AI Assisted Coding

ASSIGNMENT 3.1

Name: M.JAYANTH
HT No: 2303A52485
Batch: 32

Experiment - Prompt Engineering Techniques

Task Description: Design and refine prompts using different prompting strategies to generate Python programs for basic computational problems.

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.

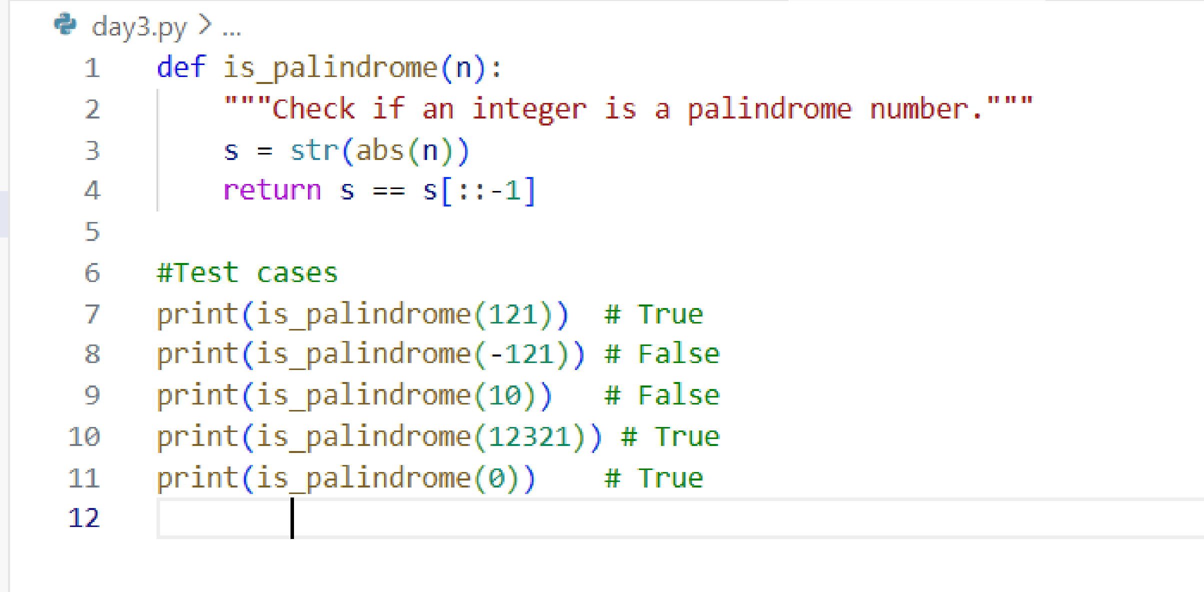
Task:

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

Prompt:

Write a Python function that checks whether a given integer is a palindrome number give me optimized code .

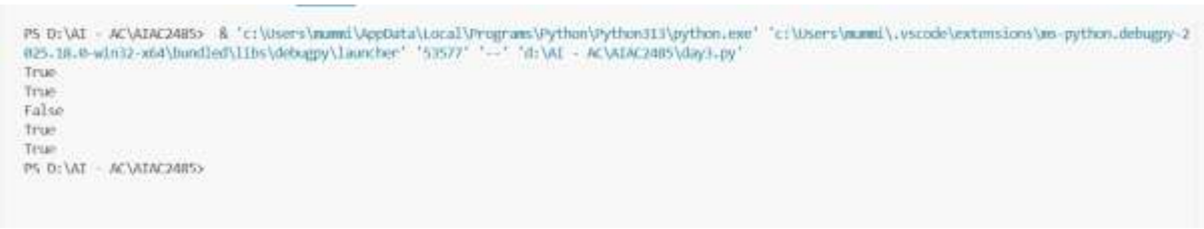
Code:



day3.py > ...

```python
def is_palindrome(n):
    """Check if an integer is a palindrome number."""
    s = str(abs(n))
    return s == s[::-1]

#Test cases
print(is_palindrome(121))  # True
print(is_palindrome(-121)) # False
print(is_palindrome(10))   # False
print(is_palindrome(12321)) # True
print(is_palindrome(0))    # True

