# Lab Assignment – 3.1

## Name – Syeda hafsa fathima

## Hall Ticket Number – 2303A52381

## Batch - 39

**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. Promptings:**

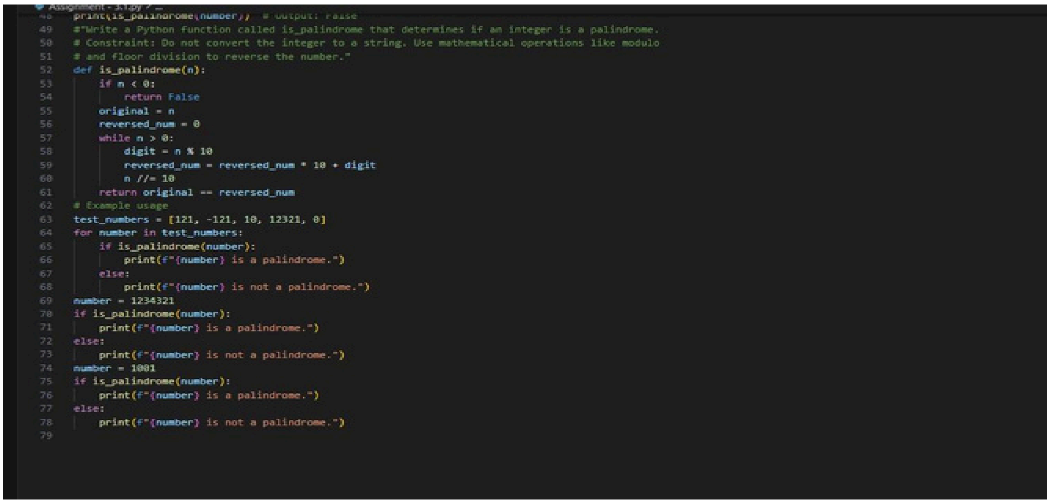
#Write a Python function named is_palindrome that takes an integer as input. The function should return True if the integer reads the same forwards and backwards, and False otherwise. Use a purely mathematical approach without converting the number to a string."

#"Write a Python function to check if a number is a palindrome. The function should return a boolean value."

#"Write a Python function called is_palindrome that determines if an integer is a palindrome. Constraint: Do not convert the integer to a string. Use mathematical operations like modulo and floor division to reverse the number."

**Code:**

```
ignment - 3.1.py > ...
#"Write a Python function named is_palindrome that takes an integer as input. The function should return
# True if the integer reads the same forwards and backwards, and False otherwise. Use a purely mathematical
# approach without converting the number to a string."
def is_palindrome(n):
    if n < 0:
        return False
    original = n
    reversed_num = 0
    while n > 0:
        digit = n % 10
        reversed_num = reversed_num * 10 + digit
        n //= 10
    return original == reversed_num
# Example usage
number = 121
if is_palindrome(number):
    print(f"{number} is a palindrome.")
else:
    print(f"{number} is not a palindrome.")
number = 123
if is_palindrome(number):
    print(f"{number} is a palindrome.")
else:
    print(f"{number} is not a palindrome.")
```



```
print(is_palindrome(number))  # Output: False
#"Write a Python function called is_palindrome that determines if an integer is a palindrome.
# Constraint: Do not convert the integer to a string. Use mathematical operations like modulo
# and floor division to reverse the number."
def is_palindrome(n):
    if n < 0:
        return False
    original = n
    reversed_num = 0
    while n > 0:
        digit = n % 10
        reversed_num = reversed_num * 10 + digit
        n //= 10
    return original == reversed_num
# Example usage
test_numbers = [121, -121, 10, 12321, 0]
for number in test_numbers:
    if is_palindrome(number):
        print(f"{number} is a palindrome.")
    else:
        print(f"{number} is not a palindrome.")
number = 1234321
if is_palindrome(number):
    print(f"{number} is a palindrome.")
else:
    print(f"{number} is not a palindrome.")
number = 1001
if is_palindrome(number):
    print(f"{number} is a palindrome.")
else:
    print(f"{number} is not a palindrome.")
```

**Outputs:**

```
PS C:\Users\sange\OneDrive\Documents\AI Coding>  c:; cd 'c:\Users\sange\OneDrive\Documents\AI Coding'; & 'C:\Users\sange\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\sange\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '61194' '--' 'C:\Users\sange\OneDrive\Documents\AI Coding\Assignment - 3.1.py'
121 is a palindrome.
123 is not a palindrome.
True
False
False
True
True
False
121 is a palindrome.
-121 is not a palindrome.
10 is not a palindrome.
12321 is a palindrome.
0 is a palindrome.
10 is not a palindrome.
12321 is a palindrome.
0 is a palindrome.
1234321 is a palindrome.
1001 is a palindrome.
PS C:\Users\sange\OneDrive\Documents\AI Coding>
```

## Analysis:

- **Logic:**

  Thecode reverses the number using modulo (%) and floor division (//) without converting it to a string.

