AI Assisted Coding (III Year) Assignment

NAME : K.AKSHAY

HT NO :2303A52209

BATCH :35

Lab 2: Prompt Engineering – Improving Prompts and Context Management

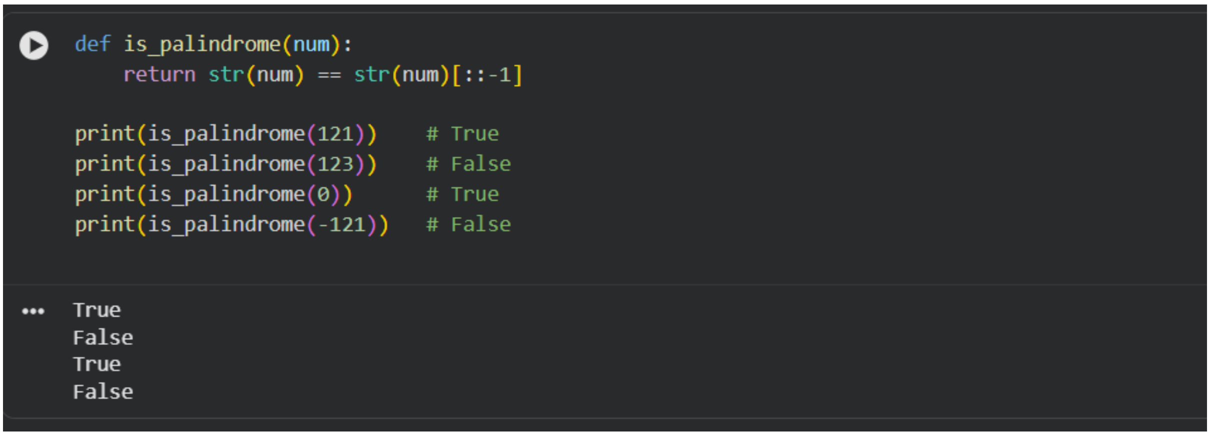
Lab Objectives

1. ***To understand and apply different prompt engineering techniques for generating Python programs using AI-assisted tools.***
2. ***To analyze how context and examples affect correctness and efficiency of AI-generated code.***
3. ***To refine real-world Python programs through iterative prompt improvement.***

Task 1 Zero-Shot Prompting – Palindrome Number Program

Prompt: Write a Python function to check whether a given number is a palindrome

Code :



```
def is_palindrome(num):
    return str(num) == str(num)[::-1]

print(is_palindrome(121))    # True
print(is_palindrome(123))    # False
print(is_palindrome(0))      # True
print(is_palindrome(-121))   # False
```

```
True
False
True
False
```

Observations

- ***Works correctly for positive integers and zero***
- ***Negative numbers are not handled explicitly (logical ambiguity)***
- ***Relies on string conversion instead of numeric logic***

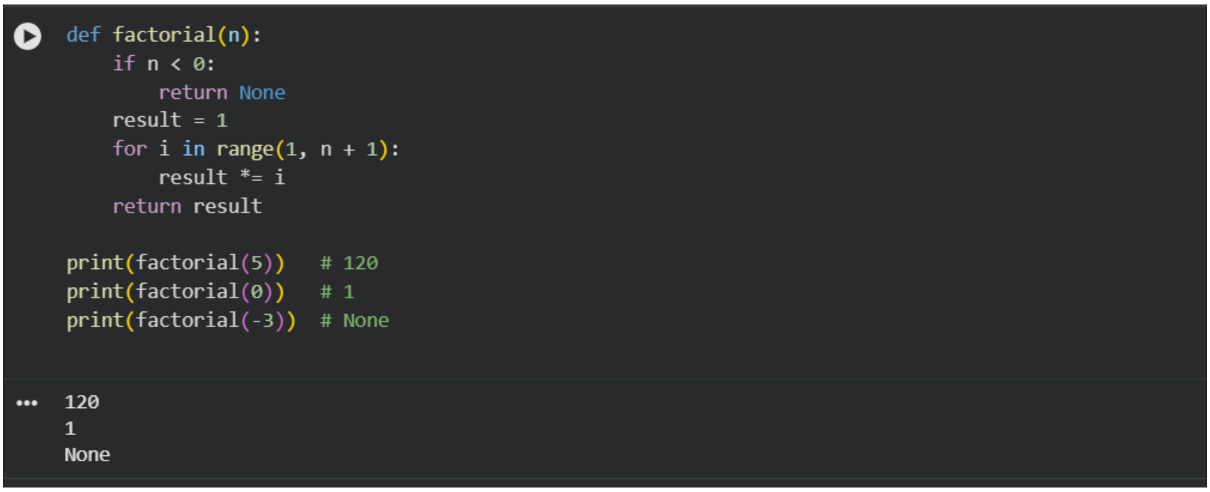
Conclusion

Zero-shot prompting produces concise code but often ignores edge cases unless explicitly instructed.

