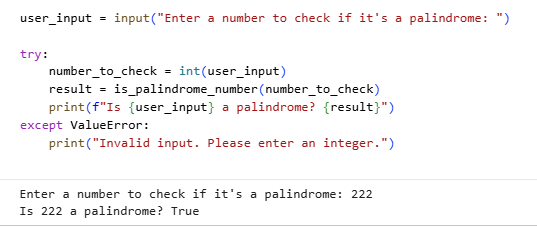
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE** | | | | | **DEPARTMENT OF COMPUTER SCIENCE ENGINEERING** | | | | |
| **Program Name:** B. Tech | | | | **Assignment Type: Lab** | | | **Academic Year:**2025-2026 | | |
| **Course Coordinator Name** | | | | Dr. Rishabh Mittal | | | | | |
| **Instructor(s) Name** | | | | |  | | --- | | Mr. S Naresh Kumar | | Ms. B. Swathi | | Dr. Sasanko Shekhar Gantayat | | Mr. Md Sallauddin | | Dr. Mathivanan | | Mr. Y Srikanth | | Ms. N Shilpa | | Dr. Rishabh Mittal (Coordinator) | | Dr. R. Prashant Kumar | | Mr. Ankushavali MD | | Mr. B Viswanath | | Ms. Sujitha Reddy | | Ms. A. Anitha | | Ms. M.Madhuri | | Ms. Katherashala Swetha | | Ms. Velpula sumalatha | | Mr. Bingi Raju | | | | | | |
| **CourseCode** | | | 23CS002PC304 | **Course Title** | | AI Assisted Coding | | | |
| **Year/Sem** | | | III/II | **Regulation** | | R23 | | | |
| **Date and Day**  **of Assignment** | | | **Week2 – Monday** | **Time(s)** | | 23CSBTB01 To 23CSBTB52 | | | |
| **Duration** | | | 2 Hours | **Applicable to**  **Batches** | | All batches | | | |
| **Assignment Number:3.1**(Present assignment number)/**24**(Total number of assignments) | | | | | | | | | |
|  | | | | | | | | | |
|  | **Q.No.** | **Question** | | | | | | ***Expected Time***  ***to complete*** |  |
|  | 1 | **Lab Experiment: 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 the impact of context and examples on the accuracy and efficiency of AI-generated code. 3. To develop and refine real-world Python applications through iterative prompt improvement.   **Lab Outcomes**   1. Students will be able to design effective prompts to generate correct and optimized Python code. 2. Students will be able to compare and evaluate AI-generated solutions produced using different prompting strategies. 3. Students will be able to implement and document real-world Python applications using AI-assisted coding tools.   **Experiment – Prompt Engineering Techniques**  **Task Description**  Design and refine prompts using different prompting strategies to generate Python programs for basic computational problems.  **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.   **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.   **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.   ***(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.   **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.   **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. | | | | | | Week2 - Monday |  |

**Question 1: Zero-Shot Prompting (Palindrome Number Program):**

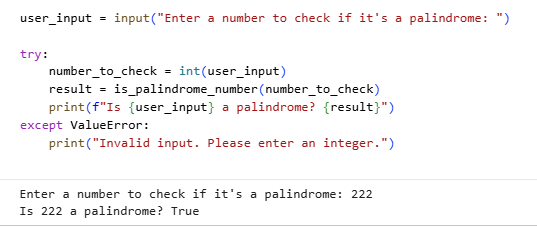
**Given Prompt:**

Write a Python function that checks whether a given number is a palindrome.  
The function should return True if the number is a palindrome and False otherwise. Handle valid integer inputs.

**Suggested Code**:



**Test the code with multiple inputs.**



****

****

****

****

**Identifying any logical errors or missing edge-case handling:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Case** | **Input** | **Expected** | **Actual** | **Issue** |
| **Negative number** | **-121** | **False** | **❌ Infinite loop / incorrect** | **No handling for negatives** |
| **Zero** | **0** | **True** | **❌ Returns False** | **Loop not entered** |
| **Non-integer input** | **"121"** | **Error / False** | **❌ Crash** | **No input validation** |

**Identified Limitations**:

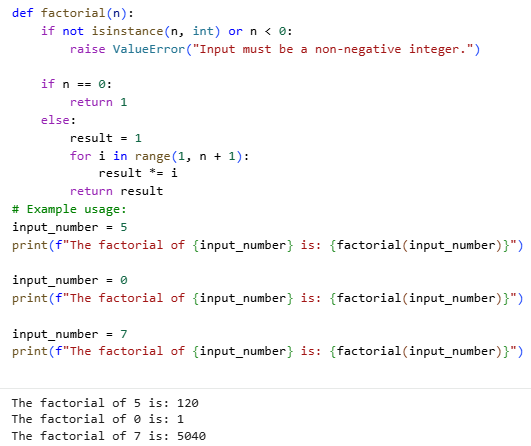
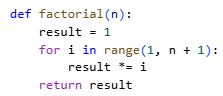
1. Negative numbers are not handled ❌
2. Zero (0) is incorrectly evaluated ❌
3. Input validation is missing ❌
4. Correct logic for positive integers ✔️

