HTNO:2303A52089

NAME:N Ashwitha reddy

BATCH:40

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.

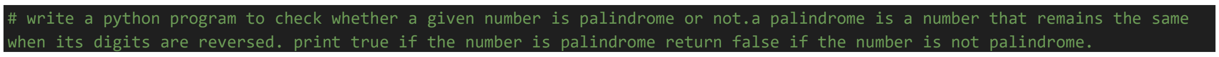
Task:

• Record the AI-generated code.

• Test the code with multiple inputs.

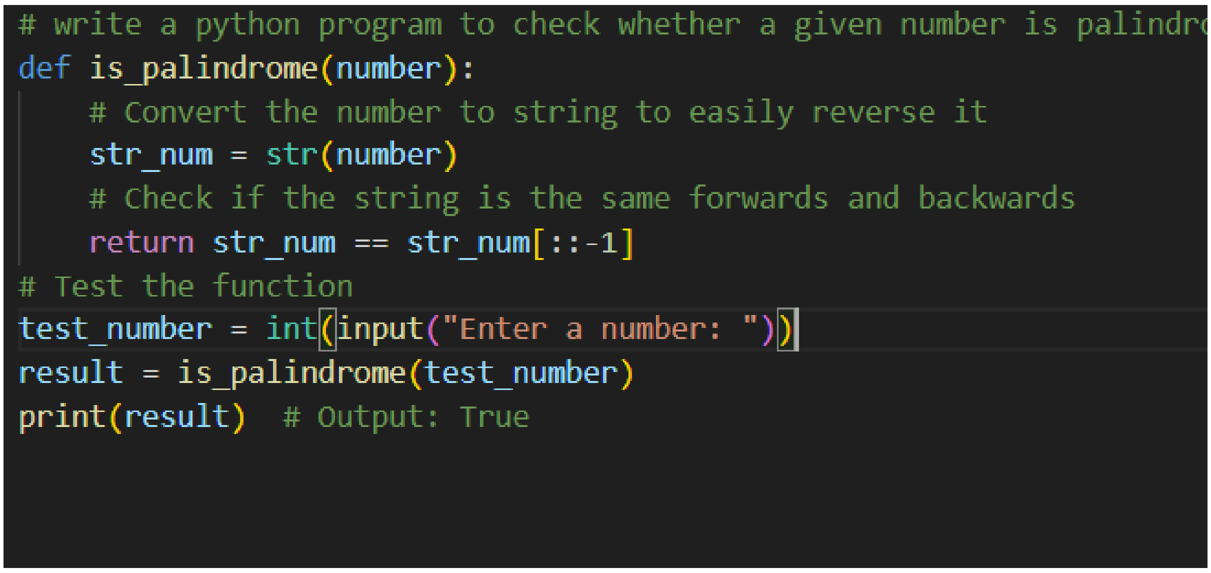
• Identify any logical errors or missing edge-case handling.

PROMPT:



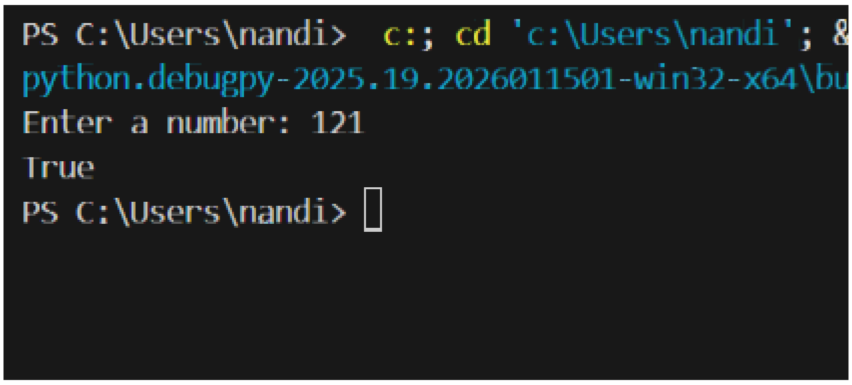
```
# write a python program to check whether a given number is palindrome or not.a palindrome is a number that remains the same
when its digits are reversed. print true if the number is palindrome return false if the number is not palindrome.
```

