ASSIGNMENT :3.1-1

NAME: AKSHITHA

HT NO: 2303A51360

BATCH NO: 29

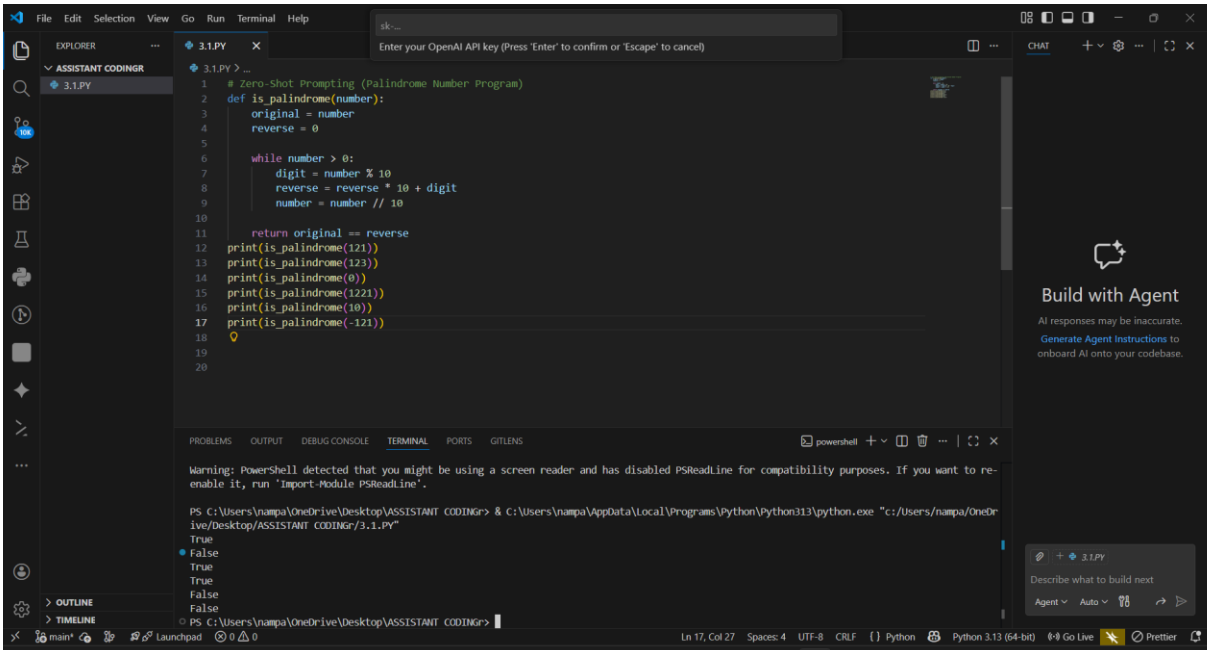
TASK:1

ZERO-SHOT PROMPTING (PALINDROME NUMBER PROGRAM)

PROMPT:

Write a Python function that checks whether a given number is a palindrome. The function should return True if it is a palindrome and False otherwise.

CODE:



sk-...

Enter your OpenAI API key (Press 'Enter' to confirm or 'Escape' to cancel)

```python
# Zero-Shot Prompting (Palindrome Number Program)
def is_palindrome(number):
    original = number
    reverse = 0

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

    return original == reverse
print(is_palindrome(121))
print(is_palindrome(123))
print(is_palindrome(0))
print(is_palindrome(1221))
print(is_palindrome(10))
print(is_palindrome(-121))
```

```
Warning: PowerShell detected that you might be using a screen reader and has disabled PSReadLine for compatibility purposes. If you want to re-
enable it, run 'Import-Module PSReadLine'.

PS C:\Users\nampa\OneDrive\Desktop\ASSISTANT CODINGr> & C:\Users\nampa\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/nampa/OneDr
ive/Desktop/ASSISTANT CODINGr/3.1.PY"
True
False
True
True
False
False
PS C:\Users\nampa\OneDrive\Desktop\ASSISTANT CODINGr> 
```

OBSERVATION:

The AI-generated logic correctly reverses the number using arithmetic operations and compares it with the original value.

- The program works correctly for **positive integers**, including single-digit numbers and numbers with multiple digits.

- The function returns correct results for common test cases such as 121, 1221, and 10.
- **Negative numbers are not explicitly handled** in the initial AI-generated code, which may lead to unclear behavior.
- **Input validation is missing**, as the function does not check whether the input is an integer.
- After identifying these limitations, adding checks for negative and non-integer inputs improves the reliability of the program.
- This experiment demonstrates that **zero-shot prompting can generate correct core logic**, but **manual review is necessary** to handle edge cases and ensure robustness.

