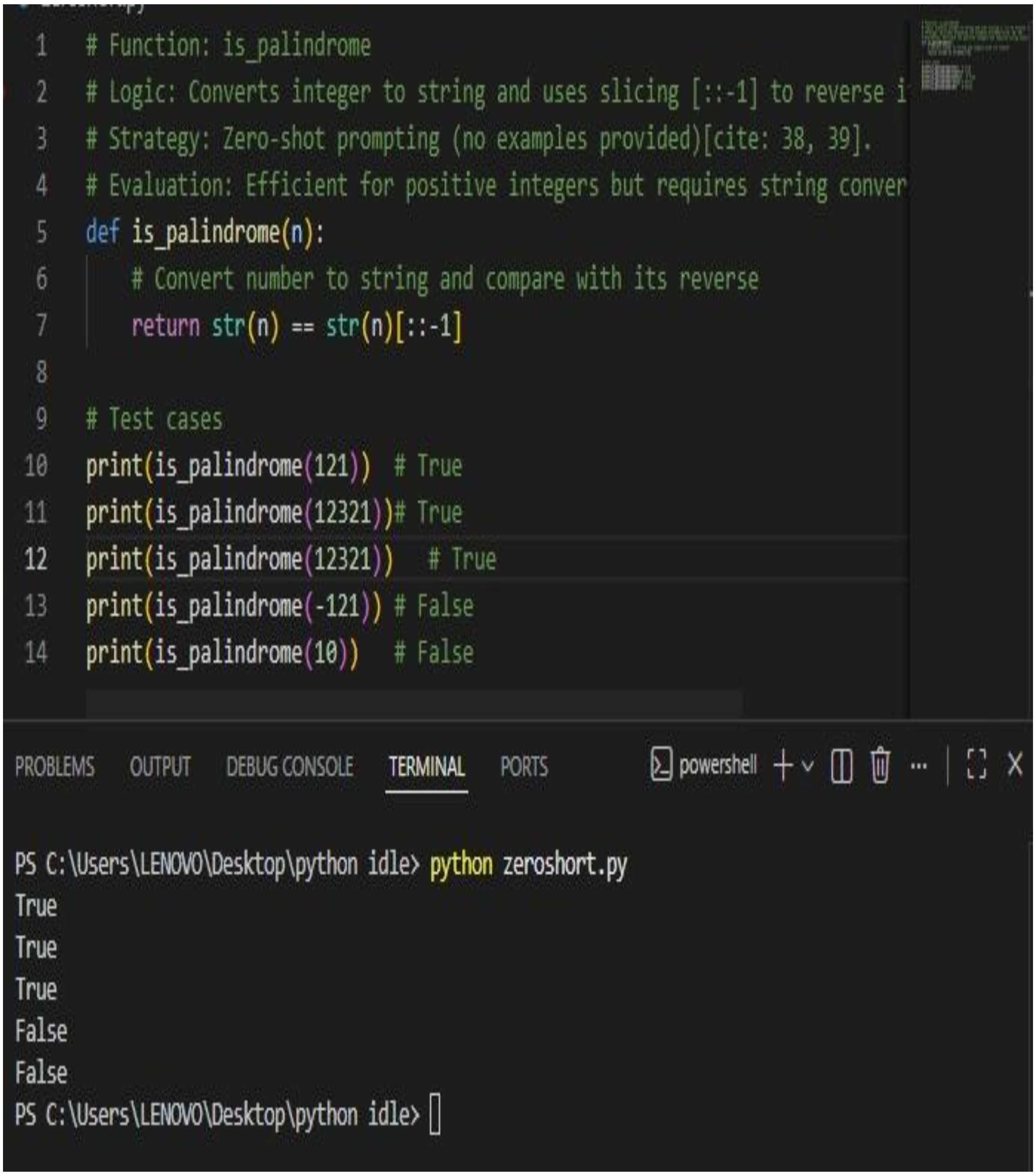
Assignment 3.1

T.akanksh

2303A51078

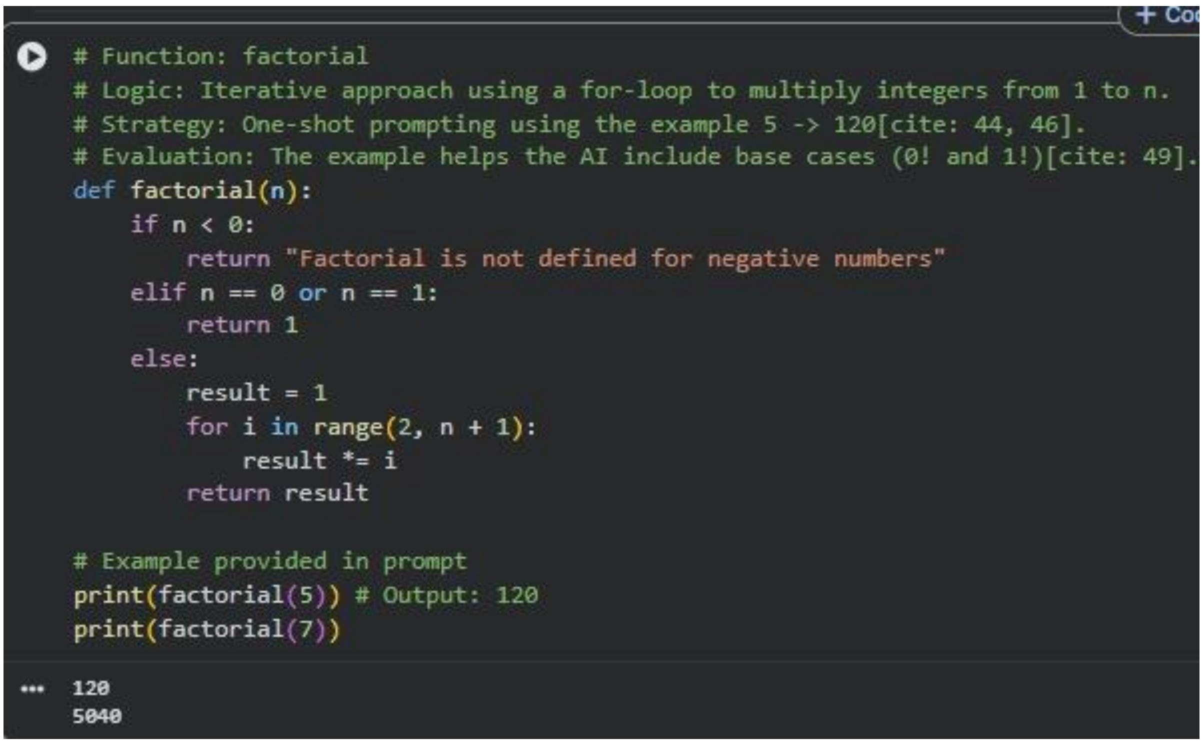
1.Zero-short (palindrome)



```python
# Function: is_palindrome
# Logic: Converts integer to string and uses slicing [::-1] to reverse i
# Strategy: Zero-shot prompting (no examples provided)[cite: 38, 39].
# Evaluation: Efficient for positive integers but requires string conver
def is_palindrome(n):
    # Convert number to string and compare with its reverse
    return str(n) == str(n)[::-1]

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

