

# ASSIGNMENT – 3.1

RAVULA AJAY

2303A52370

BATCH- 44

## 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:

checking whether given number it is palindrome or not

#### CODE:

The screenshot shows a code editor interface with the following details:

- File Explorer:** Shows files like "AI-ASS.py", "1.4-AI-ASS.py", and "1.4-AI-ASSSpy".
- Code Editor:** Displays the generated Python code for a palindrome checker. The code handles negative numbers, converts integers to strings, and checks if the string is equal to its reverse. It includes a test case section with various numbers and their expected outcomes.

```
1 def is_palindrome(n):
2     # Handle negative numbers - they are not palindromes
3     if n < 0:
4         return False
5
6     # Convert number to string and compare with its reverse
7     num_str = str(n)
8     return num_str == num_str[::-1]
9
10
11 # Test cases
12 if __name__ == "__main__":
13     test_cases = [
14         121,      # True - palindrome
15         -121,     # False - negative number
16         0,        # True - single digit
17         10,       # False - not a palindrome
18         1001,     # True - palindrome
19         12321,   # True - palindrome
20         123,     # False - not a palindrome
21         9,       # True - single digit
22         100,     # False - not a palindrome
23     ]
24
25     for num in test_cases:
26         result = is_palindrome(num)
27         print(f"Is {num} a palindrome? {result}")
```

- Output Panel:** Shows the results of running the code with the provided test cases.
- Right Sidebar:** Contains sections for "RECENT SESSIONS" (with three entries), "Ask about your code" (with a button to "Ask"), and "Explore and understand" (with a "Go Live" button).

The screenshot shows a code editor interface with the following details:

- File Explorer:** Shows files like `AI-Ass.py`, `AI-Ass.pyx`, `1.4-AI-Ass.py`, and `AI-Ass.py`.
- Code Editor:** Displays Python code for checking if a number is a palindrome. The code includes a test function `is_palindrome` and a main loop for testing various numbers.
- Terminal:** Shows the execution of the script and its output, confirming the correctness of the palindrome check for various inputs.
- Bottom Status Bar:** Includes icons for search, file operations, and system status.

## 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:** Factorial Calculation giving some instructions input data

## CODE:

```

File Edit Selection View Go Run Terminal Help < > AI Assisted Code
EXPLORER ... AI-ASSISTED CODE 1.4-AI-ASS.py 1.4-AI-ASS.py
given number.
Example:
Input 5 -> Output: 120
task:
Add Context...
def factorial(n):
    """
    Calculate the factorial of a number.
    One-shot generated function based on the example: 5 -> 120
    """
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers")
    if n == 0 or n == 1:
        return 1
    result = 1
    for i in range(2, n + 1):
        result *= i
    return result

# -----#
# ZERO-SHOT SOLUTION (no example)
# -----
# Generate a Python function to compute the factorial
# -----
def factorial_zero_shot():
    """
    Zero-shot generated function (without example guidance)
    """
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers")
    if n == 0 or n == 1:
        return 1
    return n * factorial_zero_shot(n - 1)

# -----#
# TEST & COMPARISON
# -----
if __name__ == "__main__":
    test_value = 5

    print(f"One-Shot Result: {factorial(test_value)}") # Output: 120
    print(f"Zero-Shot Result: {factorial_zero_shot(test_value)}") # Output: 120
    print("\n--- Comparison ---")

```

RECENT SESSIONS

- Code optimization and... Completed Local - 3 days
- Code optimization for... Completed Local - 3 days
- Code optimization for... Completed Local - 3 days

Ask about your code

AI responses may be inaccurate.

Generate Agent Instructions to onboard AI onto your codebase.

Ln 28, Col 1 Spaces: 4 UTF-8 CRLF Python 3.13.7 13:47 12-01-2026

  

```

File Edit Selection View Go Run Terminal Help < > AI Assisted Code
EXPLORER ... AI-ASS.py 1.4-AI-ASS.py
AI-ASS.py > factorial
1 def factorial(n):
2     """
3         Calculate the factorial of a number.
4         One-shot generated function based on the example: 5 -> 120
5     """
6     if n < 0:
7         raise ValueError("Factorial is not defined for negative numbers")
8     if n == 0 or n == 1:
9         return 1
10    result = 1
11    for i in range(2, n + 1):
12        result *= i
13    return result
14
15
16 # -----#
17 # ZERO-SHOT SOLUTION (no example)
18 # -----
19 # Generate a Python function to compute the factorial
20 # -----
21 def factorial_zero_shot():
22     """
23         Zero-shot generated function (without example guidance)
24     """
25     if n < 0:
26         raise ValueError("Factorial is not defined for negative numbers")
27     if n == 0 or n == 1:
28         return 1
29     return n * factorial_zero_shot(n - 1)
30
31 # -----#
32 # TEST & COMPARISON
33 # -----
34 if __name__ == "__main__":
35     test_value = 5
36
37     print(f"One-Shot Result: {factorial(test_value)}") # Output: 120
38     print(f"Zero-Shot Result: {factorial_zero_shot(test_value)}") # Output: 120
39     print("\n--- Comparison ---")