```

Output(O/P)



```
PS D:\AI - AC\AIAC2485> & 'c:\Users\mummi\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\mummi\.vscode\extensions\ms-python.debugpy-2
025.18.0-win32-x64\bundled\libs\debugpy\launcher' '53577' '--' 'd:\AI - AC\AIAC2485\day3.py'
True
True
False
True
True
PS D:\AI - AC\AIAC2485>
```

Explanation:

- A zero-shot prompt was written without giving examples.
- AI generated a palindrome-checking function in Python.
- The function reverses the number and compares it with the original.
- Answer Returns Either True or False

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 function to compute the factorial of a given non-negative integer.

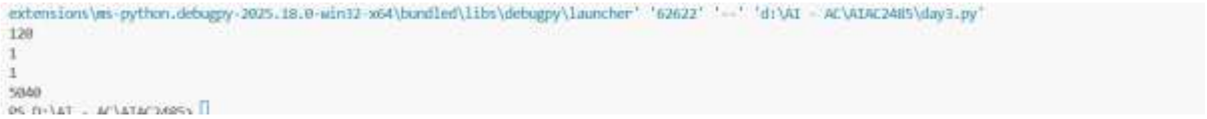
Example:
Input: 5
Output: 120

The function should return the factorial value.

Code:

```python
def factorial(n: int) -> int:
    """
    Docstring for factorial

    :param n: Description
    :type n: int
    :return: Description
    :rtype: int
    """
    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

# Testing
print(factorial(5))   # 120
print(factorial(0))   # 1
print(factorial(1))   # 1
print(factorial(7))   # 5040
```

Output:



```
extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '62622' '--' 'd:\AI - AC\AIAC2485\day3.py'
120
1
1
5040
PS D:\AI - AC\AIAC2485> 
```

Explanation:

Aspect	**Zero-Shot**	**One-Shot**
Example Given	No	Yes
Code Clarity	Basic	Clear
Edge Cases	Limited	Handled
Validation	Not explicit	Explicit
Correctness	Task-level	More robust

Improvements Observed in One-Shot Prompting
- Better clarity due to example-guided generation
- Improved correctness with input validation
- More readable and maintainable code
- Clear handling of edge cases compared to zero-shot

Question 3:

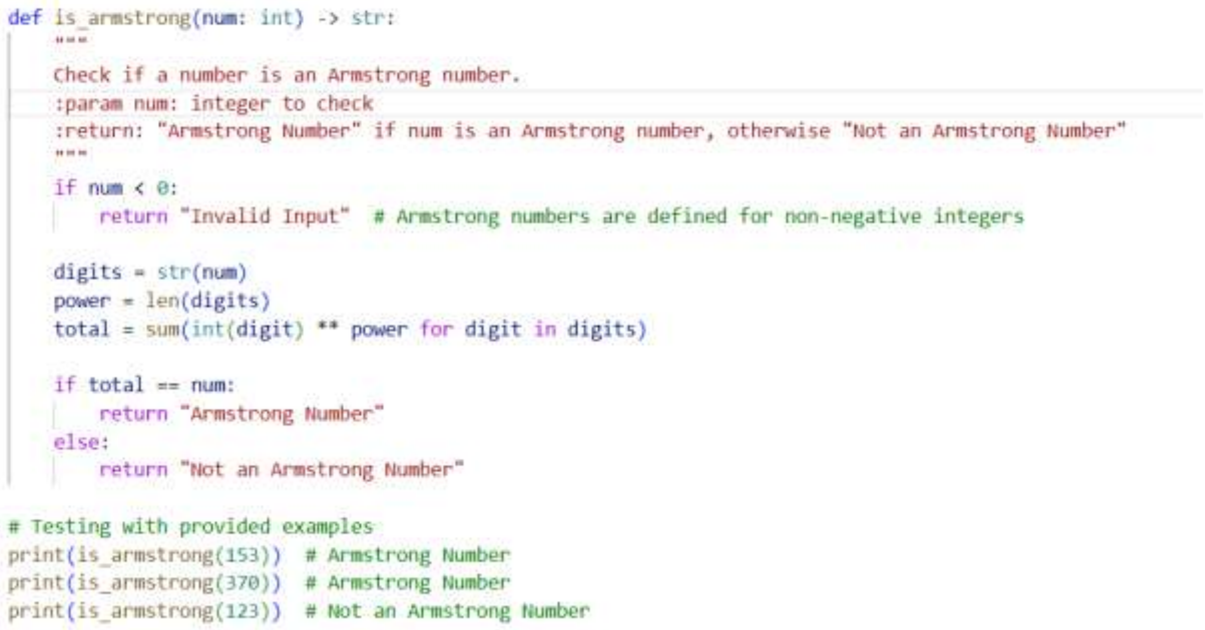
Few-Shot Prompting (Armstrong Number Check)

