Software Development Life Cycle(SDLC) stages:

**1. Requirement Analysis:**

Identification of Requirements:

* + Understand the purpose of the calculator, which is to perform various mathematical operations.
  + Identify supported operations such as addition, subtraction, multiplication, division, modulus, comparisons, logical OR, logical XOR, and factorial calculations.
  + Specify accepted input types, including integers and floats, and consider potential error scenarios, particularly division and modulus by zero.

User Expectations:

* + Gather information about user expectations, and potential use cases for the calculator.
  + Ensure that the calculator meets user needs and provides a good experience.

**2. Design:**

Overall Structure:

* + Plan the overall structure of the program, emphasizing modular design for functions like matrix printing, input validation, and mathematical operations
  + Define functions for each operation (addition, subtraction, etc.) to organize the code logically.

User Interface:

* + Outline how users will interact with the calculator, including input prompts and result display.
  + Consider user-friendly design for ease of use.

Function Design:

* + Design functions for arithmetic operations (addition, subtraction, multiplication, division), comparisons, logical operations (AND, OR, XOR), and factorial calculations.
  + Establish a clear plan for input validation and error handling.

**3. Implementation (Coding):**

Code Development:

* + Write the Python code based on the design plan, implementing the defined functions.
  + Include input prompts to guide users in entering numbers and selecting operations.

Functionality:

* + Code functions for various operations, ensuring they correctly perform the intended calculations.
  + Implement error handling for potential issues, such as division by zero or modulus by zero.

Code Quality:

* + Ensure code readability, maintainability, and adherence to coding standards.
  + Use meaningful variable names and comments to enhance code understanding.

**4. Testing:**

Validation:

* + Conduct thorough testing to validate the correctness of the calculator in different scenarios.
  + Test with positive and negative numbers, decimals, zero, and combinations of various operations.

Error Handling Validation:

* + Verify that error handling mechanisms work as expected, especially for division and modulus by zero scenarios.
  + Check that appropriate error messages are displayed when necessary.

Output Verification:

* + Check the output for each operation to ensure it matches the expected results.
  + Validate that the calculator produces accurate and reliable results.

**5. Deployment:**

Release:

* + Once testing is successful, make the calculator available for users.
  + Ensure that clear instructions are provided to users on how to input values and select operations.

**6. Maintenance:**

User Feedback:

* + Monitor user feedback and promptly address any reported issues.
  + Consider user suggestions for improvements or new features.

Updates:

* + Make updates to the code to address any discovered bugs or enhance existing features.
  + Ensure that the calculator remains compatible with evolving user needs.

**7. Documentation:**

User Manual:

* + Create comprehensive documentation, including a user manual explaining how to use the calculator.
  + Provide clear instructions on input methods, supported operations, and error handling.

Code Documentation:

* + Document the code for future reference and maintenance.
  + Include comments and documentation to help developers understand the code's purpose and functionality.

**8. Continuous Improvement:**

User Feedback Analysis:

* + Gather user feedback over time to identify areas for improvement.
  + Consider implementing new features or optimizing existing ones based on user requirements.

Updates for Enhancement:

* + Make regular updates to enhance the calculator's functionality and user experience.
  + Ensure that the calculator evolves to meet changing user expectations.

**9. Version Control:**

Code Versioning:

* + Implement version control to track changes made to the calculator code.
  + Maintain a version history to facilitate rollback, reference, and collaboration among developers.

This explanation emphasizes the iterative and continuous nature of the SDLC, highlighting the importance of thorough testing, user documentation, and ongoing improvement to deliver a reliable and user-friendly Python calculator application.