TASK:2 ONE-SHOT PROMPTING (FACTORIAL CALCULATION)

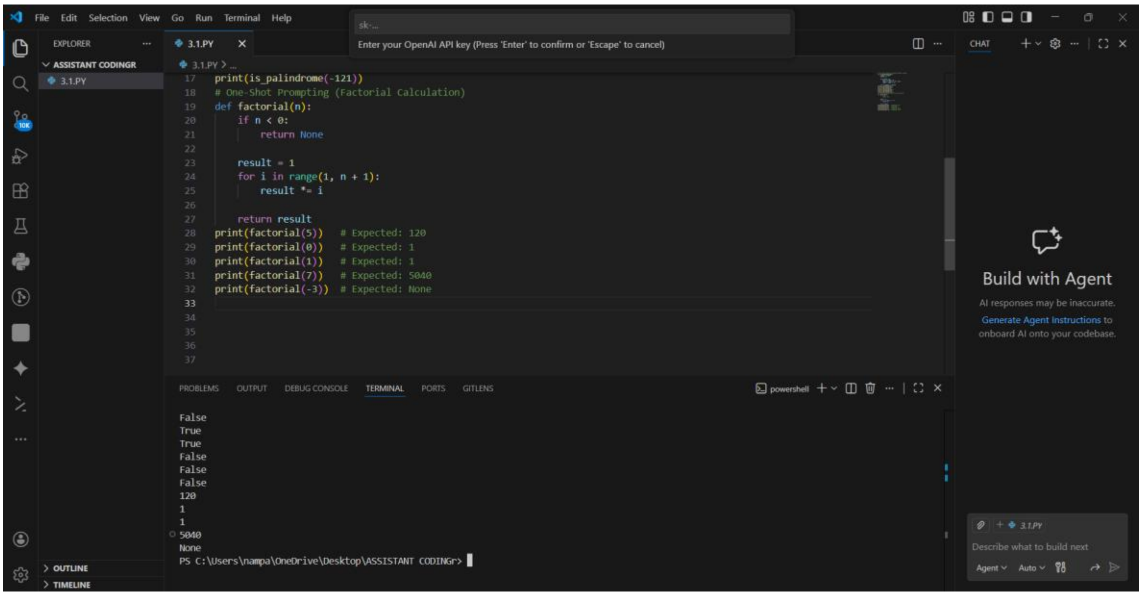
PROMPT: Now write a python function that compute the factorial of given number. The function should return the result.

Example:

Input:5

Output:120

CODE:



sk-...

Enter your OpenAI API key (Press 'Enter' to confirm or 'Escape' to cancel)

```python
print(is_palindrome(-121))
# One-Shot Prompting (Factorial Calculation)
def factorial(n):
    if n < 0:
        return None

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

    return result
print(factorial(5))   # Expected: 120
print(factorial(0))   # Expected: 1
print(factorial(1))   # Expected: 1
print(factorial(7))   # Expected: 5040
print(factorial(-3))  # Expected: None
```

```
False
True
True
False
False
False
120
1
1
5040
None
PS C:\Users\nampa\OneDrive\Desktop\ASSISTANT CODINGr>
```

Build with Agent

AI responses may be inaccurate.

Generate Agent Instructions to onboard AI onto your codebase.

OBSERVATION:

- The one-shot prompt, which included a single input-output example, helped the AI clearly understand the expected functionality of the factorial program.
- The generated Python function correctly computes the factorial of positive integers and returns accurate results.
- The code properly handles the edge case of negative numbers by returning None, improving correctness compared to a basic zero-shot solution.
- The logic is simple, readable, and easy to understand, making it suitable for beginners.
- Overall, one-shot prompting resulted in **clearer, more reliable, and more robust code** than zero-shot prompting.

