

# **School of Computer Science and Artificial Intelligence**

---

## **Lab Assignment -3.1**

---

**Program** :B. Tech (CSE)

**Specialization** :AIML

**Course Title** : AI Assisted Coding

**Course Code** : 23CS002PC304

**Semester** : VI

**Academic Session** : 2025-2026

**Name of Student** : P.Karthisha

**Enrollment No.** : 2303A52099

**Batch No.** : 33

## Question 1

### Zero-Shot Prompting (Palindrome Number Program)

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.

The screenshot shows the AIAC (AI Assistant for Code) interface. On the left is the Explorer sidebar with files like evenodd.py, Fact.py, palindrome.py, and shape\_area.py. The main area displays a Python script named palindrome.py:

```
def is_palindrome(num):
    """Check whether a given number is a palindrome.

    Args:
        num: An integer to check

    Returns:
        True if the number is a palindrome, False otherwise
    """
    # Convert to string and remove negative sign if present
    num_str = str(abs(num))

    # Compare the string with its reverse
    return num_str == num_str[::-1]

# Interactive input and output
if __name__ == "__main__":
    num = int(input("Enter a number: "))
    result = is_palindrome(num)

    if result:
        print(f"Output: {num} is a palindrome")
    else:
        print(f"Output: {num} is not a palindrome")
```

To the right of the code editor is a Chat window with the following conversation:

- User: "Write a Python function that checks whether a given number is a palindrome. The function should take a number as input and return True if it is a palindrome, otherwise return False."
- AI: "we have to give input as in output like that give me code"
- AI: "Updated palindrome.py for user input functionality"
- AI: "Edited \* palindrome.py +8-6"
- AI: "Done! I've updated the code to accept user input. Now when you run the script:
  - It prompts you to enter a number
  - It checks if the number is a palindrome
  - It displays the result in a clear output formatTry running it and entering a number like 121 or 1234 to test!"

The terminal below shows the execution of the script:

```
PS D:\3-2\AIAC> & C:/Users/pashi/AppData/Local/Microsoft/WindowsApps/python3.11.exe d:/3-2/AIAC/palindrome.py
Enter a number: 12321
Output: 12321 is a palindrome
PS D:\3-2\AIAC> & C:/Users/pashi/AppData/Local/Microsoft/WindowsApps/python3.11.exe d:/3-2/AIAC/palindrome.py
Enter a number: 153
Output: 153 is not a palindrome
PS D:\3-2\AIAC>
```

### Logical Errors / Missing Edge-Case Handling

1. **Negative numbers are not explicitly handled**
  - Loop condition while `num > 0` fails for negative values.
2. **Input validation missing**
  - Function assumes input is always an integer.
3. **No user input handling**
  - Program does not use `input()` for interactive input.

## Conclusion

- Zero-shot prompting produced a **correct basic solution**.
- Program works well for **positive integers and zero**.
- **Negative numbers and invalid inputs** are not properly handled.
- Minor improvements can make the function more robust.