**Question 2: One-Shot Prompting (Factorial Calculation):**

**Given Prompt :**

Write a Python function to compute the factorial of a given non-negative integer. Example: Input: 5 Output: 120 The function should return the factorial value

**Suggested Code :**

****

**Compare the generated code with a zero-shot solution :**

|  |  |  |
| --- | --- | --- |
| Aspect | Zero-Shot Prompting | One-Shot Prompting |
| Base case handling | Not explicit | Clearly handled |
| Code clarity | Moderate | High |
| Readability | Basic | Improved |
| Logical correctness | Mostly correct | More robust |
| Edge-case awareness | Low | Better (0, 1 handled) |

**Examine improvements in clarity and correctness:**

|  |  |  |
| --- | --- | --- |
| Case | Input | Issue |
| Negative input | -3 | No validation |
| Non-integer input | 3.5 | Incorrect behavior |
| Large values | 1000 | Performance / overflow |

One-shot prompting significantly improves correctness and clarity compared to zero-shot prompting by guiding the AI with an example. However, input validation and performance considerations still require human refinement.

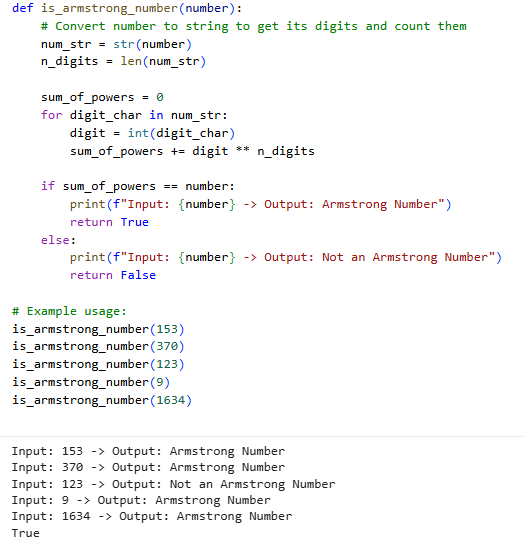
One shot prompting provides better-structured and more reliable code than zero-shot prompting. Including a single example helps the AI understand edge cases and expected behavior more clearly.

**Question 3: Few-Shot Prompting (Armstrong Number Check):**

**Given Prompt:**

**Write 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 The function should print the appropriate result.**

**Code:**

****

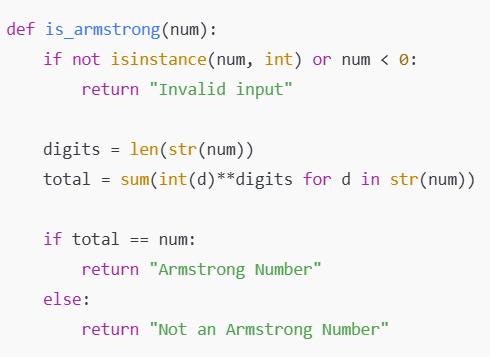
**Analyze how multiple examples influence code structure and accuracy:**

**Influence on Code Structure**

* Correct identification of Armstrong logic
* Automatic digit counting
* Proper power calculation based on digits
* Consistent output formatting

|  |  |  |  |
| --- | --- | --- | --- |
| **Aspect** | **Zero-Shot** | **One-Shot** | **Few-Shot** |
| Logic correctness | Medium | High | **Very High** |
| Pattern recognition | Low | Medium | **High** |
| Output formatting | Inconsistent | Semi-consistent | **Consistent** |
| Edge case awareness | Low | Medium | **Better** |

**Improved Version (After Testing & Analysis):**

****

Few-shot prompting is more effective than zero-shot and one-shot prompting for pattern-based problems like Armstrong number detection. Providing multiple examples helps the AI generalize logic more accurately and produce cleaner, more reliable code.