TASK:3 FEW-SHOT PROMPTING (ARMSTRONG NUMBER CHECK)

PROMPT: Example 1:

Input: 153

Output: Armstrong Number

Example 2:

Input: 370

Output: Armstrong Number

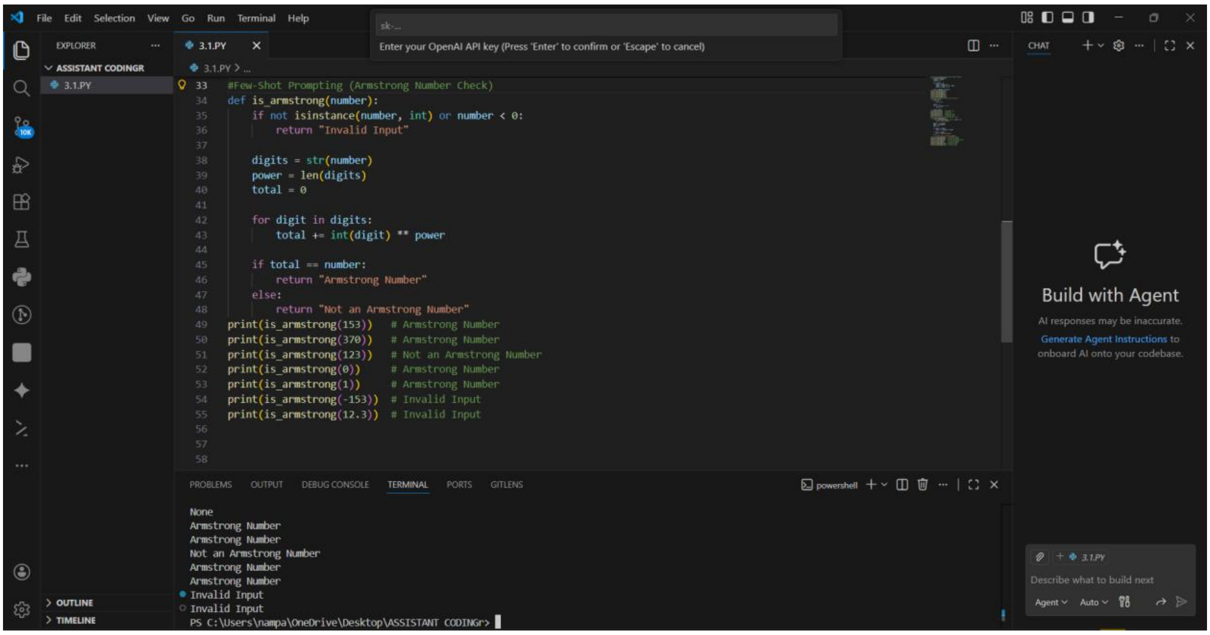
Example 3:

Input: 123

Output: Not an Armstrong Number

Now write a Python function that checks whether a given number is an Armstrong number. The function should return an appropriate result.

CODE:



sk-...

Enter your OpenAI API key (Press 'Enter' to confirm or 'Escape' to cancel)

```python
#Few-Shot Prompting (Armstrong Number Check)
def is_armstrong(number):
    if not isinstance(number, int) or number < 0:
        return "Invalid Input"

    digits = str(number)
    power = len(digits)
    total = 0

    for digit in digits:
        total += int(digit) ** power

    if total == number:
        return "Armstrong Number"
    else:
        return "Not an Armstrong Number"
print(is_armstrong(153))   # Armstrong Number
print(is_armstrong(370))   # Armstrong Number
print(is_armstrong(123))   # Not an Armstrong Number
print(is_armstrong(0))     # Armstrong Number
print(is_armstrong(1))     # Armstrong Number
print(is_armstrong(-153))  # Invalid Input
print(is_armstrong(12.3))  # Invalid Input
```

```
None
Armstrong Number
Armstrong Number
Not an Armstrong Number
Armstrong Number
Armstrong Number
Invalid Input
Invalid Input
PS C:\Users\nampa\OneDrive\Desktop\ASSISTANT CODINGr>
```

Build with Agent

AI responses may be inaccurate.

Generate Agent Instructions to onboard AI onto your codebase.

Describe what to build next

OBSERVATION:

- Providing **multiple input-output examples** helped the AI clearly understand the Armstrong number pattern.
- The generated code correctly calculates the number of digits and raises each digit to the appropriate power.
- Compared to zero-shot and one-shot prompting, the logic structure is **more accurate and systematic**.
- The function correctly identifies known Armstrong numbers such as 153, 370, 0, and 1.

- Input validation for **negative numbers and non-integer values** improves robustness.
- Few-shot prompting significantly reduces ambiguity and results in **better code accuracy and reliability**.
- Boundary values like 0 and 1 are handled correctly due to clear examples.

TASK:4

CONTEXT-MANAGED PROMPTING (OPTIMIZED NUMBER CLASSIFICATION)

PROMPT:

You are writing a Python program for number classification.