CODE:



```python
# write a python program to check whether a given number is palindr
def is_palindrome(number):
    # Convert the number to string to easily reverse it
    str_num = str(number)
    # Check if the string is the same forwards and backwards
    return str_num == str_num[::-1]
# Test the function
test_number = int(input("Enter a number: "))
result = is_palindrome(test_number)
print(result)  # Output: True
```

OUTPUT:



```
PS C:\Users\nandi>  c:; cd 'c:\Users\nandi'; &
python.debugpy-2025.19.2026011501-win32-x64\bu
Enter a number: 121
True
PS C:\Users\nandi> 
```

ANALYSIS:

I gave a direct prompt asking the AI to generate a Python function to check whether a number is a palindrome without any examples. The output worked for normal cases, but handling of negative numbers was unclear. The main challenge was missing edge-case validation since zero-shot prompting gives minimal guidance.

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:

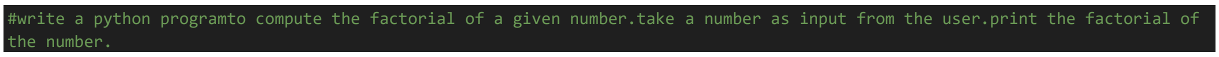
Input: 5 → Output: 120

Task:

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

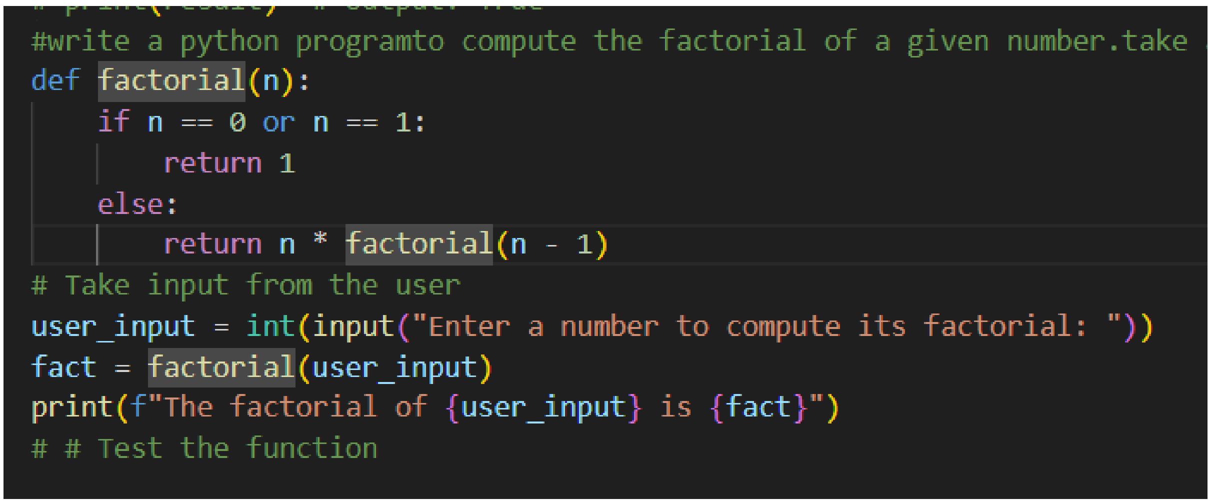
• Examine improvements in clarity and correctness.

