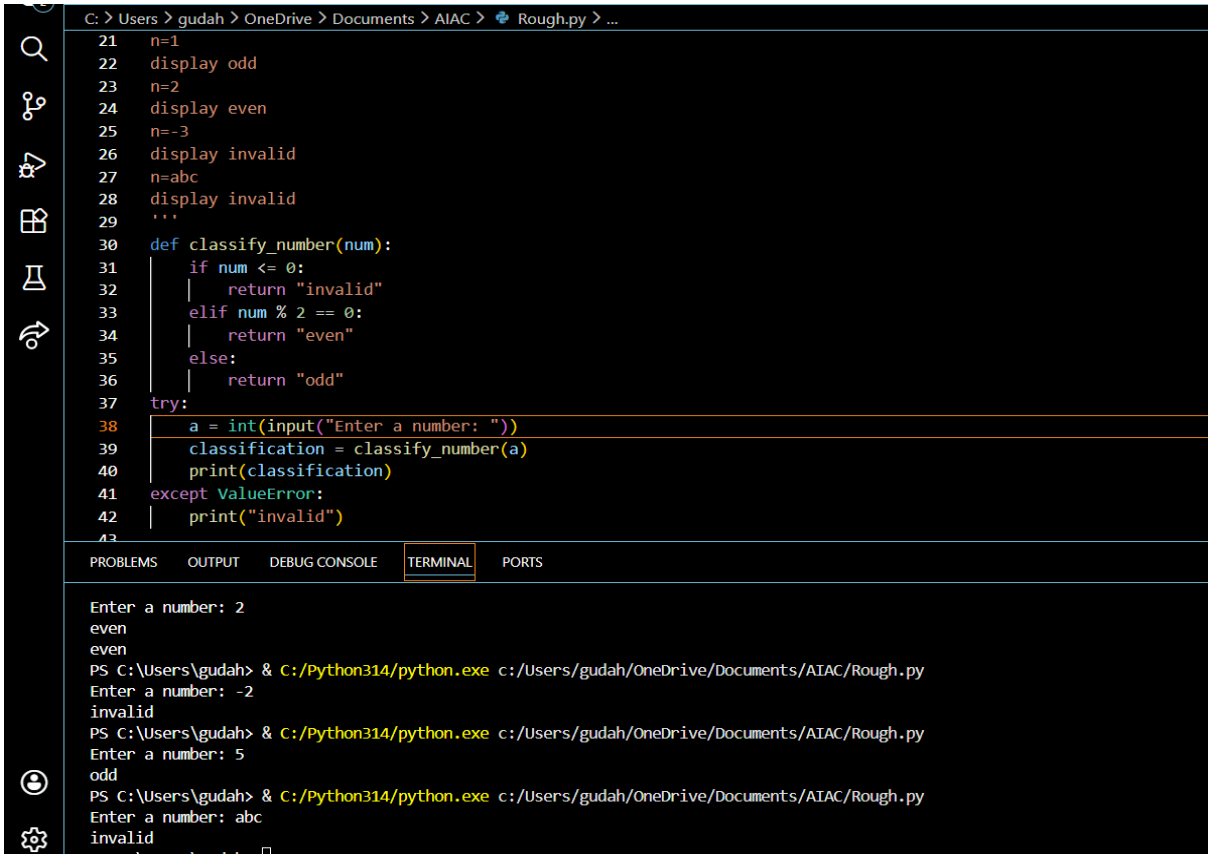
• Input: 0 → Output: Even

Task:

• Analyze how examples improve input handling and output

clarity.

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


C: > Users > gudah > OneDrive > Documents > AIAC > Rough.py > ...
21 n=1
22 display odd
23 n=2
24 display even
25 n=-3
26 display invalid
27 n=abc
28 display invalid
29 '''
30 def classify_number(num):
31 if num <= 0:
32 return "invalid"
33 elif num % 2 == 0:
34 return "even"
35 else:
36 return "odd"
37 try:
38 a = int(input("Enter a number: "))
39 classification = classify_number(a)
40 print(classification)
41 except ValueError:
42 print("invalid")
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Enter a number: 2
even
even
PS C:\Users\gudah> & C:/Python314/python.exe c:/Users/gudah/OneDrive/Documents/AIAC/Rough.py
Enter a number: -2
invalid
PS C:\Users\gudah> & C:/Python314/python.exe c:/Users/gudah/OneDrive/Documents/AIAC/Rough.py
Enter a number: 5
odd
PS C:\Users\gudah> & C:/Python314/python.exe c:/Users/gudah/OneDrive/Documents/AIAC/Rough.py
Enter a number: abc
invalid