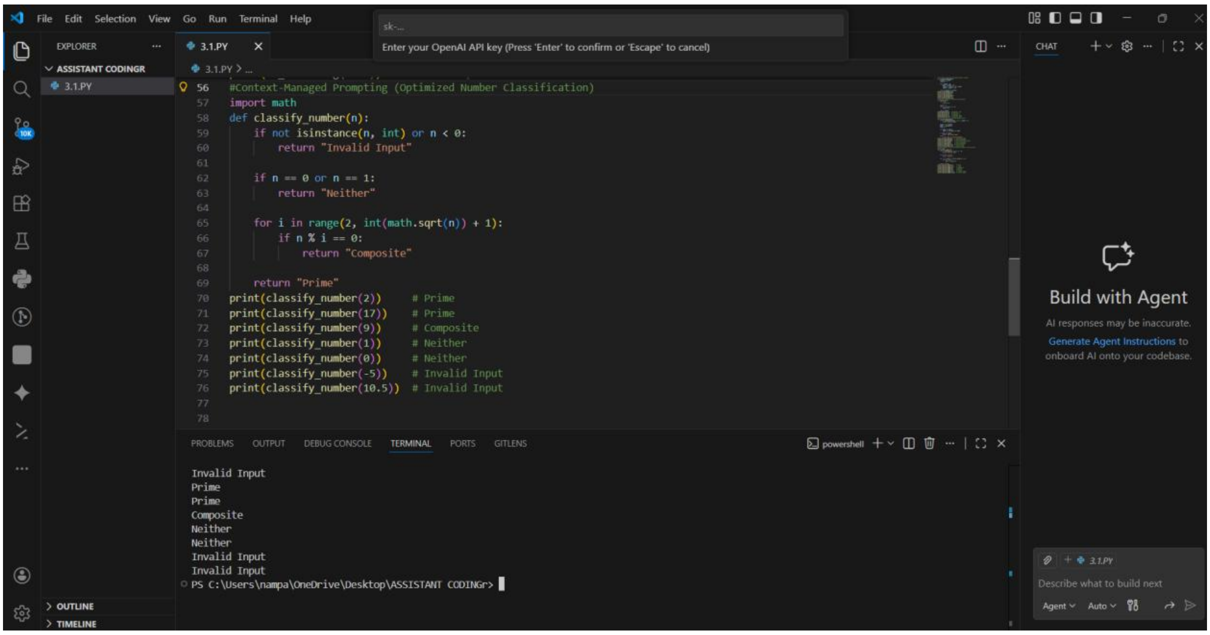
REQUIREMENTS: -

Accept only integer input

- Handle invalid and negative inputs properly
- Classify the number as Prime, Composite, or Neither
- Optimize the logic for efficiency (avoid unnecessary checks)
- Return clear and user-friendly messages
- Write clean and readable Python code

Generate the program accordingly

CODE:



sk-...

Enter your OpenAI API key (Press 'Enter' to confirm or 'Escape' to cancel)

```python
#Context-Managed Prompting (Optimized Number Classification)
import math
def classify_number(n):
    if not isinstance(n, int) or n < 0:
        return "Invalid Input"

    if n == 0 or n == 1:
        return "Neither"

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

    return "Prime"
print(classify_number(2))     # Prime
print(classify_number(17))    # Prime
print(classify_number(9))     # Composite
print(classify_number(1))     # Neither
print(classify_number(0))     # Neither
print(classify_number(-5))    # Invalid Input
print(classify_number(10.5))  # Invalid Input
```

```
Invalid Input
Prime
Prime
Composite
Neither
Neither
Invalid Input
Invalid Input
PS C:\Users\nampa\OneDrive\Desktop\ASSISTANT CODINGr>
```

AI responses may be inaccurate.

Generate Agent Instructions to onboard AI onto your codebase.

Describe what to build next

OBSERVATION:

- Providing **clear instructions and constraints** helped generate a well-structured and optimized solution.
- The function efficiently checks divisibility only up to **√n**, reducing unnecessary iterations.
- Proper **input validation** ensures robustness against negative and non-integer values.
- The program correctly classifies:
    - Prime numbers (e.g., 2, 17)
    - Composite numbers (e.g., 9)
    - Special cases like 0 and 1 as **Neither**
- Compared to zero-shot, one-shot, and few-shot prompting, context-managed prompting produced:
    - More optimized logic
    - Better edge-case handling