PROMPT:



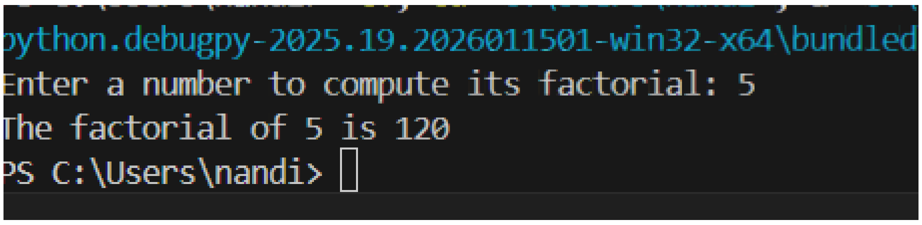
```
#write a python programto compute the factorial of a given number.take a number as input from the user.print the factorial of the number.
```

CODE:



```python
# print(result)  # Output: True
#write a python programto compute the factorial of a given number.take
def factorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n - 1)
# Take input from the user
user_input = int(input("Enter a number to compute its factorial: "))
fact = factorial(user_input)
print(f"The factorial of {user_input} is {fact}")
# # Test the function
```

OUTPUT:



```
python.debugpy-2025.19.2026011501-win32-x64\bundled
Enter a number to compute its factorial: 5
The factorial of 5 is 120
PS C:\Users\nandi> 
```

ANALYSIS:

I provided one input-output example (5 → 120) along with the prompt to guide the AI. The generated code was clearer and correctly handled base cases like 0. Compared to zero-shot, the example improved correctness and reduced logical ambiguity.

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

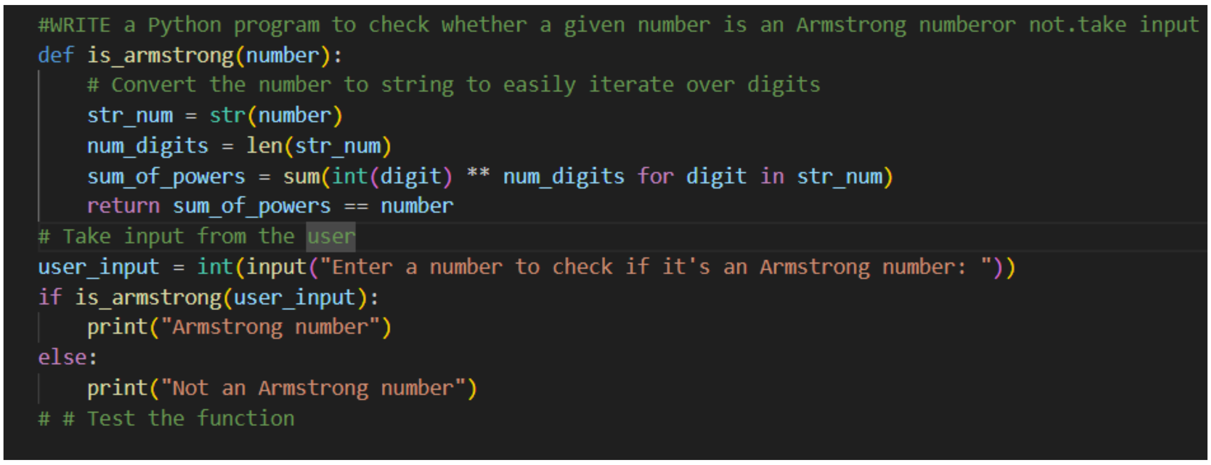
accuracy.

• Test the function with boundary values and invalid inputs.

PROMPT:

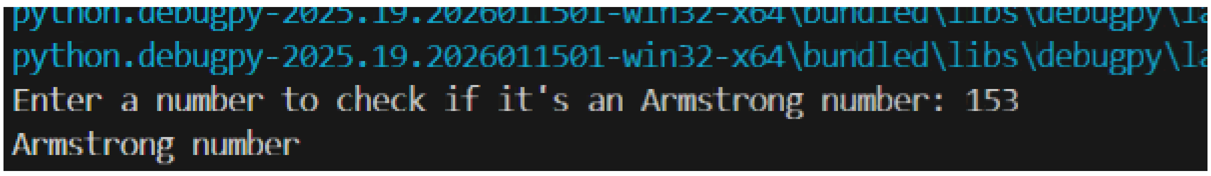
```
#WRITE a Python program to check whether a given number is an Armstrong numberor not.take input from the user. print
"Armstrong number" if the number is an Armstrong number otherwise print "Not an Armstrong number".
```