Task 2: One-Shot Prompting – Factorial Calculation

Prompt: Write a Python function to calculate the factorial of a number. Example: Input: 5 → Output: 120

Code 1:



```
def factorial(n):
    if n < 0:
        return None
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result

print(factorial(5))    # 120
print(factorial(0))    # 1
print(factorial(-3))   # None
```

```
120
1
None
```

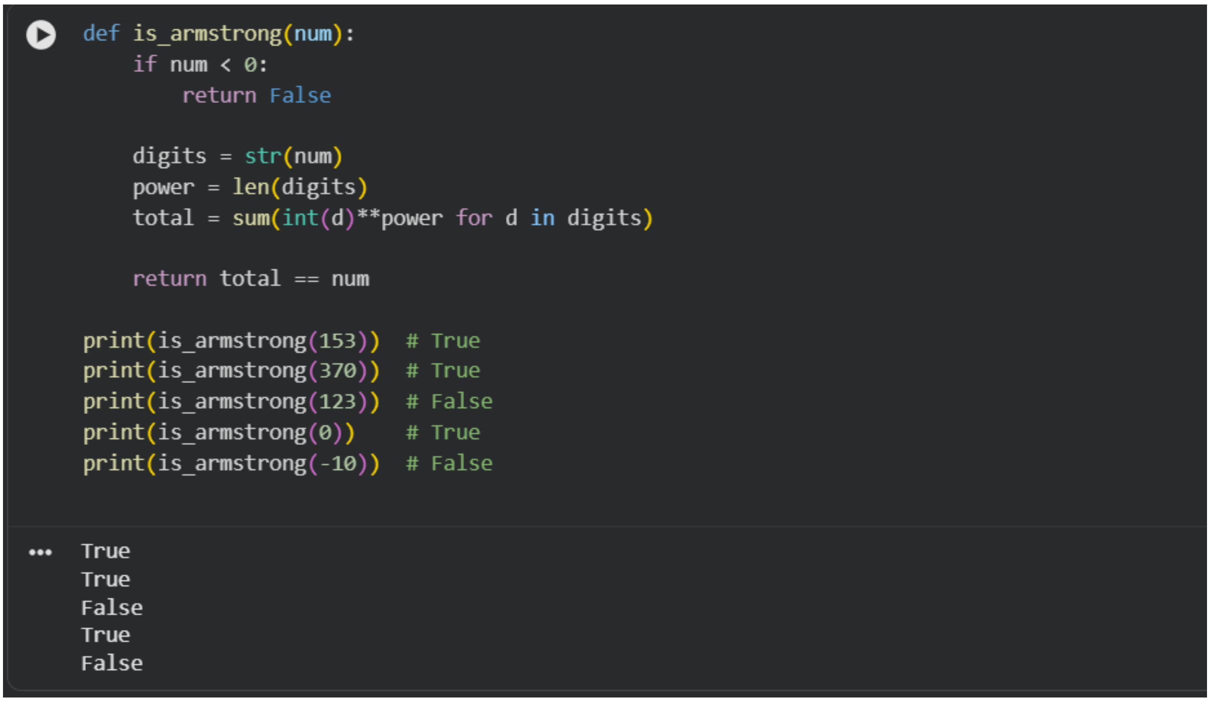
Conclusion

Providing even a single example improves correctness and defensive logic.

TASK-3 Few-Shot Prompting – Armstrong Number Check

***Prompt: "Write a Python function to check Armstrong numbers.
Examples:***
Input: 153 → Armstrong Number
Input: 370 → Armstrong Number
Input: 123 → Not an Armstrong Number"

Code:



```python
def is_armstrong(num):
    if num < 0:
        return False

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

    return total == num

print(is_armstrong(153))  # True
print(is_armstrong(370))  # True
print(is_armstrong(123))  # False
print(is_armstrong(0))    # True
print(is_armstrong(-10))  # False
```

```
True
True
False
True
False
```

Analysis

- ***Multiple examples guided correct logic structure***
- ***Handles zero and negative values properly***
- ***Code is cleaner and more reliable than zero-shot***