```

RECENT SESSIONS

- Code optimization and... Completed Local - 3 days
- Code optimization for... Completed Local - 3 days
- Code optimization for... Completed Local - 3 days

Ask about your code

AI responses may be inaccurate.

Generate Agent Instructions to onboard AI onto your codebase.

Ln 28, Col 25 Spaces: 4 UTF-8 CRLF Python 3.13.7 13:47 12-01-2026

### 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: Armstrong Number Check

- Input: 153 → Output: Armstrong Number

- Input: 370 → Output: Armstrong Number
  - Input: 123 → Output: Not an Armstrong Number

## CODE:

The screenshot displays two instances of Microsoft Visual Studio Code running on a Windows operating system. Both windows have the title bar "Q AI Assisted Code".

**Left Window (Top):**

- Explorer:** Shows three files: "AI-Ass.py", "1.4-AI-Ass.py", and "AI-Ass.py".
- Code Editor:** Displays Python code for an Armstrong number checker. The code defines a function `is\_armstrong\_number` that checks if a given number is an Armstrong number by comparing it with the sum of its own digits each raised to the power of the number of digits. It includes test cases and prints "Armstrong Number Checker - Test Results".
- Output:** Shows the command-line interface for running the code.
- Status Bar:** Shows the current file is "AI-Ass.py", line 41, column 50, and the date is 12-01-2026.

**Right Window (Bottom):**

- Recent Sessions:** Lists completed code optimizations.
- Code Optimization Panel:** A sidebar with a "Ask about your code" button, an AI response placeholder ("AI responses may be inaccurate"), and a "Generate Agent" button.
- Output:** Shows the command-line output of the Python code execution, including test cases and results.
- Status Bar:** Shows the current file is "AI-Ass.py", line 41, column 50, and the date is 12-01-2026.

#### **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: Optimized Number Classification CODE:

The image shows two side-by-side instances of Microsoft Visual Studio Code (VS Code) running on a Windows desktop. Both instances have dark themes and are displaying the same Python script named `AI-Ass.py`. The script contains a function `classify_number(num)` that classifies integers as prime, composite, or neither. It includes input validation, handles edge cases like negative numbers and zero, and uses an optimized prime check by only checking odd divisors up to the square root of the number. A main function tests the classifier with a list of numbers. The bottom window shows the terminal output, which displays the classification results for each number in the test case list.

```
def classify_number(num):
    """
    Classifies a number as prime, composite, or neither.

    Args:
        num: Integer to classify

    Returns:
        str: Classification result
    """
    # Input validation
    try:
        num = int(num)
    except (ValueError, TypeError):
        return "Invalid input: Please enter a valid integer."
    # Handle special cases
    if num < 0:
        return "Neither: Negative numbers are not classified."
    if num == 0 or num == 1:
        return "Neither: 0 and 1 are neither prime nor composite."
    # Check for prime
    if num == 2:
        return "Prime"
    if num % 2 == 0:
        return "Composite"
    # Optimized prime check: only check odd divisors up to sqrt(num)
    i = 3
    while i * i < num:
        if num % i == 0:
            return "Composite"
        i += 2
    return "Prime"

def main():
    """Main function to test the classifier."""
    test_cases = [2, 3, 4, 15, 17, 20, 97, 100, 1, 0, -5, "abc"]
    print("Number Classification Results:\n")
    for num in test_cases:
        print(classify_number(num))

if __name__ == "__main__":
    main()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\adapala\OneDrive\Desktop\AI Assisted Code> & "C:/Users/adapala/AppData/Local/Programs/Python/Python313/python.exe" "c:/Users/adapala/OneDrive/Desktop/AI Assisted Code/AI-Ass.py"
Number Classification Results:
2: Prime
3: Prime
4: Composite
15: Composite
17: Prime
20: Composite
97: Prime
100: Composite
1: Neither: 0 and 1 are neither prime nor composite.
0: Neither: Negative numbers are not classified.
-5: Neither: Negative numbers are not classified.
abc: Invalid input: Please enter a valid integer.
```

## 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:** Perfect Number Check without providing any examples **CODE:**