CODE:



```python
#WRITE a Python program to check whether a given number is an Armstrong numberor not.take input
def is_armstrong(number):
    # Convert the number to string to easily iterate over digits
    str_num = str(number)
    num_digits = len(str_num)
    sum_of_powers = sum(int(digit) ** num_digits for digit in str_num)
    return sum_of_powers == number
# Take input from the user
user_input = int(input("Enter a number to check if it's an Armstrong number: "))
if is_armstrong(user_input):
    print("Armstrong number")
else:
    print("Not an Armstrong number")
# # Test the function
```

OUTPUT:



```
python.debugpy-2025.19.2026011501-win32-x64\bundled\libs\debugpy\la
python.debugpy-2025.19.2026011501-win32-x64\bundled\libs\debugpy\la
Enter a number to check if it's an Armstrong number: 153
Armstrong number
```

ANALYSIS:

I gave multiple input-output examples to clearly define Armstrong number behavior. The AI-generated code followed a structured approach using digit count and power calculations. Few-shot prompting improved accuracy, though validating negative and non-integer inputs remained a challenge.

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.

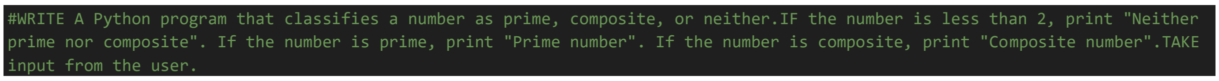
Task:

• Ensure proper input validation.

• Optimize the logic for efficiency.

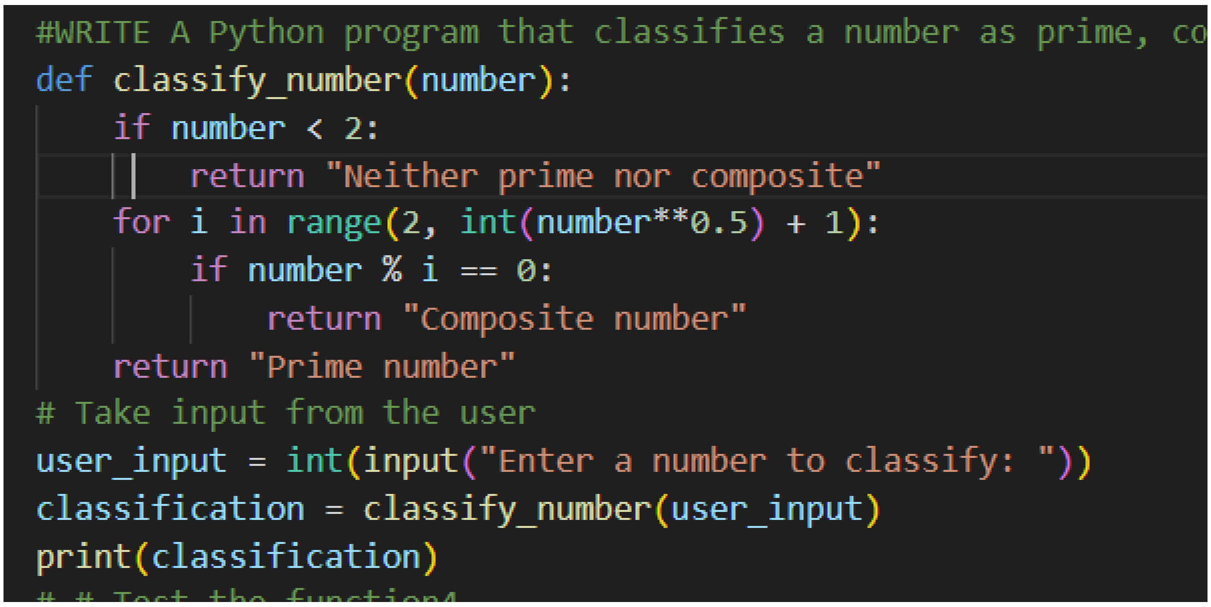
• Compare the output with earlier prompting strategies.