Conclusion

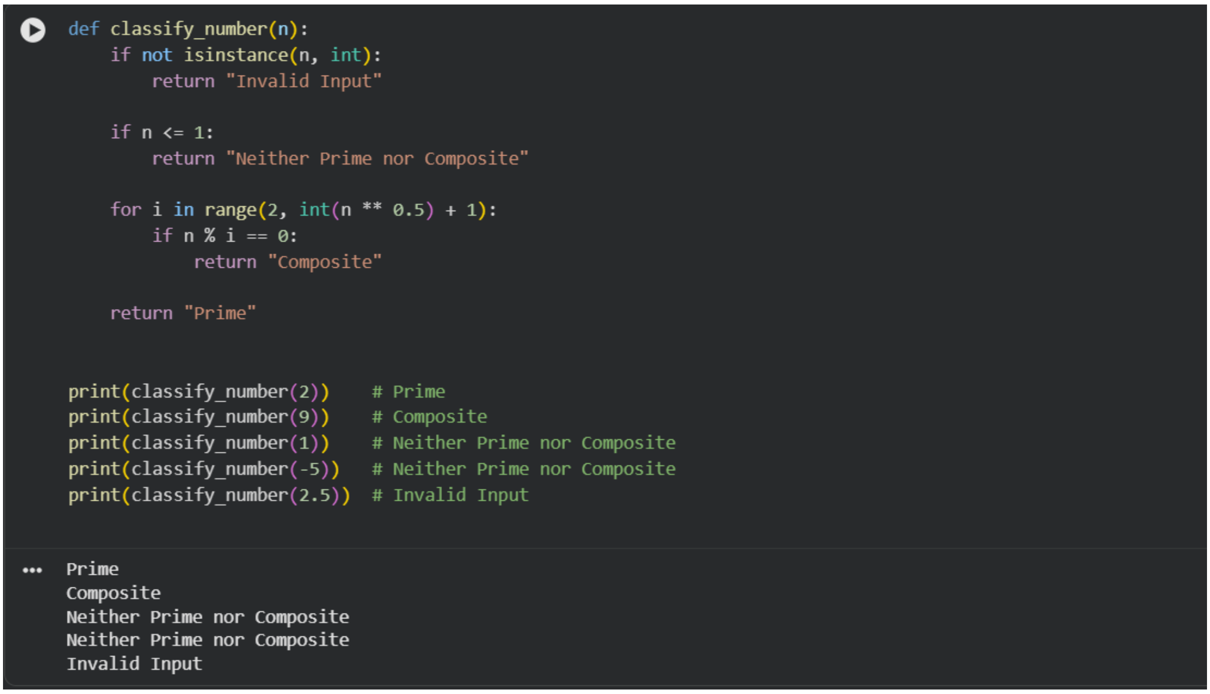
Few-shot prompting significantly improves accuracy and robustness.

TASK -4 Context-Managed Prompting – Optimized Number Classification

Prompt: "Generate an optimized Python program to classify a number as prime, composite, or neither.
Constraints:

- ***Handle invalid inputs***
- ***Use efficient logic***
- ***Do not use unnecessary loops"***

Code:



```python
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 Prime nor Composite
print(classify_number(-5))   # Neither Prime nor Composite
print(classify_number(2.5))  # Invalid Input
```

```
Prime
Composite
Neither Prime nor Composite
Neither Prime nor Composite
Invalid Input
```