The screenshot shows the Visual Studio Code interface with the 'AI ASSISTED CODE' extension active. In the Explorer sidebar, there are two files: '1.4-AI-ASS.py' and 'AI-ASS.py'. The main editor window displays the following Python code:

```

1 def is_perfect_number(n):
2     """
3         Check if a number is a perfect number.
4         A perfect number is a positive integer that equals the sum of its proper divisors
5         (divisors excluding the number itself).
6     """
7     if n <= 1:
8         return False
9
10    divisor_sum = 0
11    for i in range(1, n):
12        if n % i == 0:
13            divisor_sum += i
14
15    return divisor_sum == n
16
17 # Test cases
18 test_numbers = [6, 28, 496, 8128, 10, 1, 0, -6, 100]
19
20 print("Perfect Number Checker")
21 print("-" * 40)
22 for num in test_numbers:
23     result = is_perfect_number(num)
24     print(f"{num}: {result}")
25

```

The status bar at the bottom indicates the file is 'Python' and the code is at 'Line 25, Col 30'. The right sidebar shows recent sessions and an 'Ask about your code' section.

  

The screenshot shows a web browser window with the same Python code displayed. The status bar at the bottom indicates the file is 'Python' and the code is at 'Line 25, Col 30'. The right sidebar shows recent sessions and an 'Ask about your code' section.

  

The screenshot shows a second web browser window with the same Python code displayed. The status bar at the bottom indicates the file is 'Python' and the code is at 'Line 25, Col 30'. The right sidebar shows recent sessions and an 'Ask about your code' section.

**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
- Analyze how examples improve input handling and output clarity.
- Test the program with negative numbers and non-integer inputs.

## PROMPT:

## Even or Odd Classification with Validation

- Input: 8 → Output: Even
  - Input: 15 → Output: Odd
  - Input: 0 → Output: Even

## **CODE:**

The screenshot displays two instances of an AI-powered code editor interface, likely from Microsoft's Visual Studio Code, comparing the generated code quality and execution results.

**Top Editor (AI Assisted Code):**

- Code:**

```
1 def classify_number(user_input):
2     """
3         Classify a number as even or odd with proper input validation.
4     """
5     Examples:
6     - Input: 8 → Output: Even
7     - Input: 15 → Output: Odd
8     - Input: 0 → Output: Even
9     - Input: -4 → Output: Even
10    - Input: -7 → Output: Odd
11
12    try:
13        # Validate and convert input to integer
14        number = int(user_input)
15
16        # Classify as even or odd
17        if number % 2 == 0:
18            return f"Input: {number} → Output: Even"
19        else:
20            return f"Input: {number} → Output: Odd"
21
22    except ValueError:
23        return f"Error: '{user_input}' is not a valid integer. Please enter a whole number."
24
25
26 # Test cases
27 if __name__ == "__main__":
28     test_inputs = [8, 15, 0, -4, -7, 3.5, "abc", -10, 1]
29
30     for test in test_inputs:
31         result = classify_number(test)
32         print(result)
```
- Recent Sessions:** Shows three sessions: "Code optimization and... completed", "Code optimization for... completed", and "Code optimization for... completed".
- Bottom Editor (AI Assisted Code):**
- Code:**

```
1 def classify_number(user_input):
2     """
3         Classify a number as even or odd with proper input validation.
4     """
5     Examples:
6     - Input: 8 → Output: Even
7     - Input: 15 → Output: Odd
8     - Input: 0 → Output: Even
9     - Input: -4 → Output: Even
10    - Input: -7 → Output: Odd
11
12    try:
13        # Validate and convert input to integer
14        number = int(user_input)
15
16        # Classify as even or odd
17        if number % 2 == 0:
18            return f"Input: {number} → Output: Even"
19        else:
20            return f"Input: {number} → Output: Odd"
21
22    except ValueError:
23        pass
```
- Output Terminal:**

```
PS C:\Users\adapala\OneDrive\Desktop\AI Assisted Code> & "C:/Users/adapala\OneDrive\Desktop\AI Assisted Code/AI-ASS.py"
Testing Even/Odd Classification:
Input: 8 → Output: Even
Input: 15 → Output: Odd
Input: 0 → Output: Even
Input: -4 → Output: Even
Input: -7 → Output: Odd
Input: 3 → Output: Odd
Error: "abc" is not a valid integer. Please enter a whole number.
Input: -10 → Output: Even
Input: 1 → Output: Odd

PS C:\Users\adapala\OneDrive\Desktop\AI Assisted Code>
```
- Recent Sessions:** Shows three sessions: "Code optimization and... completed", "Code optimization for... completed", and "Code optimization for... completed".