PROMPT:



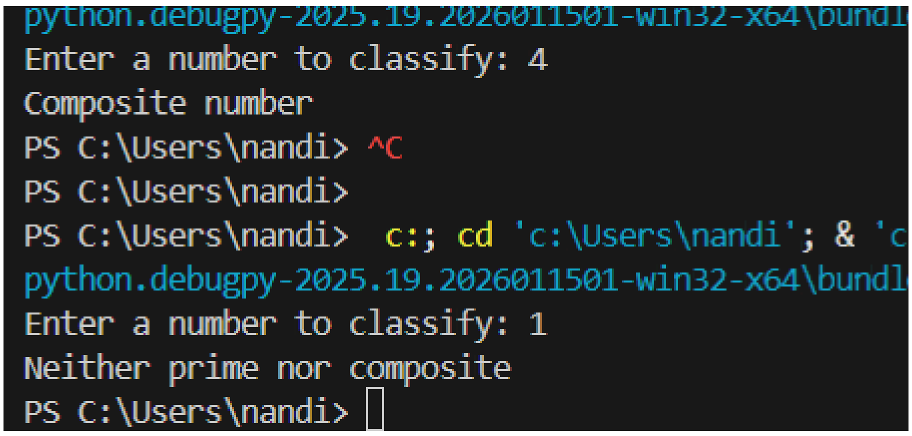
```
#WRITE A Python program that classifies a number as prime, composite, or neither.IF the number is less than 2, print "Neither
prime nor composite". If the number is prime, print "Prime number". If the number is composite, print "Composite number".TAKE
input from the user.
```

CODE:



```python
#WRITE A Python program that classifies a number as prime, co
def classify_number(number):
    if number < 2:
        return "Neither prime nor composite"
    for i in range(2, int(number**0.5) + 1):
        if number % i == 0:
            return "Composite number"
    return "Prime number"
# Take input from the user
user_input = int(input("Enter a number to classify: "))
classification = classify_number(user_input)
print(classification)
# # Test the function
```

OUTPUT:



```
python.debugpy-2025.19.2026011501-win32-x64\bundl
Enter a number to classify: 4
Composite number
PS C:\Users\nandi> ^C
PS C:\Users\nandi>
PS C:\Users\nandi>  c:; cd 'c:\Users\nandi'; & 'c
python.debugpy-2025.19.2026011501-win32-x64\bundl
Enter a number to classify: 1
Neither prime nor composite
PS C:\Users\nandi>
```

ANALYSIS:

I included clear instructions, constraints, and efficiency requirements in the prompt. The AI generated optimized logic using square-root checks and proper validation. Context management helped in producing clean, efficient, and well-structured code compared to earlier prompts.

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.

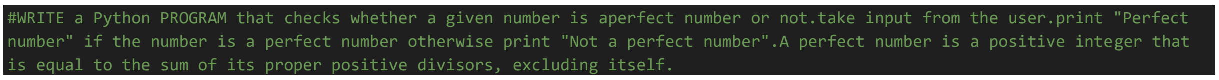
Task:

• Record the AI-generated code.

• Test the program with multiple inputs.