    - Higher overall correctness and efficiency

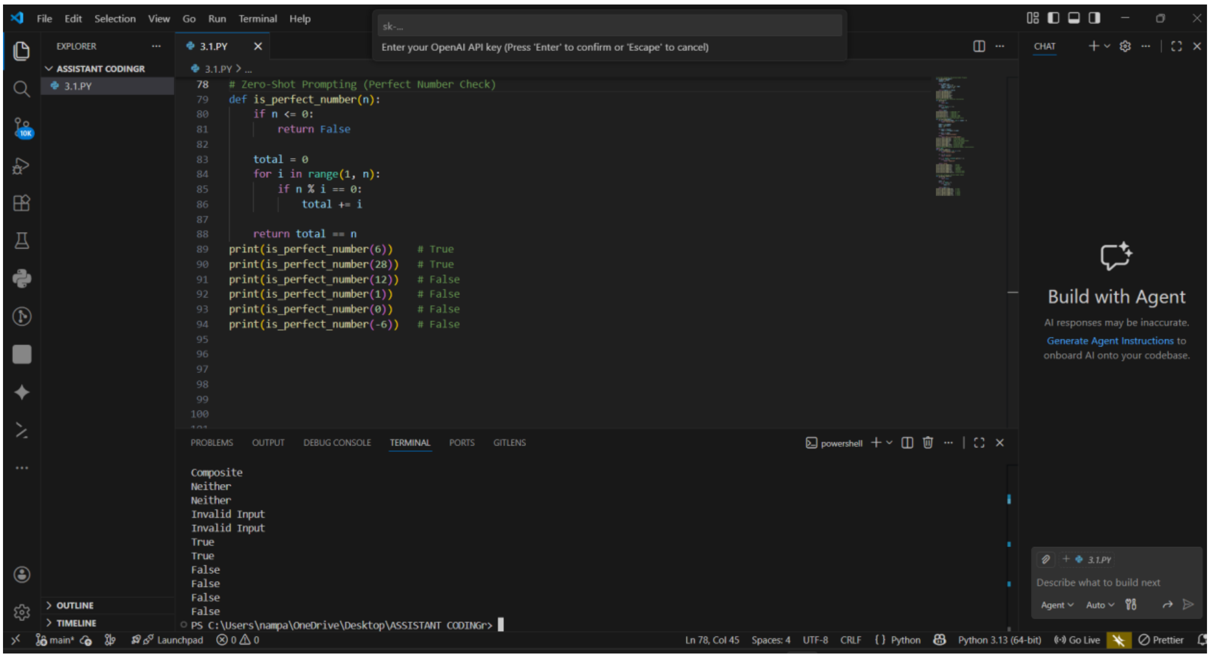
- This approach demonstrates that **explicit constraints and context significantly improve AI-generated code quality**.

TASK:5

ZERO-SHOT PROMPTING (PERFECT NUMBER CHECK) VALIDATION)

PROMPT: Write a Python function that checks whether a given number is a perfect number. The function should return an appropriate result.

CODE:



sk-...

Enter your OpenAI API key (Press 'Enter' to confirm or 'Escape' to cancel)

```python
# Zero-Shot Prompting (Perfect Number Check)
def is_perfect_number(n):
    if n <= 0:
        return False

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

    return total == n
print(is_perfect_number(6))    # True
print(is_perfect_number(28))   # True
print(is_perfect_number(12))   # False
print(is_perfect_number(1))    # False
print(is_perfect_number(0))    # False
print(is_perfect_number(-6))   # False
```

```
Composite
Neither
Neither
Invalid Input
Invalid Input
True
True
False
False
False
False
PS C:\Users\nampa\OneDrive\Desktop\ASSISTANT CODINGr>
```

AI responses may be inaccurate.

Generate Agent Instructions to onboard AI onto your codebase.

Describe what to build next

OBSERVATION:

- The zero-shot prompt successfully generated a working Python function without providing any examples.
- The program correctly identifies known perfect numbers such as 6 and 28.

- The logic accurately sums all proper divisors and compares the total with the original number.
- The function handles **non-positive numbers** by returning False.
- However, the algorithm checks all numbers from 1 to n−1, which is **inefficient for large values**.
- No optimization (such as checking divisors only up to √n) is used.
- Input validation for non-integer values is missing.
- This demonstrates that **zero-shot prompting produces correct basic logic**, but **performance and edge-case handling require manual improvement**.