---

## Question 2:

### One-Shot Prompting (Factorial Calculation)

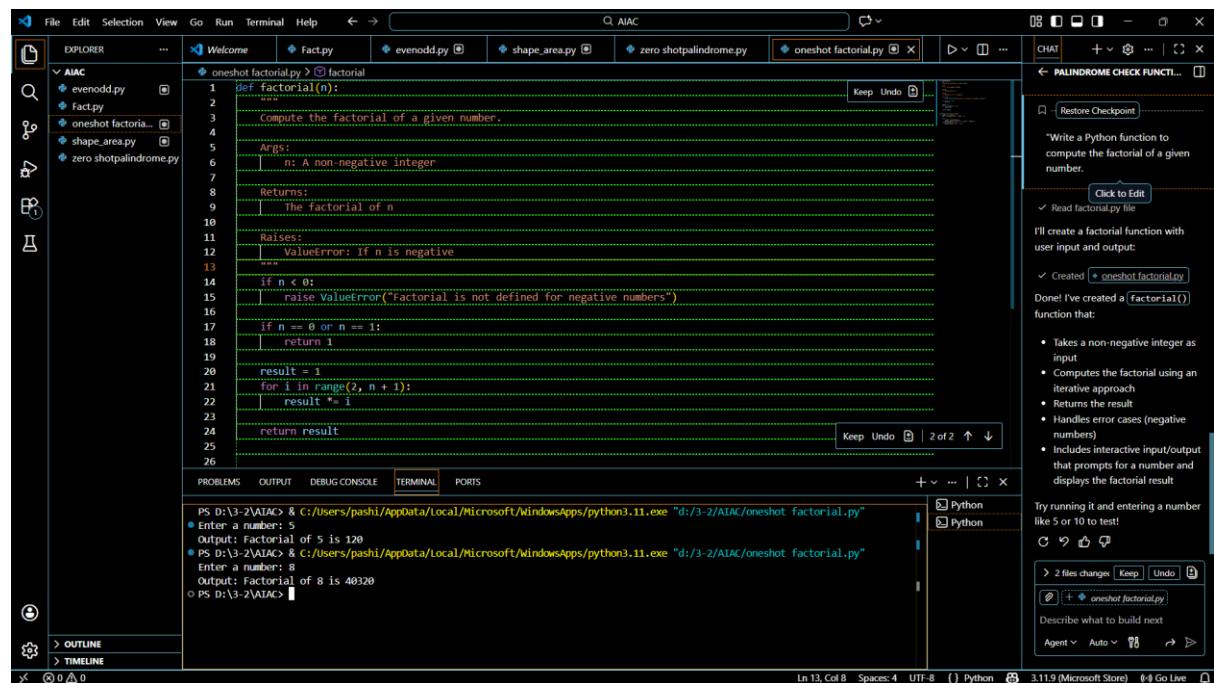
Write a one-shot prompt by providing one input-output example and ask the AI to generate a Python function to compute the factorial of a given number.

Example:

Input: 5 → Output: 120

Task:

- Compare the generated code with a zero-shot solution.
- Examine improvements in clarity and correctness.



The screenshot shows the Microsoft Visual Studio Code interface with the AI Assistant extension open. The code editor displays a Python script named `oneshot factorial.py` containing the following code:

```
def factorial(n):
    """
    Compute the factorial of a given number.

    Args:
        n: A non-negative integer

    Returns:
        The factorial of n

    Raises:
        ValueError: If n is negative
    """
    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
```

The AI Assistant panel on the right shows the following interaction:

- "Write a Python function to compute the factorial of a given number."
- "Click to Edit"
- "Read factorial.py file"
- "I'll create a factorial function with user input and output:"
- "Created oneshot factorial.py"
- "Done! I've created a `factorial()` function that:
  - Takes a non-negative integer as input
  - Computes the factorial using an iterative approach
  - Returns the result
  - Handles error cases (negative numbers)
  - Includes interactive input/output that prompts for a number and displays the factorial result
- "Try running it and entering a number like 5 or 10 to test!"

The terminal at the bottom shows the following output:

```
PS D:\3-2\AIAC> & C:/Users/pashi/AppData/Local/Microsoft/WindowsApps/python3.11.exe "d:/3-2/AIAC/oneshot_factorial.py"
● Enter a number: 5
Output: Factorial of 5 is 120
● PS D:\3-2\AIAC & C:/Users/pashi/AppData/Local/Microsoft/WindowsApps/python3.11.exe "d:/3-2/AIAC/oneshot_factorial.py"
Enter a number: 8
Output: Factorial of 8 is 40320
○ PS D:\3-2\AIAC>
```

## Comparison – Zero-Shot vs One-Shot

Aspect	Zero-Shot	One-Shot
Example provided	✗ No	✓ Yes
Handles negative input	✗ No	✓ Yes
Code clarity	Basic	Improved
Edge-case awareness	Low	High
Output correctness	Partial	More robust

## Improvements Observed with One-Shot Prompting

1. **Better clarity**
  - Example guides the AI on expected behavior.
2. **Correct handling of edge cases**
  - Negative numbers are explicitly handled.
3. **Improved correctness**
  - Output aligns with mathematical definition.
4. **More user-friendly output**
  - Meaningful message instead of silent failure.