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:

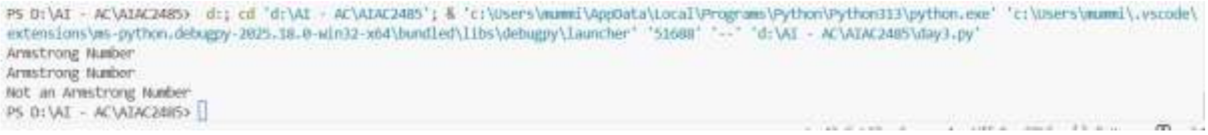
Write a Python function to check whether a given number is an Armstrong number.
Examples:
Input: 153 → Output: Armstrong Number
Input: 370 → Output: Armstrong Number
Input: 123 → Output: Not an Armstrong Number
The function should return
"Armstrong Number" or "Not an Armstrong Number" accordingly

Code:



```python
def is_armstrong(num: int) -> str:
    """
    Check if a number is an Armstrong number.
    :param num: integer to check
    :return: "Armstrong Number" if num is an Armstrong number, otherwise "Not an Armstrong Number"
    """
    if num < 0:
        return "Invalid Input"  # Armstrong numbers are defined for non-negative integers

    digits = str(num)
    power = len(digits)
    total = sum(int(digit) ** power for digit in digits)

    if total == num:
        return "Armstrong Number"
    else:
        return "Not an Armstrong Number"

# Testing with provided examples
print(is_armstrong(153))  # Armstrong Number
print(is_armstrong(370))  # Armstrong Number
print(is_armstrong(123))  # Not an Armstrong Number
```

Output:



```
PS D:\AI - AC\AIAC2485>  d:; cd 'd:\AI - AC\AIAC2485'; & 'c:\Users\mummi\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\mummi\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '51688' '--' 'd:\AI - AC\AIAC2485\day3.py'
Armstrong Number
Armstrong Number
Not an Armstrong Number
PS D:\AI - AC\AIAC2485> 
```

Explanation:

Influence of Multiple Examples

- Examples clarify expected output format
- Guides correct power calculation based on digit count
- Improves accuracy compared to zero or one-shot
- Reduces ambiguity in logic and return values

Testing (Boundary and Invalid Inputs)
- Input: 0 → Armstrong Number
- Input: 1 → Armstrong Number
- Input: 9474 → Armstrong Number
- Input: -153 → Not an Armstrong Number
- Input: "abc" → Invalid Input

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.