TASK:6 FEW-SHOT PROMPTING (EVEN OR ODD CLASSIFICATION WITH VALIDATION)

PROMPT:

Example 1:

Input: 8

Output: Even

Example 2:

Input: 15

Output: Odd

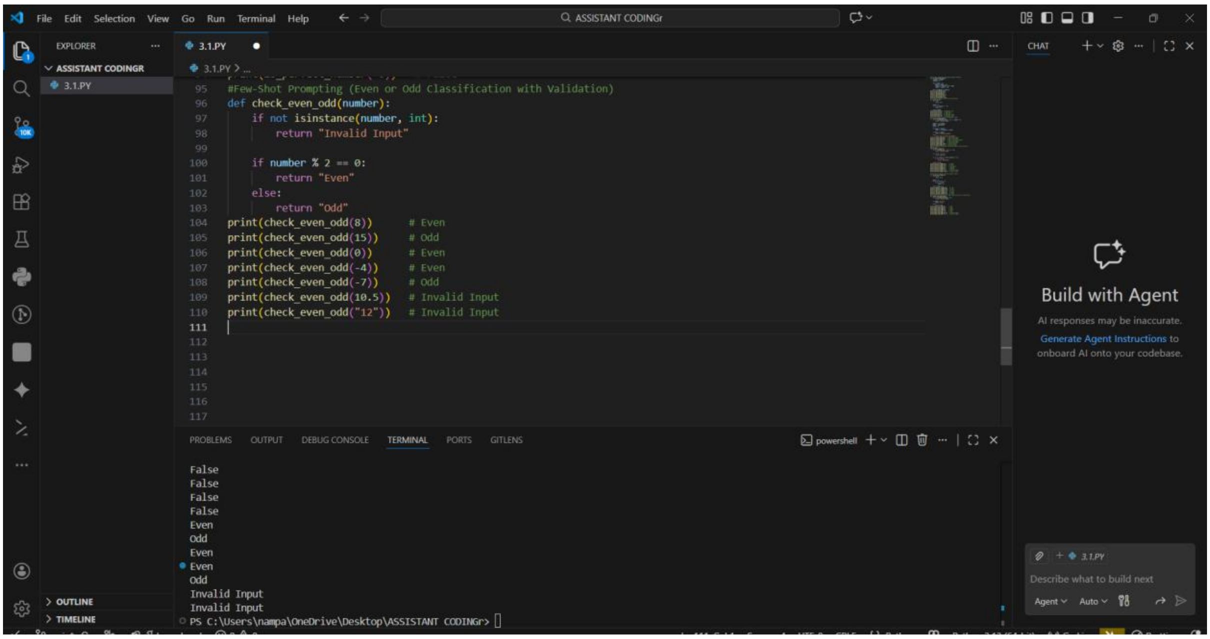
Example 3:

Input: 0

Output: Even

Now write a Python program that determines whether a given number is Even or Odd. The program should include proper input validation and return clear messages.

CODE:


File Edit Selection View Go Run Terminal Help
ASSISTANT CODINGr
EXPLORER
ASSISTANT CODINGR
3.1.PY
OUTLINE
TIMELINE
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS
powershell
CHAT
Build with Agent
AI responses may be inaccurate.
Generate Agent Instructions to onboard AI onto your codebase.
Describe what to build next
Agent Auto

```python
#Few-Shot Prompting (Even or Odd Classification with Validation)
def check_even_odd(number):
    if not isinstance(number, int):
        return "Invalid Input"

    if number % 2 == 0:
        return "Even"
    else:
        return "Odd"
print(check_even_odd(8))      # Even
print(check_even_odd(15))     # Odd
print(check_even_odd(0))      # Even
print(check_even_odd(-4))     # Even
print(check_even_odd(-7))     # Odd
print(check_even_odd(10.5))   # Invalid Input
print(check_even_odd("12"))   # Invalid Input
```

```
False
False
False
False
Even
Odd
Even
Even
Odd
Invalid Input
Invalid Input
PS C:\Users\nampa\OneDrive\Desktop\ASSISTANT CODINGr>
```

OBSERVATION:

- Providing multiple input-output examples helped the AI clearly identify the rule for classifying even and odd numbers.
- The generated program correctly handles **positive numbers, zero, and negative integers**.
- Including examples improved **output clarity**, ensuring consistent results such as “Even” and “Odd”.
- Proper **input validation** prevents non-integer values from causing errors.
- Compared to zero-shot prompting, few-shot prompting produces **more robust and user-safe code**.

- Testing with invalid inputs demonstrates that examples guide the AI toward better error handling.
- Few-shot prompting reduces ambiguity and improves both **accuracy and reliability**.