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	**: K. Shailaja**
Enrollment No.	**: 2303A52155**
Batch No.	**: 34**

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

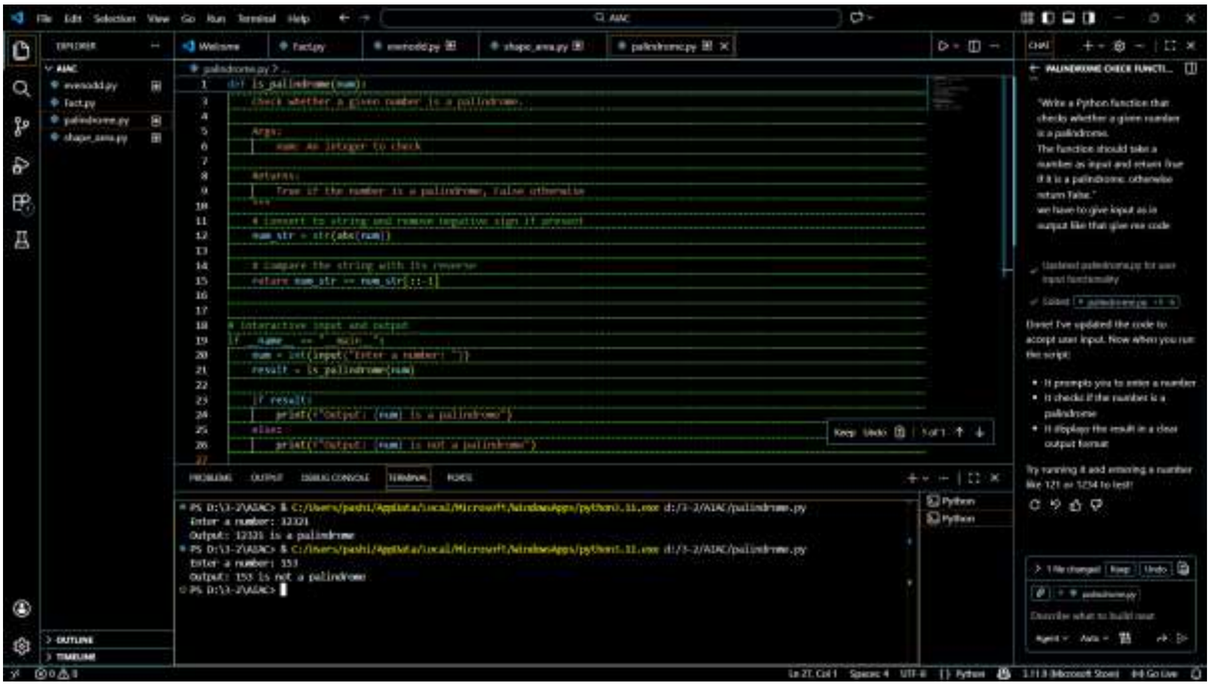
a Python function that checks whether a given number is a palindrome.

Task:

• Record the AI-generated code.

• Test the code with multiple inputs.

• Identify any logical errors or missing edge-case handling.



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.

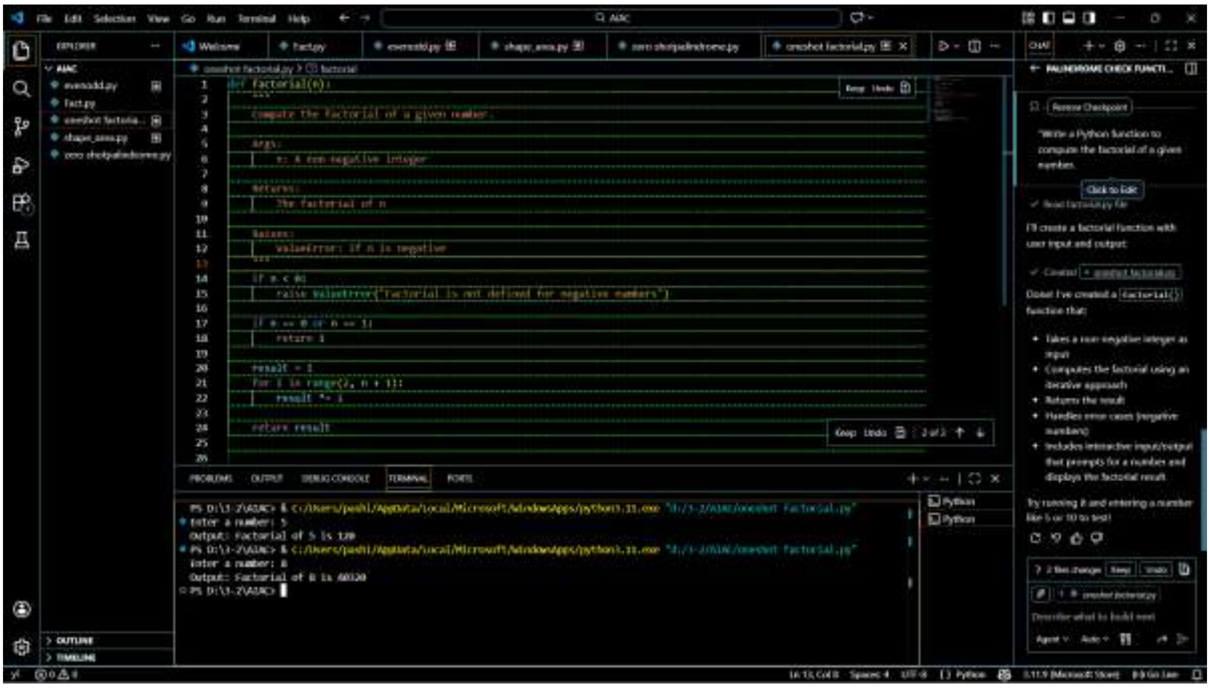
Example:

Input: 5 → Output: 120

Task:

• Compare the generated code with a zero-shot solution.

• Examine improvements in clarity and correctness.



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

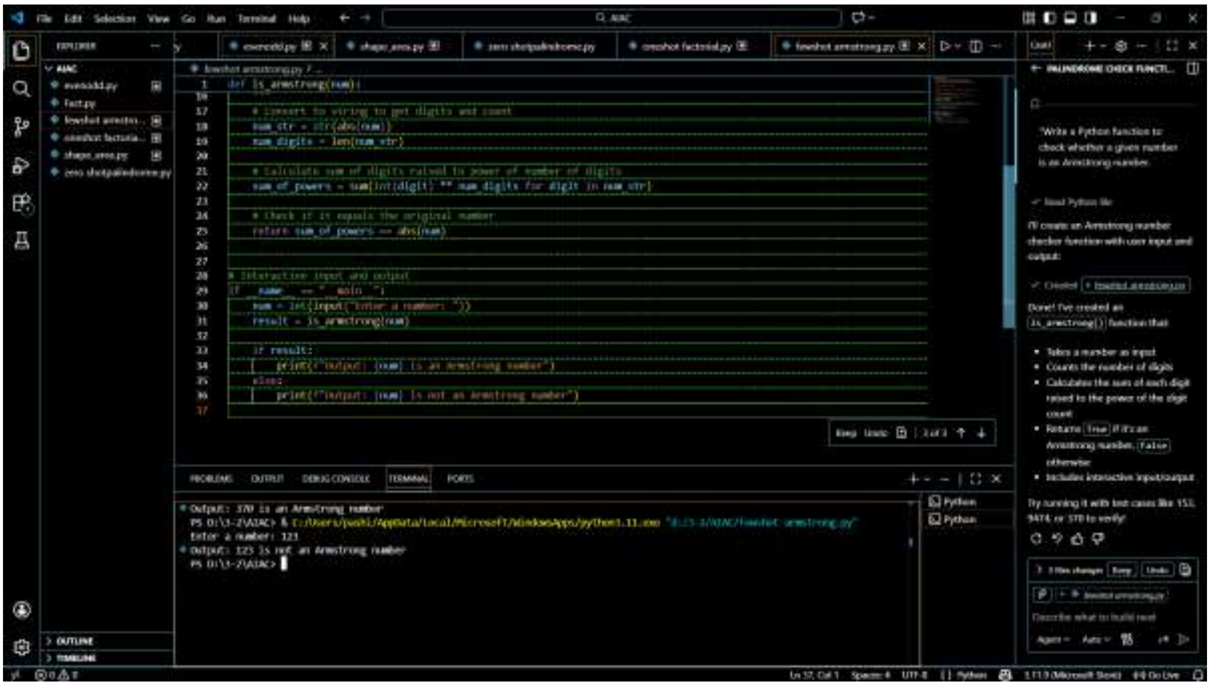
• Input: 370 → Output: Armstrong Number

• 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.