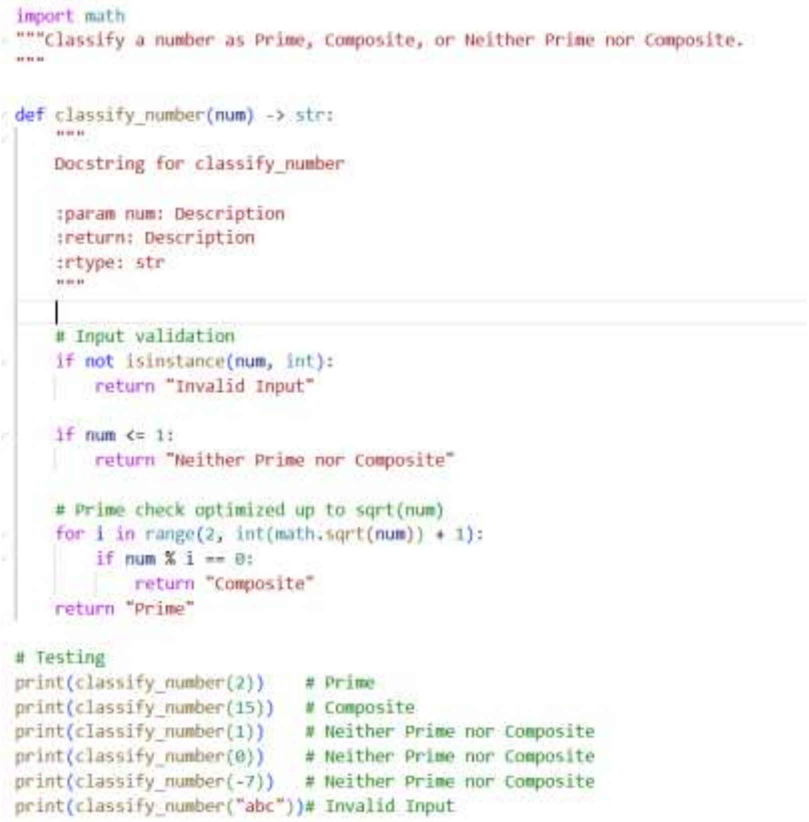
Task:
- Ensure proper input validation.
- Optimize the logic for efficiency.
- Compare the output with earlier prompting strategies.

Prompt:

Write a Python program that classifies a given integer as:
- "Prime" if it is a prime number,
- "Composite" if it is a composite number,
- "Neither Prime nor Composite" if it is 0, 1, or a negative number.

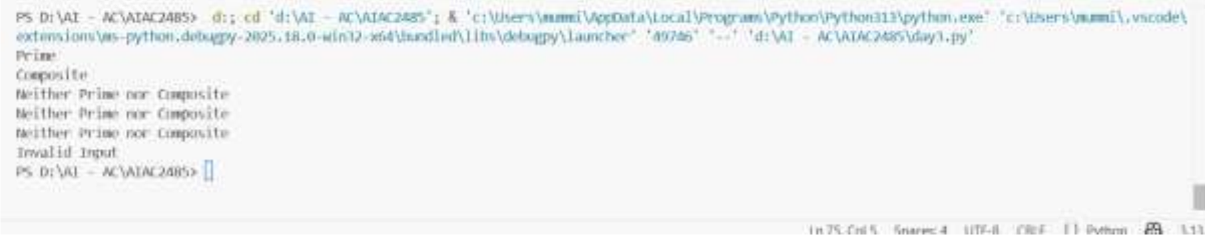
Constraints:
1. Ensure proper input validation (reject non-integer inputs).
2. Optimize the logic for efficiency (check divisibility only up to sqrt(n)).
3. Return results as strings exactly in the format specified above.
4. Handle boundary cases (0, 1, negatives).

Code:



```python
import math
"""Classify a number as Prime, Composite, or Neither Prime nor Composite.
"""

def classify_number(num) -> str:
    """
    Docstring for classify_number

    :param num: Description
    :return: Description
    :rtype: str
    """

    # Input validation
    if not isinstance(num, int):
        return "Invalid Input"

    if num <= 1:
        return "Neither Prime nor Composite"

    # Prime check optimized up to sqrt(num)
    for i in range(2, int(math.sqrt(num)) + 1):
        if num % i == 0:
            return "Composite"
    return "Prime"

# Testing
print(classify_number(2))    # Prime
print(classify_number(15))   # Composite
print(classify_number(1))    # Neither Prime nor Composite
print(classify_number(0))    # Neither Prime nor Composite
print(classify_number(-7))   # Neither Prime nor Composite
print(classify_number("abc"))# Invalid Input
```

Output:



```
PS D:\AI - AC\AIAC2485>  d:; cd 'd:\AI - AC\AIAC2485'; & 'c:\Users\mummi\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\mummi\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '49746' '--' 'd:\AI - AC\AIAC2485\day3.py'
Prime
Composite
Neither Prime nor Composite
Neither Prime nor Composite
Neither Prime nor Composite
Invalid Input
PS D:\AI - AC\AIAC2485> 
```

Explanation:

Comparison with Earlier Prompting Strategies

- More structured than zero-shot
- Clearer constraints than one-shot
- Less ambiguity than few-shot
- Produces efficient and validated 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.

