

BATCH 29

N. BLESSY

2303A51062

## ASSIGNMENT 3.1

```
▶ def is_palindrome(num):
    original = num
    rev = 0
    while temp > 0:
        digit = temp % 10
        rev = rev * 10 + digit
        temp //= 10
    return num == rev

print(is_palindrome(121))
print(is_palindrome(123))
print(is_palindrome(0))
print(is_palindrome(1221))
print(is_palindrome(-121))
```

---

```
... True
False
True
True
False
```

### Analysis & Justification

- ✓ Correctly checks palindrome for positive integers
- ✓ Efficient logic using digit reversal
- ✗ Negative numbers are not handled explicitly
- ✗ Does not validate non-integer input
- ✓ Zero-shot prompting produced a simple but incomplete solution

Conclusion:

Zero-shot prompting works for basic cases but often misses edge-case handling.

```
[2] 0s  def factorial(n):
        if n < 0:
            return "Factorial not defined for negative numbers"

        result = 1
        for i in range(1, n + 1):
            result *= i
        return result
print(factorial(5))
print(factorial(0))
print(factorial(1))
print(factorial(-3))

...
120
1
1
Factorial not defined for negative numbers
```

### Analysis & Justification

- ✓ One example guided correct behavior
- ✓ Improved **validation and clarity**
- ✓ Handles  $0!$  correctly
- ✓ One-shot prompting leads to **more robust solutions**

```
▶ def is_armstrong(num):
    if num < 0:
        return "Invalid input"

    temp = num
    digits = len(str(num))
    total = 0

    while temp > 0:
        digit = temp % 10
        total += digit ** digits
        temp //= 10

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

print(is_armstrong(153))
print(is_armstrong(370))
print(is_armstrong(123))
print(is_armstrong(0))
print(is_armstrong(-10))
```

```
... Armstrong Number
Armstrong Number
Not an Armstrong Number
Armstrong Number
Invalid input
```

## Analysis & Justification

- ✓ Examples influenced **output format**
- ✓ Correct power calculation using digit count
- ✓ Handles negative numbers
- △ For 0, logic works but not explicitly mentioned
- ✓ Few-shot prompting improves **accuracy and structure**

```

def classify_number(n):
    if not isinstance(n, int):
        return "Invalid input"

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

    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            return "Composite"

    return "Prime"
print(classify_number(2))      # Prime
print(classify_number(9))      # Composite
print(classify_number(1))      # Neither
print(classify_number(-5))     # Neither
print(classify_number(7.5))    # Invalid input

```

```

Prime
Composite
Neither Prime nor Composite
Neither Prime nor Composite
Invalid input

```

## Analysis & Justification

- ✓ Optimized using  $\sqrt{n}$  approach
- ✓ Clear constraints improved logic quality
- ✓ Best performance among all strategies



```

def is_perfect(num):
    if num <= 0:
        return False

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

    return total == num
print(is_perfect(6))      # True
print(is_perfect(28))      # True
print(is_perfect(12))      # False
print(is_perfect(1))       # False

```

```

...  True
      True
      False
      False

```

## Analysis & Justification

- ✓ Correct logic
- ✗ Inefficient loop till num
- ✗ No optimization ( $\sqrt{n}$ )
- ✓ Zero-shot again gives **working but inefficient** solution

---

```
▶ def even_or_odd(n):
    if not isinstance(n, int):
        return "Invalid input"

    if n % 2 == 0:
        return "Even"
    else:
        return "Odd"
print(even_or_odd(8))
print(even_or_odd(15))
print(even_or_odd(0))
print(even_or_odd(-4))
print(even_or_odd(3.5))
```

---

... Even  
Odd  
Even  
Even  
Invalid input

---

## Analysis & Justification

- ✓ Handles negative numbers correctly
- ✓ Clear output formatting
- ✓ Input validation added due to examples
- ✓ Few-shot prompting improves **robustness and clarity**