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

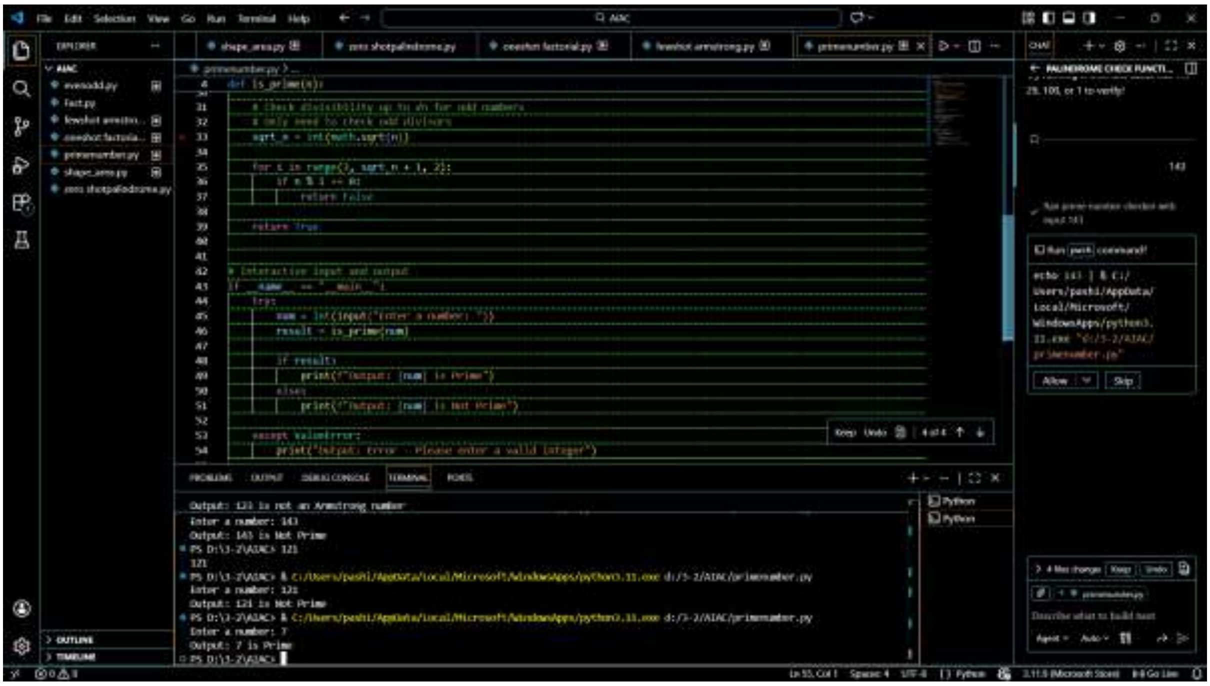
number as prime, composite, or neither.

Task:

• Ensure proper input validation.

• Optimize the logic for efficiency.

• Compare the output with earlier prompting strategies.



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 √n**.
- If a factor exists beyond √n, it would already have appeared before it.
- This reduces time complexity from **O(n)** to **O(√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)	❌	⚠️	✅ √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)

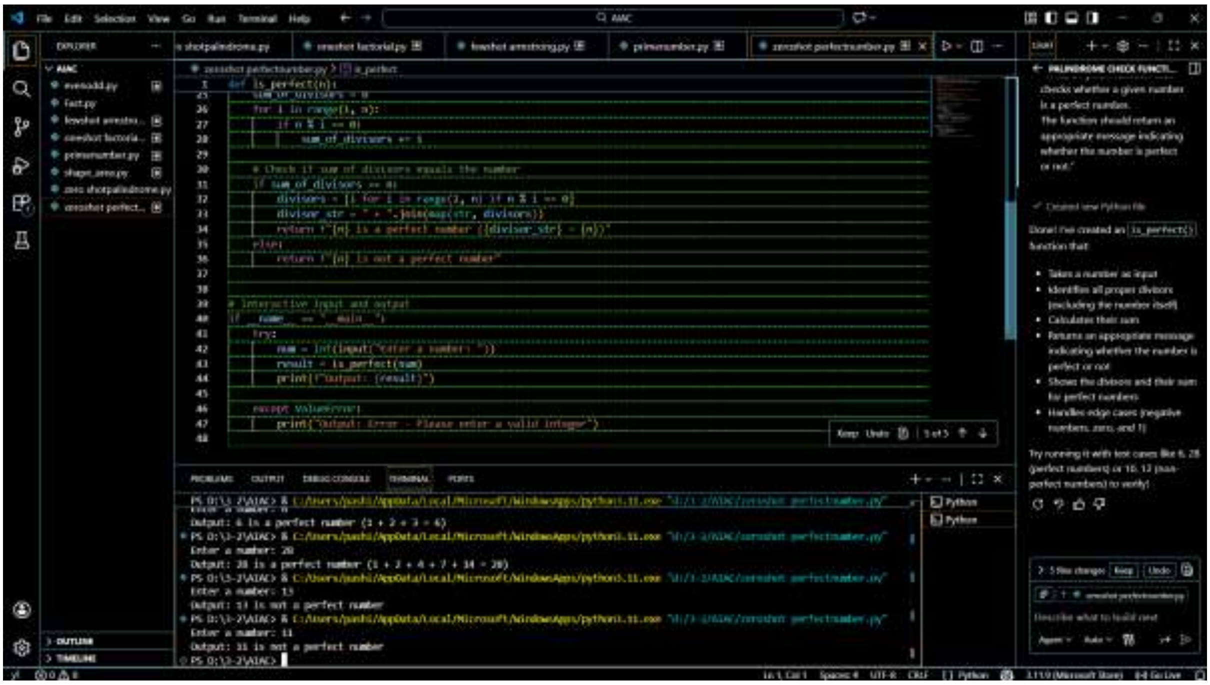
Write a zero-shot prompt (without providing any examples) to generate a Python function that checks whether a given number is a perfect number.

Task:

• Record the AI-generated code.

• Test the program with multiple inputs.

• Identify any missing conditions or inefficiencies in the logic.



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 **√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