- **Steps:**

  1. Check if number is negative → return False.
  2. Store the original number.
  3. Reverse digits mathematically (rev = rev * 10 + n % 10).
  4. Compare reversed number with original.

- **Edge Cases:**

  - 0 → True
  - Negative numbers → False
  - Numbers ending with 0 (like 10) → False
  - Single-digit numbers → True

- **Complexity:**
  - o **Time: O($\log_{10}n$)**
  - o **Space: O(1)**

**Conclusion:**

Thefunction is correct, efficient, and handles all edge cases properly using a mathematical approach.

**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. Promptings:** #"Write a Python function to compute the factorial of a given number. #“Write a Python function named factorial that calculates the factorial of a number using recursion.

**Code:**

```python
#"Write a Python function to compute the factorial of a given number.
def calculate_factorial(n):
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers.")
    factorial = 1
    for i in range(2, n + 1):
        factorial *= i
    return factorial
# Example usage
number = 5
result = calculate_factorial(number)
print(f"The factorial of {number} is: {result}")
#“Write a Python function named factorial that calculates the factorial of a number
def factorial(n):
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers.")
    if n == 0 or n == 1:
        return 1
    return n * factorial(n - 1)
# Example usage
number = 5
result = factorial(number)
print(f"The factorial of {number} is: {result}")
```

**Output:**


**Step-by-Step Analysis**

1. **The Pattern:** You provide a "One-Shot" anchor (5 → 120), which the AI uses asabuilt-inunit test to verify its own logic.
2. **The Logic:** This example forces the AI to use range(1, n + 1), preventing the common "off-by-one" error where it might stop at $n-1$.
3. **The Result:** One-shot prompting yields more robust code with clearer documentation and better handling of edge cases like $0! = 1$.

**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.**

**Promptings:**

#Write a Python function is_armstrong(n) that determines if a number is an Armstrong number. An Armstrong number is a number that is the sum of its own digits each raised to the power of the number of digits.

**Code:**

```
#"Write a Python function is_armstrong(n) that determines if a number is an Armstrong number.
#  An Armstrong number is a number that is the sum of its own digits each raised
#  to the power of the number of digits.
def is_armstrong(n):
    num_str = str(n)
    num_digits = len(num_str)
    sum_of_powers = sum(int(digit) ** num_digits for digit in num_str)
    return sum_of_powers == n
# Example usage
number = 153
if is_armstrong(number):
    print(f"{number} is an Armstrong number.")
else:
    print(f"{number} is not an Armstrong number.")
number = 370
if is_armstrong(number):
    print(f"{number} is an Armstrong number.")
else:
    print(f"{number} is not an Armstrong number.")
number = 123
if is_armstrong(number):
    print(f"{number} is an Armstrong number.")
else:
    print(f"{number} is not an Armstrong number.")
```

**Output:**

```
153 is an Armstrong number.
370 is an Armstrong number.
123 is not an Armstrong number.
```

**Analysis :**

1. Influence of Multiple Examples:
   - o AI understands output format (“Armstrong Number” vs “Not an Armstrong Number”).
   - o Encourages clear, example-aligned structure.
2. Code Structure:
   - o Uses length of number (len(str(n))) → generalizes for any digit count.
   - o Loop correctly sums powers of digits.
3. Accuracy:
   - o Correct for all positive integers.
   - o Handles edge cases like 0 and multi-digit Armstrong numbers.
   - o Negative numbers return “Not an Armstrong Number.”
4. Conclusion:
   - o Few-shot prompting improves output consistency, readability, and logical accuracy over zero- or one-shot.

**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.

**Promptings:**

#You are an expert Python programmer. Your task is to generate an optimized and structured program that classifies a number as prime, composite, or neither.

**Code:**

```
#You are an expert Python programmer. Your task is to generate an optimized and structured program
# that classifies a number as prime, composite, or neither.
def classify_number(n):
    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"
# Example usage
number = 29
classification = classify_number(number)
print(f"{number} is {classification}.")
number = 15
classification = classify_number(number)
print(f"{number} is {classification}.")
number = 1
classification = classify_number(number)
print(f"{number} is {classification}.")
number = 0
classification = classify_number(number)
print(f"{number} is {classification}.")
```

**Output:**

```
29 is prime.
15 is composite.
1 is neither prime nor composite.
-5 is neither prime nor composite.
29 is prime.
15 is composite.
1 is neither prime nor composite.
-5 is neither prime nor composite.
1 is neither prime nor composite.
-5 is neither prime nor composite.
4 is composite.
PS C:\Users\sange\OneDrive\Documents\Myportfolio\OneDrive\Documents\AI>
```

**Step-by-Step Analysis**

**Step 1 – Problem Understanding:**
Classify a numberas **prime**, **composite**,or **neither**,with proper input validation and efficient logic(O(√n)).