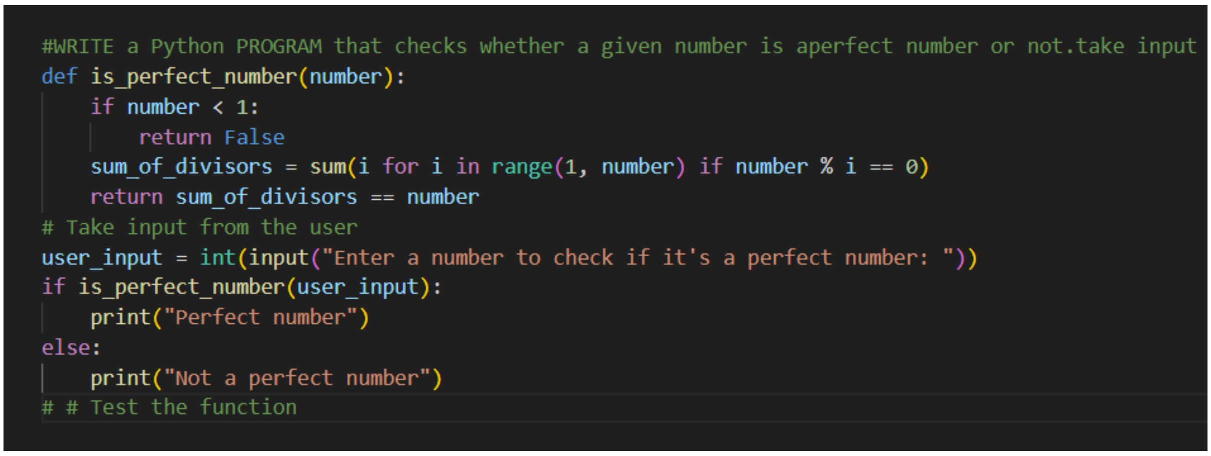
• Identify any missing conditions or inefficiencies in the logic.

PROMPT:



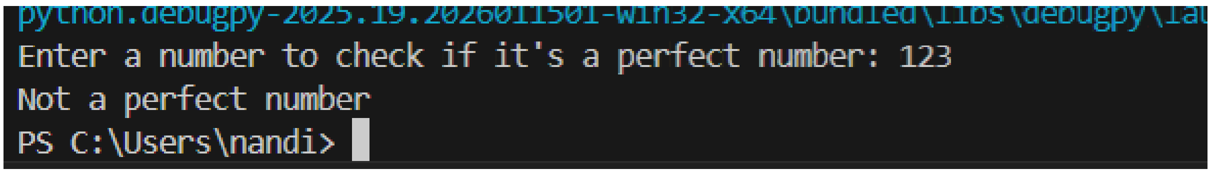
```
#WRITE a Python PROGRAM that checks whether a given number is aperfect number or not.take input from the user.print "Perfect
number" if the number is a perfect number otherwise print "Not a perfect number".A perfect number is a positive integer that
is equal to the sum of its proper positive divisors, excluding itself.
```

CODE:



```python
#WRITE a Python PROGRAM that checks whether a given number is aperfect number or not.take input
def is_perfect_number(number):
    if number < 1:
        return False
    sum_of_divisors = sum(i for i in range(1, number) if number % i == 0)
    return sum_of_divisors == number
# Take input from the user
user_input = int(input("Enter a number to check if it's a perfect number: "))
if is_perfect_number(user_input):
    print("Perfect number")
else:
    print("Not a perfect number")
# # Test the function
```

OUTPUT:



```
python.debugpy-2025.19.2026011501-win32-x64\bundled\libs\debugpy\lau
Enter a number to check if it's a perfect number: 123
Not a perfect number
PS C:\Users\nandi>
```

ANALYSIS:

I asked the AI to generate a perfect number checker without examples. The code correctly calculated divisor sums but was not optimized for large numbers. The main challenge was inefficiency due to checking all divisors.