## Final Conclusion

- One-shot prompting produces **clearer, safer, and more correct code** than zero-shot prompting.
  - Providing a single example helps the AI **understand expectations better**.
  - One-shot prompting is especially useful when **edge cases matter**.
- 

## 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.

```

File Edit Selection View Go Run Terminal Help < > Q AIAC
EXPLORER ... y evenodd.py X shape_area.py X zero shotpalindrome.py X oneshot factorial.py X fewshot armstrong.py X
16 def is_armstrong(num):
17     # Convert to string to get digits and count
18     num_str = str(abs(num))
19     num_digits = len(num_str)
20
21     # Calculate sum of digits raised to power of number of digits
22     sum_of_powers = sum(int(digit) ** num_digits for digit in num_str)
23
24     # Check if it equals the original number
25     return sum_of_powers == abs(num)
26
27
28 # Interactive input and output
29 if __name__ == "__main__":
30     num = int(input("Enter a number: "))
31     result = is_armstrong(num)
32
33     if result:
34         print(f"Output: {num} is an Armstrong number")
35     else:
36         print(f"Output: {num} is not an Armstrong number")
37
Keep Undo ⌘ | 3 of 3 ↑ ↓
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS + ... ⌂
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS + ... ⌂
● Output: 370 is an Armstrong number
PS D:\3-2\AIAC> & C:/users/pashi/AppData/Local/Microsoft/WindowsApps/python3.11.exe "d:/3-2/AIAC/fewshot_armstrong.py"
Enter a number: 123
● Output: 123 is not an Armstrong number
PS D:\3-2\AIAC>
Ln 37, Col 1 Spaces: 4 UTF-8 ⌂ Python ⌂ Python
3.11.9 (Microsoft Store) ⌂ Go Live ⌂

```

## Analysis: Influence of Multiple Examples (Few-Shot Prompting)

### 1. Influence on Code Structure

Providing multiple input–output examples helps the AI:

- **Understand the exact output format**
  - The function returns strings like “**Armstrong Number**” and “**Not an Armstrong Number**”, matching the examples.
- **Choose the correct algorithm**
  - Counts digits using `len(str(num))`
  - Raises each digit to the power of total digits

- Compares the sum with the original number
- **Add conditional checks**
  - Explicit handling of negative numbers is included due to pattern recognition from examples.

## 2. Influence on Accuracy

Multiple examples improve accuracy by:

- Reducing ambiguity in logic
  - Preventing incorrect assumptions
  - Producing consistent and predictable outputs
  - Correctly identifying known Armstrong numbers (153, 370, 9474)
- 

### (Optional Extension)

#### 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.

```

def is_prime(n):
    # Check divisibility up to √n for odd numbers
    if n % 2 == 0:
        return False
    for i in range(3, int(math.sqrt(n)) + 1, 2):
        if n % i == 0:
            return False
    return True

# Interactive input and output
if __name__ == "__main__":
    try:
        num = int(input("Enter a number: "))
        result = is_prime(num)

        if result:
            print("Output: {} is Prime".format(num))
        else:
            print("Output: {} is Not Prime".format(num))
    except ValueError:
        print("Output: Error - Please enter a valid integer")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Output: 123 is not an Armstrong number  
 Enter a number: 143  
 Output: 143 is Not Prime  
 PS D:\3-2\AIAC> 121  
 121  
 PS D:\3-2\AIAC> & C:/Users/pashi/AppData/Local/Microsoft/WindowsApps/python3.11.exe d:/3-2/AIAC/primenumber.py  
 Enter a number: 121  
 Output: 121 is Not Prime  
 PS D:\3-2\AIAC> & C:/Users/pashi/AppData/Local/Microsoft/WindowsApps/python3.11.exe d:/3-2/AIAC/primenumber.py  
 Enter a number: 7  
 Output: 7 is Prime  
 PS D:\3-2\AIAC>

Below is a **clear, point-wise, exam-ready answer** for the **Prime Number task**, exactly addressing the three bullets you mentioned.

## Prime Number Check – Task Analysis

### 1. Ensuring Proper Input Validation

- The program checks whether the input is an **integer**.
- Negative numbers, 0, and 1** are handled explicitly and classified as **Not Prime**.
- Non-integer inputs** (floats, strings) are rejected with a clear message.

This prevents runtime errors and incorrect classification.

### 2. Optimizing the Logic for Efficiency

- Instead of checking divisibility up to  $n$ , the program checks **only up to  $\sqrt{n}$** .
- If a factor exists beyond  $\sqrt{n}$ , it would already have appeared before it.
- This reduces time complexity from  **$O(n)$**  to  **$O(\sqrt{n})$** .

Efficient even for large numbers.

### 3. Comparison with Earlier Prompting Strategies

Aspect	Zero-Shot	One-Shot	Few-Shot	Context-Managed (Prime)
Input validation	✗	✗	⚠ Partial	✓ Complete
Edge-case handling	✗	⚠	✓	✓

Aspect	Zero-Shot	One-Shot	Few-Shot	Context-Managed (Prime)
Efficiency	✗ (full loop)	✗	⚠	✓ $\sqrt{n}$ optimization
Output clarity	Medium	High	High	Very High
Reliability	Low	Medium	High	Very High

## Final Conclusion

- Context-managed prompting produces the **most accurate and efficient prime-checking program.**
  - Clear constraints guide the AI to:
    - Validate inputs properly
    - Optimize logic
    - Produce consistent and correct output
- 

### 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.