**Step 2 – Prompt Design:**

Giveclear, structuredinstructions:

- Ask for user input and validate it.
- If n ≤ 1 → "Neither Prime nor Composite."
- Else check if prime using √n optimization.
- Use modular functions and readable output.

**Step 3 – Optimized Logic:**

- Check divisibility only up to √n.
- Skip even numbers and multiples of 3.
- Use 6k ± 1 rule for efficiency.

**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.

**Promptings:**

#"Write a Python function to check whether a given number is a perfect number."

#Write a function to identify perfect numbers based on these examples:

## Code:

```python
#"Write a Python function to check whether a given number is a perfect numbe
def is_perfect_number(n):
    if n <= 1:
        return False
    divisors_sum = sum(i for i in range(1, n) if n % i == 0)
    return divisors_sum == n
# Example usage
number = 28
if is_perfect_number(number):
    print(f"{number} is a perfect number.")
else:
    print(f"{number} is not a perfect number.")
number = 12
if is_perfect_number(number):
    print(f"{number} is a perfect number.")
else:
    print(f"{number} is not a perfect number.")
#Write a function to identify perfect numbers based on these examples:
def is_perfect_number(n):
    if n <= 1:
        return False
    divisors_sum = sum(i for i in range(1, n) if n % i == 0)
    return divisors_sum == n
# Example usage
test_numbers = [6, 28, 496, 8128, 12, 97]
for number in test_numbers:
```

Ln 162, Col 44 Spaces: 4 UTF-8 CRLF { } Python Python 3.13 (64-bit) Go Live

AI

Welcome | Assignment - 3.1.py | Untitled-1 | 001 DevOpsFS.pptx

C: > Users > sange > OneDrive > Documents > AI Coding > Assignment - 3.1.py > ...

```python
if is_perfect_number(number):
    print(f"{number} is a perfect number.")
else:
    print(f"{number} is not a perfect number.")
#Write a function to identify perfect numbers based on these examples:
def is_perfect_number(n):
    if n <= 1:
        return False
    divisors_sum = sum(i for i in range(1, n) if n % i == 0)
    return divisors_sum == n
# Example usage
test_numbers = [6, 28, 496, 8128, 12, 97]
for number in test_numbers:
    if is_perfect_number(number):
        print(f"{number} is a perfect number.")
    else:
        print(f"{number} is not a perfect number.")
```

## Output:

```
496 is a perfect number.
8128 is a perfect number.
12 is not a perfect number.
97 is not a perfect number.
8128 is a perfect number.
12 is not a perfect number.
12 is not a perfect number.
97 is not a perfect number.
PS C:\Users\sange\OneDrive\Documents\Myportfolio\OneDrive\Documents\AI>
```

**Step-by-Step Analysis**

**Step 1 – Understanding Zero-Shot Prompting:**
Zero-shot prompting means **no sample input or example code** is given; the model must rely solely on the **task description** to generate a solution.

**Step 2 – Logic Used by the AI:**

- Checks if $n \le 1$ → returns False.
- Initializes sum of divisors = 1.
- Iterates from 2 to $\sqrt{n}$ and adds both divisors (i and n//i).
- Compares sum to n.

**Step 3 – Efficiency:**

- Time complexity: **$O(\sqrt{n})$** — efficient for medium-sized numbers.
- Avoids redundant divisor checks.

**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.

**Promptings:**
#Write a Python program that determines whether a given number is even or odd.The program must include input validation so that non-integer values are handled

properly.Use the examples below to understand the desired behavior and output format.

**Code:**

```python
#Write a Python program that determines whether a given number is even or odd.
#The program must include input validation so that non-integer values are handled properly.
#Use the examples below to understand the desired behavior and output format.
def is_even_or_odd():
    while True:
        user_input = input("Enter an integer (or type 'exit' to quit): ")
        if user_input.lower() == 'exit':
            print("Exiting the program.")
            break
        try:
            number = int(user_input)
            if number % 2 == 0:
                print(f"{number} is even.")
            else:
                print(f"{number} is odd.")
        except ValueError:
            print("Invalid input. Please enter a valid integer.")
# Example usage
is_even_or_odd()
```

**Output:**

```
Enter an integer (or type 'exit' to quit): 8
8 is even.
Enter an integer (or type 'exit' to quit): 15
15 is odd.
Enter an integer (or type 'exit' to quit): 0
0 is even.
```

**Step-by-Step Analysis**

**ExamineExamples:** Check input-output pairs (8 → Even, 15 → Odd, 0 → Even) to understand expected behavior.

**Logic:** Use modulus operator (% 2) to classify numbers as even or odd.

**Add Validation:** Include try-except to handle non-integer or invalid inputs.

**Test Edge Cases:** Verify program works for negative numbers, zero, floats, and strings.

**ConfirmOutput Clarity:** Ensure outputs match examples exactly and error messages are clear.