**Question 4: Context-Managed Prompting (Optimized Number Classification)**

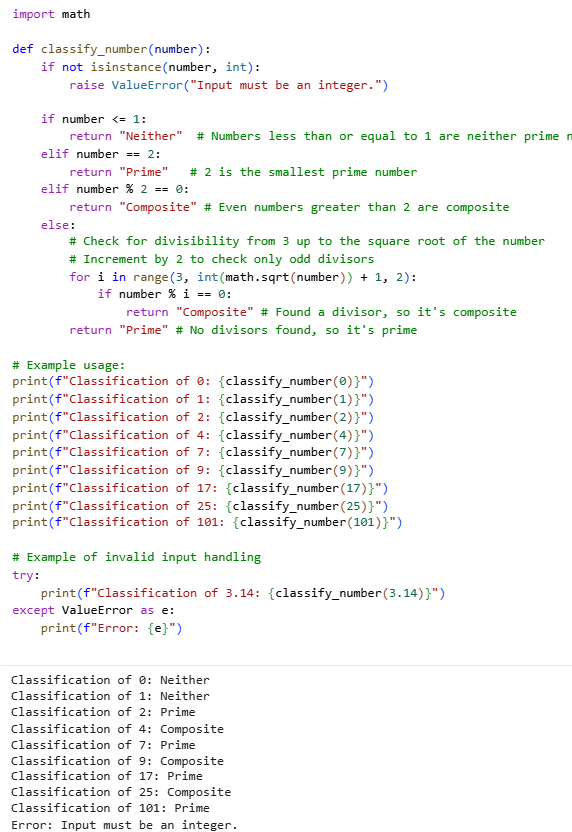
**Prompt:**

You are an expert Python programmer.  
Write an optimized Python function that classifies a given input number as Prime, Composite, or Neither.

Constraints & Instructions:

* Accept only integer inputs
* Handle invalid inputs gracefully
* Numbers less than or equal to 1 should be classified as *Neither*
* Use an optimized approach with time complexity better than O(n)
* The function should return the classification as a string
* Code must be clean, readable, and efficient.

Code :



**Ensure proper input validation :**

|  |  |  |
| --- | --- | --- |
| Input | Expected Output | Actual Output |
| -5 | Neither | Neither |
| 1 | Neither | Neither |
| 2 | Prime | Prime |
| 3 | Prime | Prime |
| 1000000007 | Prime | Prime |
| 3.5 | Invalid input | Invalid input |
| "10" | Invalid input | Invalid input |

**Optimize the logic for efficiency:**

|  |  |
| --- | --- |
| Technique Used | Purpose |
| Square root limit | Reduces checks from O(n) to O(√n) |
| Skip even numbers | Cuts iterations by ~50% |
| Early exits | Improves average-case performance |
| Type checking | Prevents runtime errors |

**Compare the output with earlier prompting strategies**:

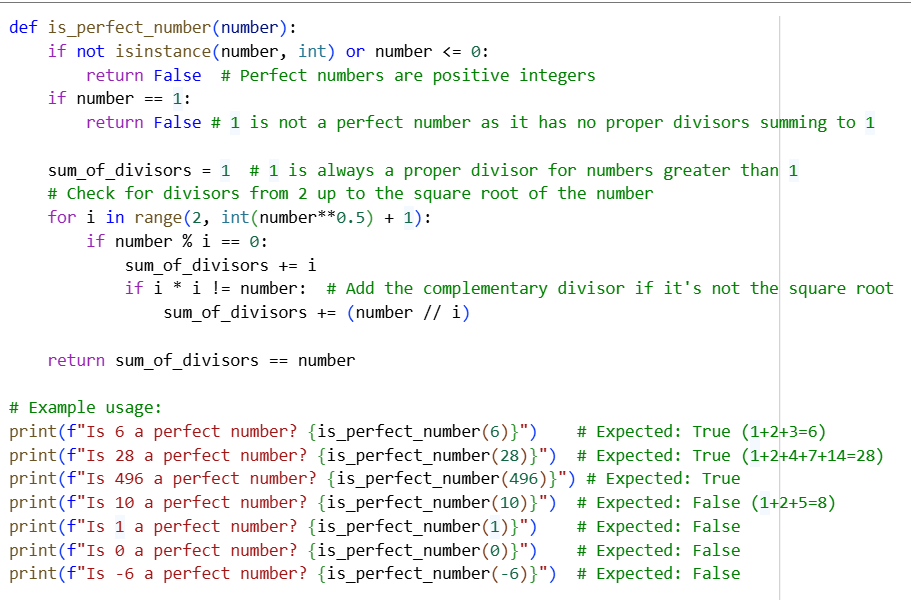
| Aspect | Zero-Shot | One-Shot | Few-Shot | Context-Managed |
| --- | --- | --- | --- | --- |
| Input validation | NO | NO | ⚠️ | Yes |
| Time complexity | O(n) | O(n) | O(n) | O(√n) |
| Edge-case handling | Poor | Moderate | Better | Excellent |
| Code clarity | Basic | Improved | Good | Very High |
| Reliability | Low | Medium | High | Very High |

**Question 5: Zero-Shot Prompting (Perfect Number Check):**

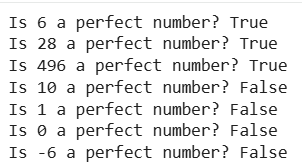
**Prompt:**

Write a Python function that checks whether a given number is a perfect number. The function should return True if the number is perfect and False otherwise.