```
PS C:\Users\LENOVO\Desktop\python idle> python zeroshort.py
True
True
True
False
False
PS C:\Users\LENOVO\Desktop\python idle>
```

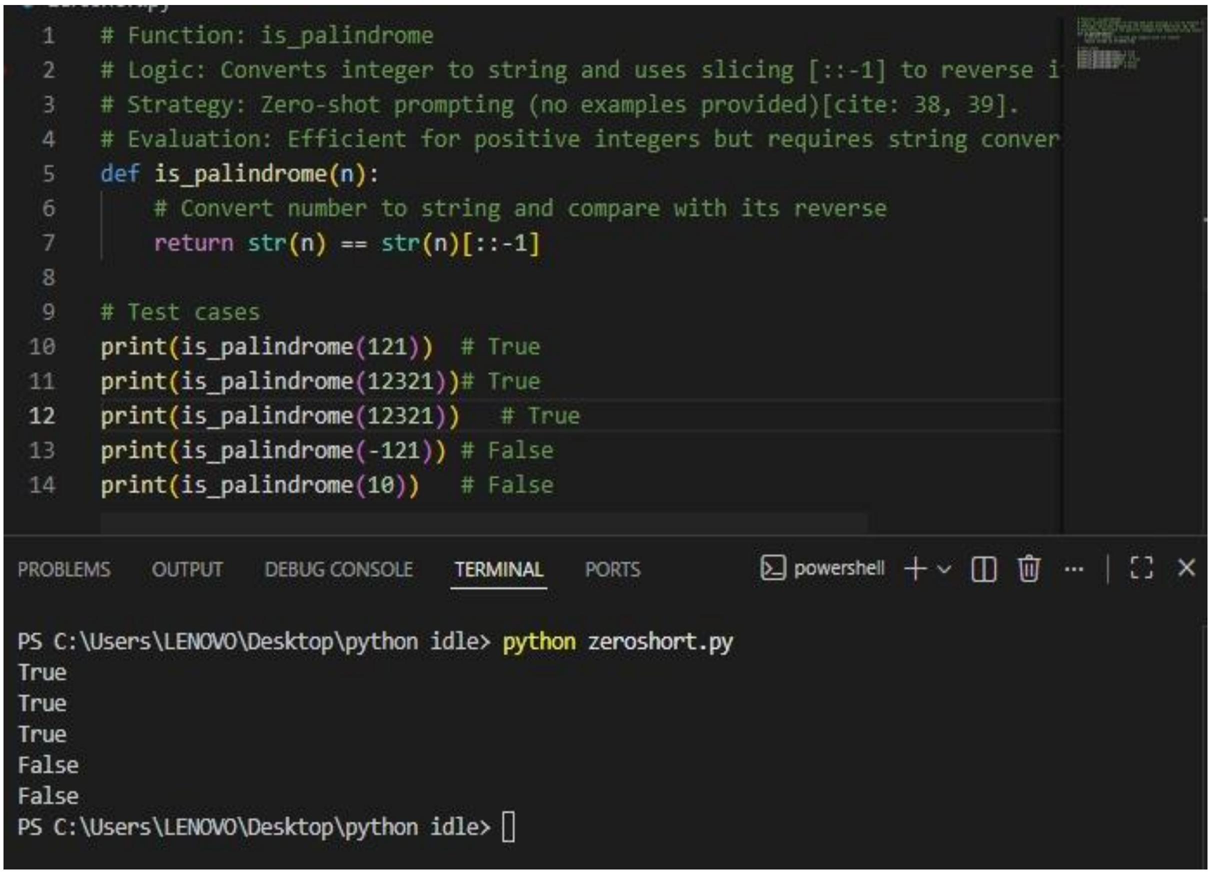
2.one-short (factorial)



```
# Function: factorial
# Logic: Iterative approach using a for-loop to multiply integers from 1 to n.
# Strategy: One-shot prompting using the example 5 -> 120[cite: 44, 46].
# Evaluation: The example helps the AI include base cases (0! and 1!)[cite: 49].
def factorial(n):
    if n < 0:
        return "Factorial is not defined for negative numbers"
    elif n == 0 or n == 1:
        return 1
    else:
        result = 1
        for i in range(2, n + 1):
            result *= i
        return result