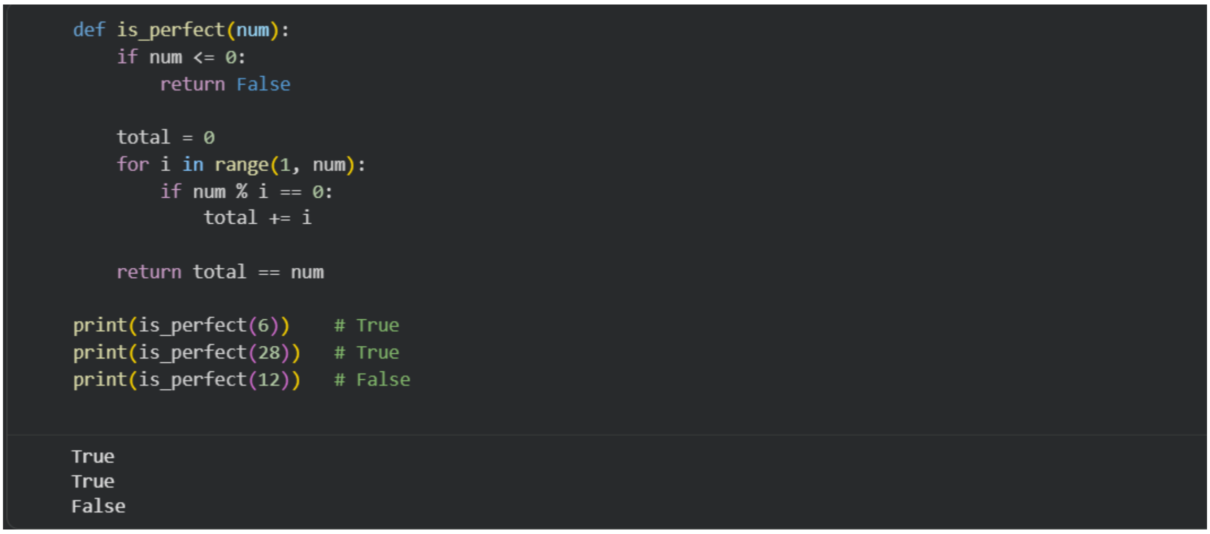
Conclusion

Context-managed prompts produce the most production-ready code with clear constraints and validation.

TASK -5 Zero-Shot Prompting – Perfect Number Check

Prompt: “Write a Python function to check whether a number is a perfect number.”

Code:



```
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
```

```
True
True
False
```

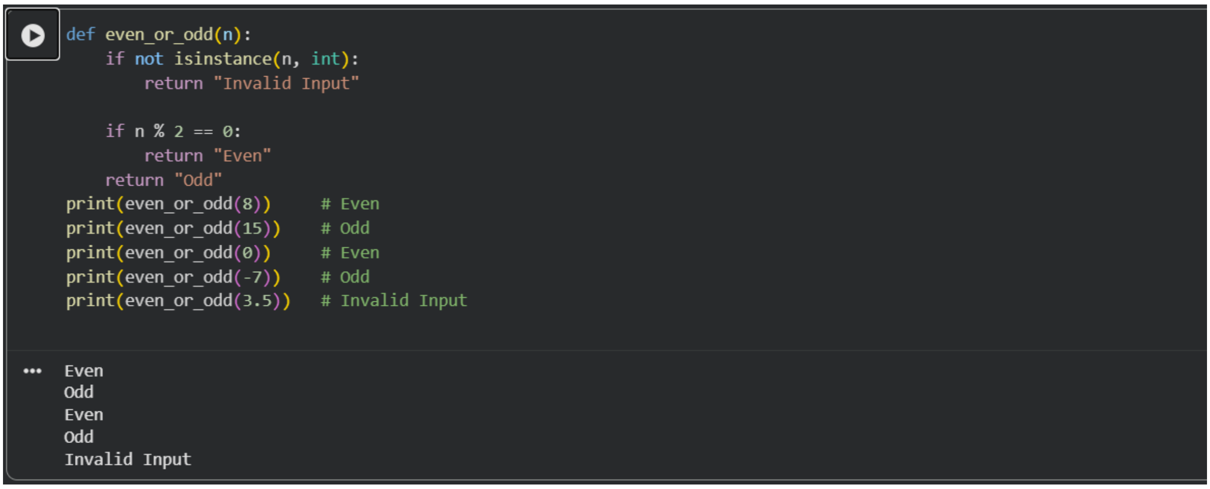
Issues Identified

- ***Inefficient loop runs up to num-1***
- ***No optimization using square root***
- ***Still logically correct, just slow***

TASK -6 Few-Shot Prompting – Even or Odd Classification with Validation.

Prompt: "Write a Python program to check even or odd with validation.
Examples:
Input: 8 → Even
Input: 15 → Odd
Input: 0 → Even"

Code:



```python
def even_or_odd(n):
    if not isinstance(n, int):
        return "Invalid Input"

    if n % 2 == 0:
        return "Even"
    return "Odd"
print(even_or_odd(8))     # Even
print(even_or_odd(15))    # Odd
print(even_or_odd(0))     # Even
print(even_or_odd(-7))    # Odd
print(even_or_odd(3.5))   # Invalid Input
```

```
Even
Odd
Even
Odd
Invalid Input
```

Analysis

- ***Examples improved validation handling***
- ***Output clarity improved***
- ***Works for negative integers and rejects non-integers***

Final Conclusion

Prompt engineering directly affects code correctness, efficiency, and robustness. Zero-shot prompting is suitable only for trivial tasks. Few-shot and context-managed prompting consistently produce better-structured, validated, and optimized Python programs. Clear instructions and examples significantly improve AI-assisted code generation.