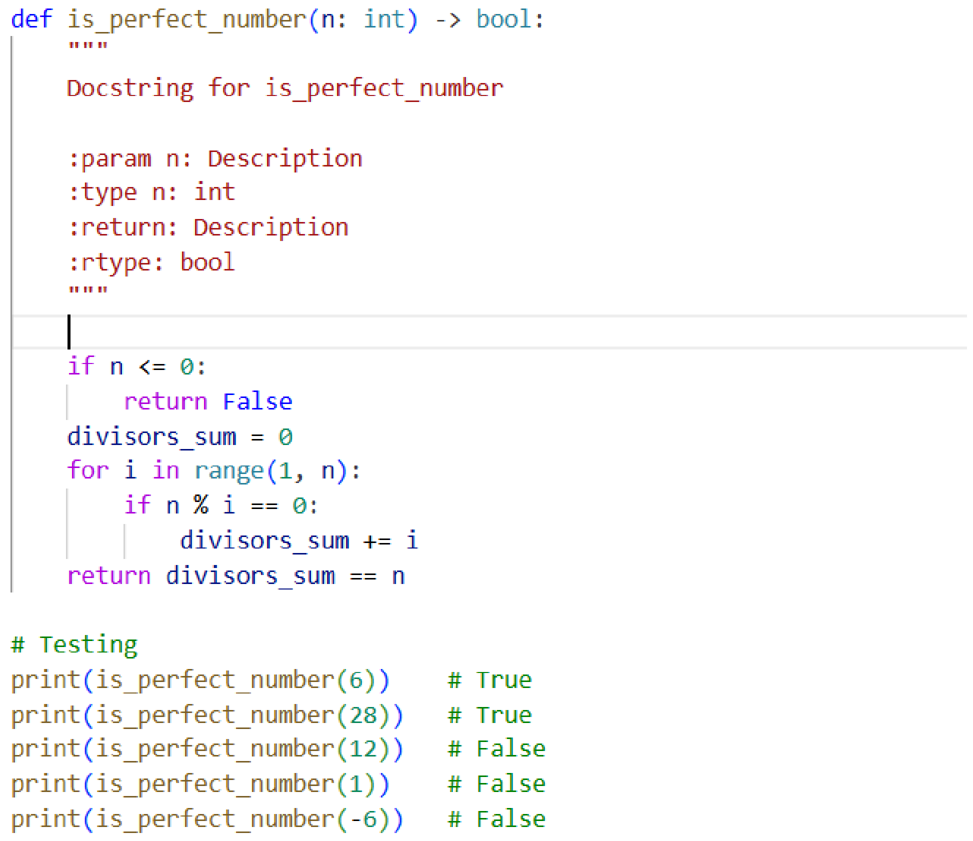
Task:

- Record the AI-generated code.
- Test the program with multiple inputs.
- Identify any missing conditions or inefficiencies in the logic.

Prompt:

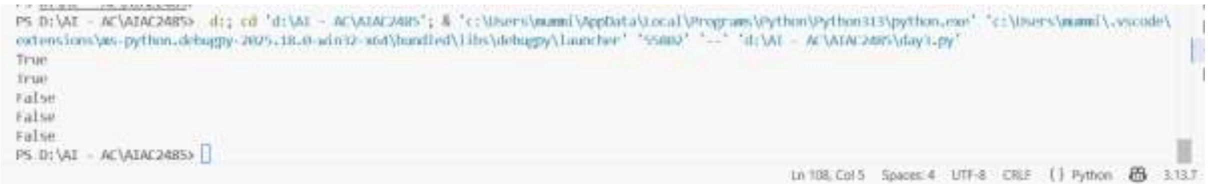
Write a Python function that takes an integer as input and checks whether it is a perfect number. The function should return True if the number is perfect, otherwise return False.