# Example provided in prompt
print(factorial(5)) # Output: 120
print(factorial(7))
```

```
120
5040
```

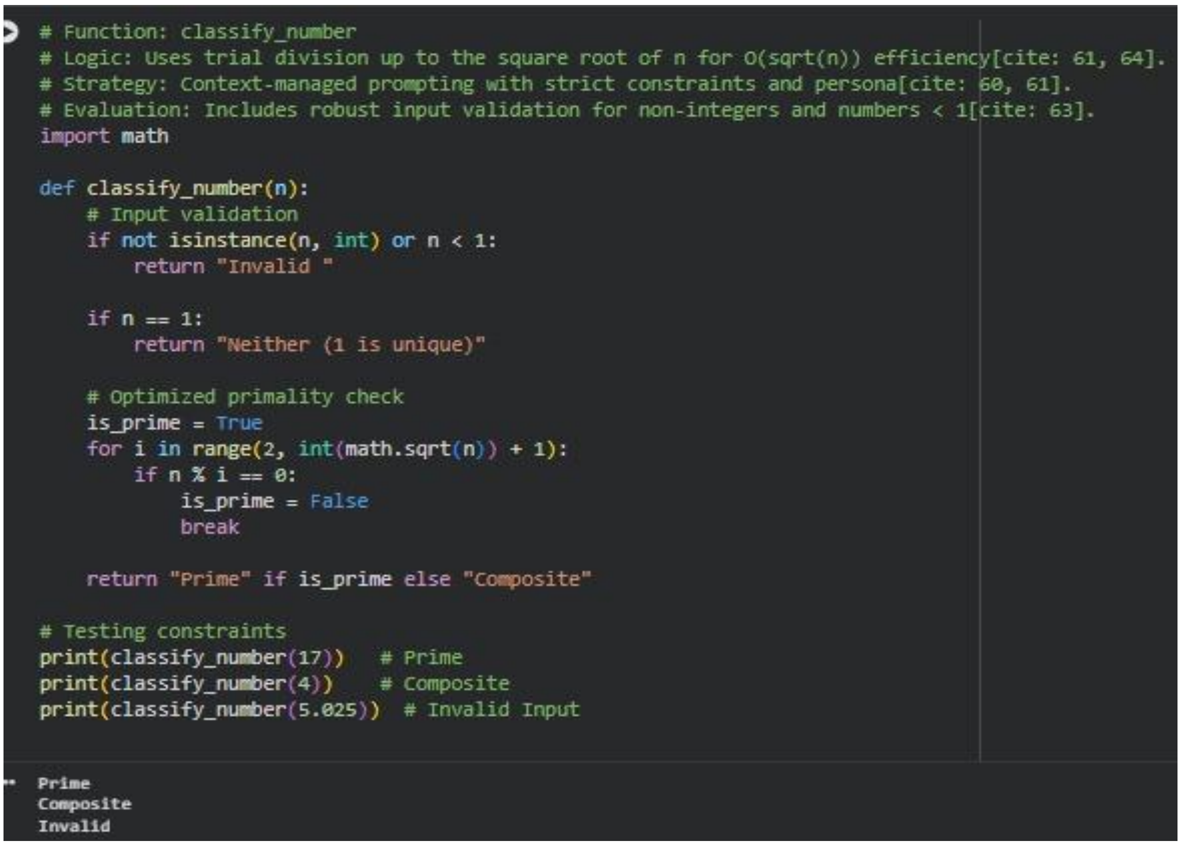
3.few-short (arm num r not)



```
# Function: is_palindrome
# Logic: Converts integer to string and uses slicing [::-1] to reverse i
# Strategy: Zero-shot prompting (no examples provided)[cite: 38, 39].
# Evaluation: Efficient for positive integers but requires string conver
def is_palindrome(n):
    # Convert number to string and compare with its reverse
    return str(n) == str(n)[::-1]

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

```
PS C:\Users\LENOVO\Desktop\python idle> python zeroshort.py
True
True
True
False
False
PS C:\Users\LENOVO\Desktop\python idle>
```