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

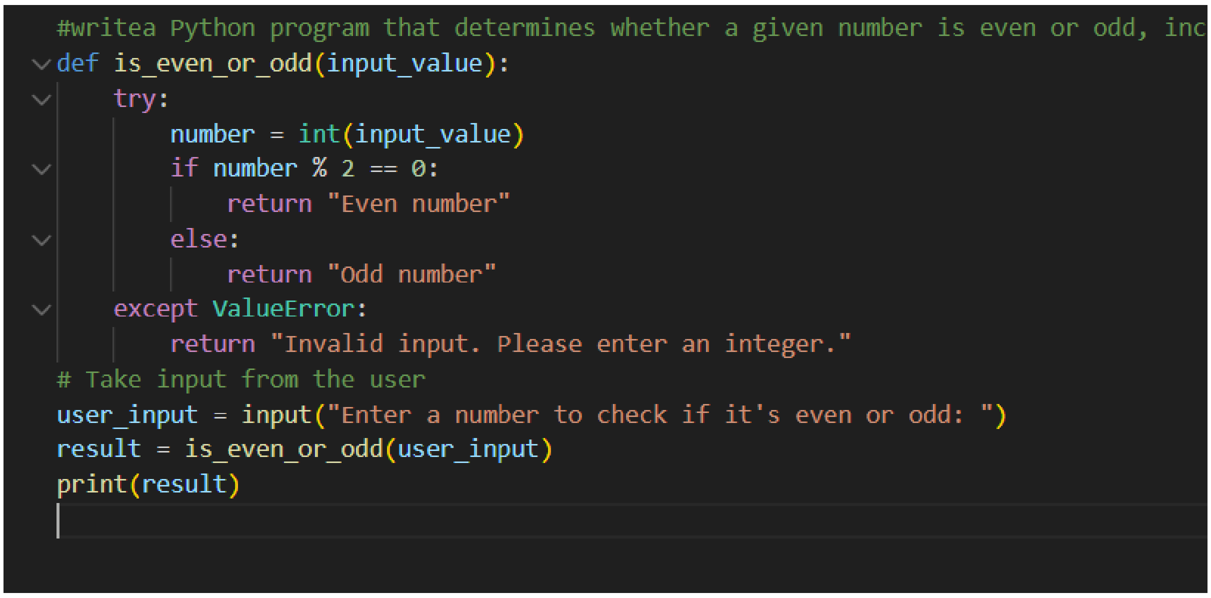
clarity.

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

PROMPT:

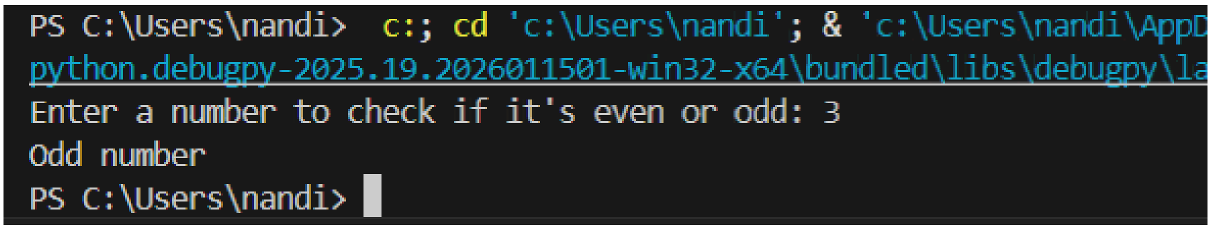
```
#writea Python program that determines whether a given number is even or odd, including proper input validation.take input
from the user.print "Even number" if the number is even otherwise print "Odd number".if the input is not a valid integer,
print "Invalid input. Please enter an integer".
```

CODE:



```python
#writea Python program that determines whether a given number is even or odd, inc
def is_even_or_odd(input_value):
    try:
        number = int(input_value)
        if number % 2 == 0:
            return "Even number"
        else:
            return "Odd number"
    except ValueError:
        return "Invalid input. Please enter an integer."
# Take input from the user
user_input = input("Enter a number to check if it's even or odd: ")
result = is_even_or_odd(user_input)
print(result)
```

OUTPUT:



```
PS C:\Users\nandi>  c:; cd 'c:\Users\nandi'; & 'c:\Users\nandi\AppD
python.debugpy-2025.19.2026011501-win32-x64\bundled\libs\debugpy\la
Enter a number to check if it's even or odd: 3
Odd number
PS C:\Users\nandi>
```

ANALISYS:

I provided multiple examples including zero to guide output behavior. The AI added input validation and correctly handled negative numbers. Few-shot prompting significantly improved clarity, robustness, and error handling for invalid inputs.