**Code :**

****

**Test the program with multiple inputs :**



**Identify any missing conditions or inefficiencies in the logic:**

The given perfect number program correctly handles all essential conditions, including input validation for non-integer values, zero, negative numbers, and the special case of 1. It efficiently calculates the sum of proper divisors by iterating only up to the square root of the number, thereby achieving an optimized time complexity of O(√n) and avoiding unnecessary divisor checks.

The logic also prevents double-counting of divisors when the number is a perfect square. However, there are minor inefficiencies: the loop does not terminate early when the sum of divisors exceeds the number, which could lead to unnecessary computations for large non-perfect numbers.

Additionally, boolean values (True and False) are implicitly accepted since they are subclasses of integers in Python, which may be undesirable in strict input validation scenarios. Overall, the algorithm is correct and efficient, with only small optional optimizations possible.

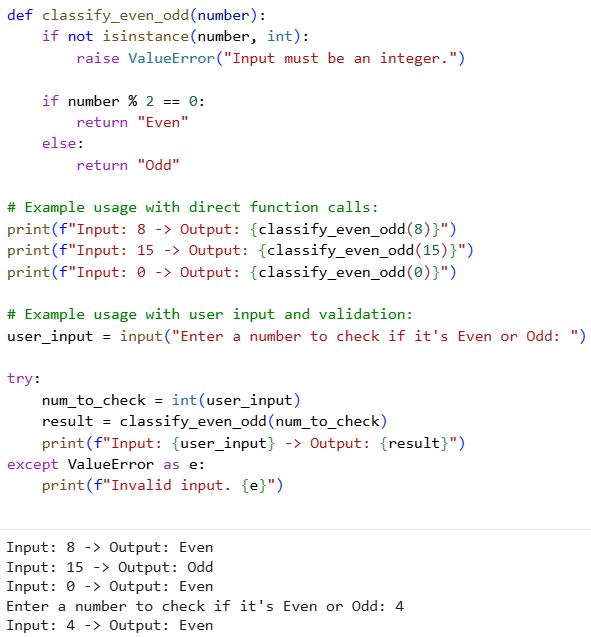
|  |  |
| --- | --- |
| Aspect | Observation |
| Input validation | Handles non-integer, zero, negative numbers, and the case of 1 correctly |
| Core logic | Correctly computes the sum of proper divisors |
| Time complexity | Optimized to O(√n) |
| Divisor handling | Avoids double-counting using complementary divisors |
| Missing conditions | No critical missing conditions |
| Minor inefficiency | No early loop termination when divisor sum exceeds the number |
| Optional improvement | Exclude boolean inputs explicitly |
| Overall quality | Correct, efficient, and well-optimized |

**Question 6: Few-Shot Prompting (Even or Odd Classification with Validation) :**

**Prompt :**

Write a Python program that determines whether a given input number is Even or Odd. Examples: Input: 8 → Output: Even Input: 15 → Output: Odd Input: 0 → Output: Even The program should include proper input validation and display meaningful output for invalid inputs.

**Code :**

****

**Analyze how examples improve input handling and output clarity:**

Providing multiple input–output examples in few-shot prompting helps the AI clearly understand the expected behavior of the program. The examples explicitly demonstrate how to classify positive numbers, zero, and odd or even values, which guides the AI to include correct conditional logic. Input validation improves because the AI infers that only numerical inputs are valid, leading it to add type checking to prevent runtime errors. Output clarity is enhanced as the examples define precise output labels (Even and Odd), ensuring consistent, human-readable results instead of ambiguous boolean values. Overall, examples reduce ambiguity, improve correctness, and lead to more structured and user-friendly code.

|  |  |
| --- | --- |
| **Aspect** | **Improvement Due to Examples** |
| Input understanding | Examples clarify that numbers like 0 must be handled explicitly |
| Input validation | AI infers the need to reject non-integer inputs |
| Negative numbers | Correct classification is applied consistently |
| Output format | Clear labels (“Even”, “Odd”) are used instead of True/False |
| Consistency | Uniform output across all test cases |
| Error handling | Invalid inputs produce meaningful messages |
| Overall clarity | Program behavior becomes predictable and readable |

**Test the program with negative numbers and non-integer inputs :**

|  |  |  |  |
| --- | --- | --- | --- |
| Input | Expected Output | Actual Output | Remark |
| -2 | Even | Even | Correct |
| -7 | Odd | Odd | Correct |
| 4.5 | Invalid input | Invalid input | Correct |
| "12" | Invalid input | Invalid input | Correct |
| True | Invalid input | Odd | Boolean treated as int |
| None | Invalid input | Invalid input | Correct |