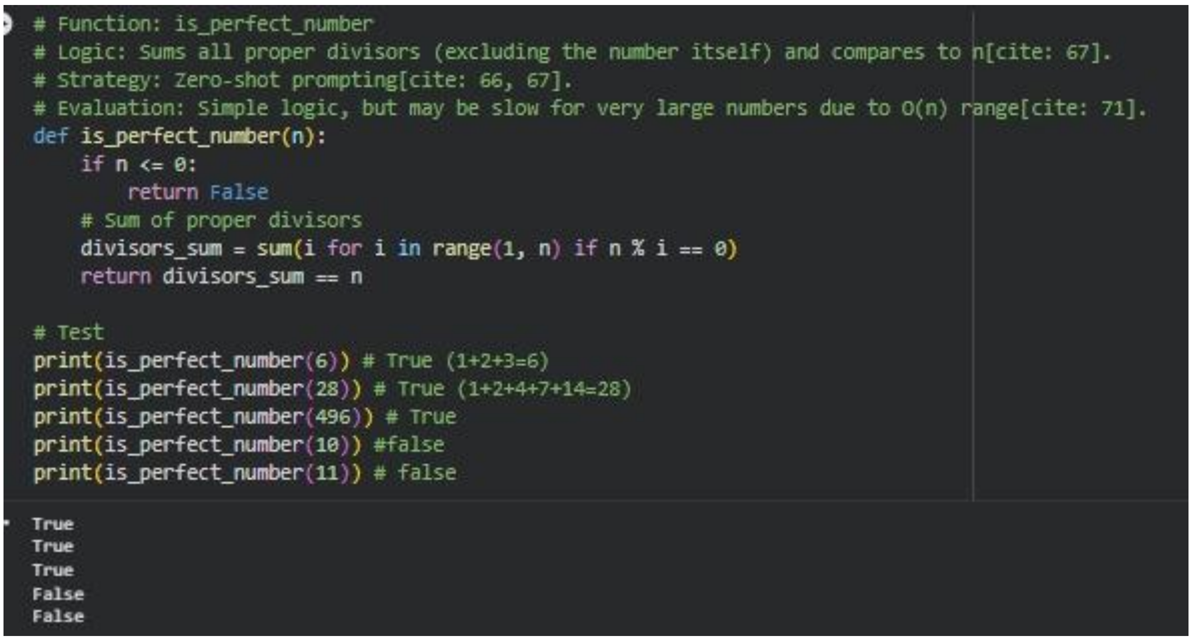
4.nnumber type (prime,compositeor,neither)



```
# Function: classify_number
# Logic: Uses trial division up to the square root of n for O(sqrt(n)) efficiency[cite: 61, 64].
# Strategy: Context-managed prompting with strict constraints and persona[cite: 60, 61].
# Evaluation: Includes robust input validation for non-integers and numbers < 1[cite: 63].
import math

def classify_number(n):
    # Input validation
    if not isinstance(n, int) or n < 1:
        return "Invalid "

    if n == 1:
        return "Neither (1 is unique)"

    # Optimized primality check
    is_prime = True
    for i in range(2, int(math.sqrt(n)) + 1):
        if n % i == 0:
            is_prime = False
            break

    return "Prime" if is_prime else "Composite"

# Testing constraints
print(classify_number(17))   # Prime
print(classify_number(4))    # Composite
print(classify_number(5.025))  # Invalid Input
```

```
Prime
Composite
Invalid
```