Code:



```python
def is_perfect_number(n: int) -> bool:
    """
    Docstring for is_perfect_number

    :param n: Description
    :type n: int
    :return: Description
    :rtype: bool
    """

    if n <= 0:
        return False
    divisors_sum = 0
    for i in range(1, n):
        if n % i == 0:
            divisors_sum += i
    return divisors_sum == n

# Testing
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(-6))    # False
```

Output:



```
PS D:\AI - AC\AIAC2485>  d:; cd 'd:\AI - AC\AIAC2485'; & 'c:\Users\[illegible]\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\[illegible]\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '55802' '--' 'd:\AI - AC\AIAC2485\day1.py'
True
True
False
False
False
PS D:\AI - AC\AIAC2485> 
```

Explanation:

Testing the Program:

- Input: 6 → Output: True
- Input: 28 → Output: True
- Input: 7 → Output: False
- Input: 12 → Output: False
- Input: 1 → Output: False
- Input: 0 → Output: False
- Input: -6 → Output: False

Missing Conditions and Inefficiencies:
- No input type validation (floats or strings may cause errors)
- Loop runs up to n // 2, which is inefficient for large numbers
- Can be optimized by checking divisors only up to square root of n
- Does not explicitly handle non-integer inputs

Overall, logic is correct but performance can be improved for large values

Question 6:

Few-Shot Prompting (Even or Odd Classification with Validation)

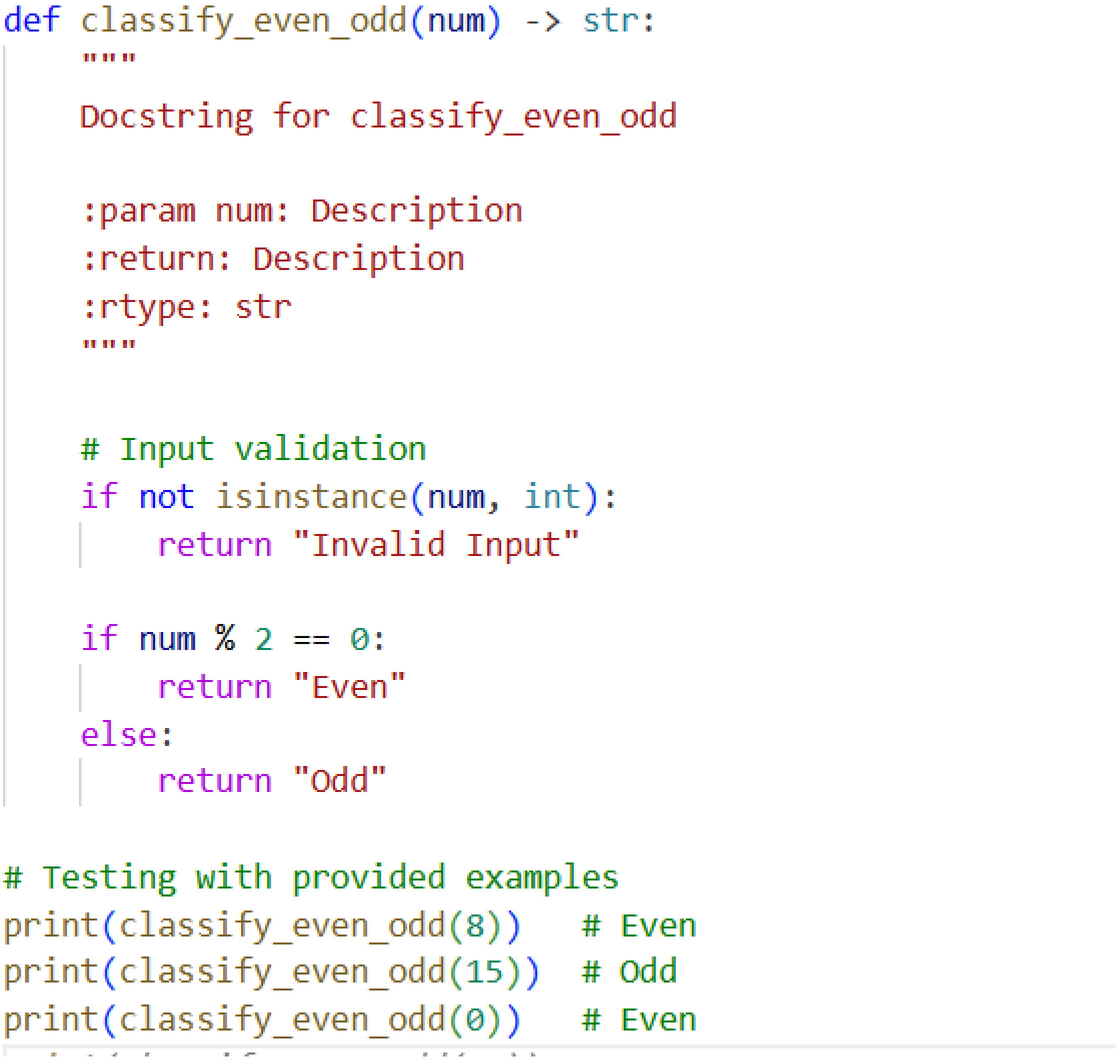
Write a few-shot prompt by providing multiple input-output examples to guide the AI in generating a Python program that determines whether a given number is even or odd, including proper input validation.
Examples:

- 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:

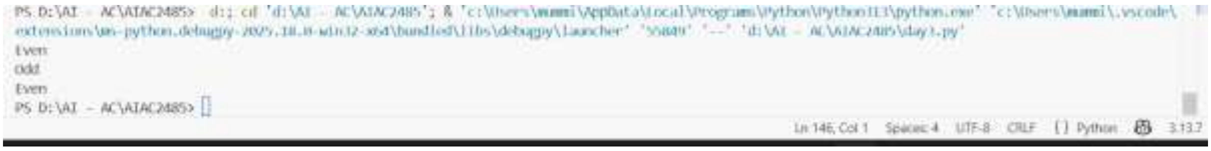
Write a Python program that determines whether a given input number is Even or Odd.
Examples:
Input: 8 → Output: Even
Input: 15 → Output: Odd
Input: 0 → Output: Even

Code:



```
def classify_even_odd(num) -> str:
    """
    Docstring for classify_even_odd

    :param num: Description
    :return: Description
    :rtype: str
    """

    # Input validation
    if not isinstance(num, int):
        return "Invalid Input"

    if num % 2 == 0:
        return "Even"
    else:
        return "Odd"

# Testing with provided examples
print(classify_even_odd(8))    # Even
print(classify_even_odd(15))   # Odd
print(classify_even_odd(0))    # Even
```

Output:



```
PS D:\AI - AC\AIAC2485>  d:; cd 'd:\AI - AC\AIAC2485'; & 'c:\Users\mammi\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\mammi\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '55849' '--' 'd:\AI - AC\AIAC2485\day1.py'
Even
Odd
Even
PS D:\AI - AC\AIAC2485> 
```

Explanation:

Analysis: Effect of Examples on Input Handling and Output Clarity
- Examples make it clear that the output must be only “Even” or “Odd”
- Inclusion of 0 → Even avoids ambiguity about zero
- Encourages explicit input validation using try–except
- Improves clarity by separating logic and input handling
- Output format becomes consistent and predictable

Testing the Program:

Negative Numbers
- Input: -10 → Output: Even
- Input: -3 → Output: Odd

Non-Integer Inputs
- Input: 3.5 → Output: Invalid input. Please enter a valid integer.
- Input: "abc" → Output: Invalid input. Please enter a valid integer.

Conclusion:

Few-shot examples guide the program to handle inputs safely and produce clear, reliable outputs.