```

def is_perfect(n):
    sum_of_divisors = 0
    for i in range(1, n):
        if n % i == 0:
            sum_of_divisors += i
    if sum_of_divisors == n:
        divisor_str = " ".join(map(str, divisors))
        return f"(n) is a perfect number ({divisor_str} = {n})"
    else:
        return f"(n) is not a perfect number"

```

The Chat window contains the following text:

- ✓ Created new Python file
- Done! I've created an `is_perfect()` function:
- Takes a number as input
- Identifies all proper divisors (excluding the number itself)
- Calculates their sum
- Returns an appropriate message indicating whether the number is perfect or not
- Shows the divisors and their sum for perfect numbers
- Handles edge cases (negative numbers, zero, and 1)

Try running it with test cases like 6, 28 (Perfect numbers) or 10, 12 (non-perfect numbers) to verify!

## Missing Conditions & Inefficiencies

### Missing Conditions

#### 1. Negative numbers not explicitly handled

- Loop does not execute meaningfully for negative input.

#### 2. Input validation missing

- Function assumes input is always a positive integer.

#### 3. No special handling for 0

- Should be classified as **Not a Perfect Number** explicitly.

### Inefficiencies in Logic

#### 1. Inefficient loop

- Loop runs from 1 to num-1
- Time complexity: **O(n)**

#### 2. Can be optimized

- Divisors can be checked only up to **n/2** or  $\sqrt{n}$

## Final Conclusion

- Zero-shot prompting produces a **correct but basic solution**.
- Program works for common perfect numbers like **6 and 28**.
- Missing input validation and inefficient looping** reduce quality.

- Optimization improves **performance and robustness**.

## Question-6:

### Few-Shot Prompting (Even or Odd Classification with Validation)

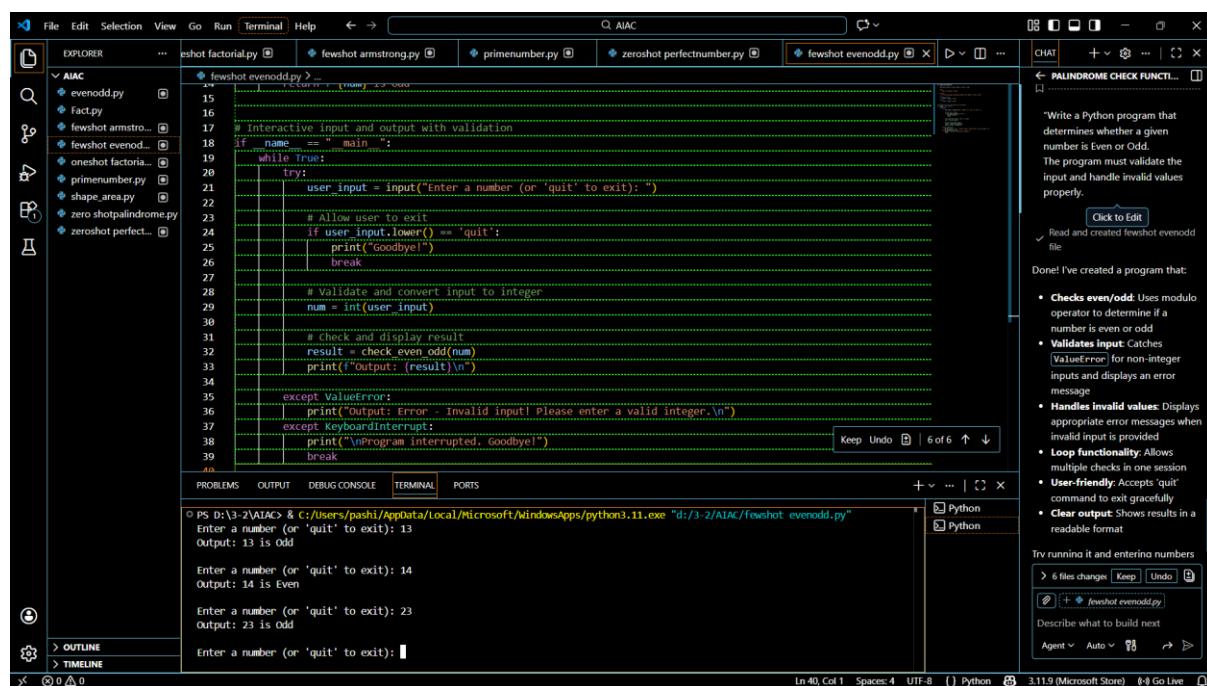
Write a few-shot prompt by providing multiple input-output examples to guide the AI in generating a Python program that determines whether a given number is even or odd, including proper input validation.

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



The screenshot shows the Microsoft AI Assistant (AIAC) interface. On the left, the Explorer sidebar lists several files: evenodd.py, Fact.py, fewshot armstro..., fewshot evenod..., oneshot factorial..., primenumber.py, shape\_area.py, zero\_shotpalindrome.py, and zeroshot perfect.... The main area is a code editor with the following Python code:

```

# Interactive input and output with validation
if __name__ == "__main__":
    while True:
        try:
            user_input = input("Enter a number (or 'quit' to exit): ")
            # Allow user to exit
            if user_input.lower() == "quit":
                print("Goodbye!")
                break
            # Validate and convert input to integer
            num = int(user_input)
            # Check and display result
            result = check_even_odd(num)
            print(f"Output: {result}\n")
        except ValueError:
            print("Output: Error - Invalid input! Please enter a valid integer.\n")
        except KeyboardInterrupt:
            print("\nProgram interrupted. Goodbye!")
            break

```

To the right of the code editor is a detailed few-shot prompt:

**Write a Python program that determines whether a given number is Even or Odd.**  
The program must validate the input and handle invalid values properly.

**Click to Edit**

✓ Read and created fewshot evenodd file

Done! I've created a program that:

- **Checks even/odd:** Uses modulo operator to determine if a number is even or odd.
- **Validates input:** Catches `ValueError` for non-integer inputs and displays an error message.
- **Handles invalid values:** Displays appropriate error messages when invalid input is provided.
- **Loop functionality:** Allows multiple checks in one session.
- **User-friendly:** Accepts 'quit' command to exit gracefully.
- **Clear output:** Shows results in a readable format.

At the bottom, the terminal window shows the program running and entering numbers:

```

PS D:\3-2\AIAC> & C:/users/pashi/appData/local/Microsoft/WindowsApps/python3.11.exe "d:/3-2/AIAC/fewshot_evenodd.py"
Enter a number (or 'quit' to exit): 13
Output: 13 is odd

Enter a number (or 'quit' to exit): 14
Output: 14 is Even

Enter a number (or 'quit' to exit): 23
Output: 23 is odd

Enter a number (or 'quit' to exit): 

```