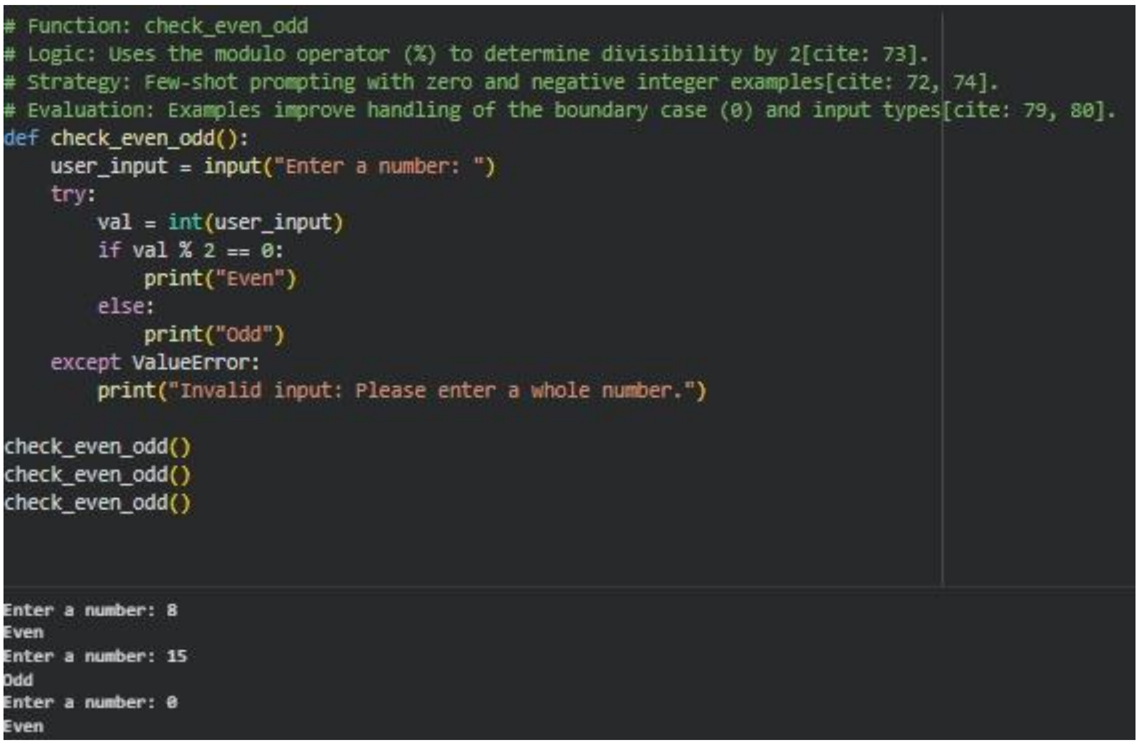
5.Zero-short(prefect number)



```
# Function: is_perfect_number
# Logic: Sums all proper divisors (excluding the number itself) and compares to n[cite: 67].
# Strategy: Zero-shot prompting[cite: 66, 67].
# Evaluation: Simple logic, but may be slow for very large numbers due to O(n) range[cite: 71].
def is_perfect_number(n):
    if n <= 0:
        return False
    # Sum of proper divisors
    divisors_sum = sum(i for i in range(1, n) if n % i == 0)
    return divisors_sum == n

# Test
print(is_perfect_number(6)) # True (1+2+3=6)
print(is_perfect_number(28)) # True (1+2+4+7+14=28)
print(is_perfect_number(496)) # True
print(is_perfect_number(10)) #false
print(is_perfect_number(11)) # false
```

```
True
True
True
False
False
```

6.few-short(even or odd)



```python
# Function: check_even_odd
# Logic: Uses the modulo operator (%) to determine divisibility by 2[cite: 73].
# Strategy: Few-shot prompting with zero and negative integer examples[cite: 72, 74].
# Evaluation: Examples improve handling of the boundary case (0) and input types[cite: 79, 80].
def check_even_odd():
    user_input = input("Enter a number: ")
    try:
        val = int(user_input)
        if val % 2 == 0:
            print("Even")
        else:
            print("Odd")
    except ValueError:
        print("Invalid input: Please enter a whole number.")

check_even_odd()
check_even_odd()
check_even_odd()
```

```
Enter a number: 8
Even
Enter a number: 15
Odd
Enter a number: 0
Even
```