Defensive programming is a software development approach focused on writing code that is resilient to errors and unexpected conditions. The goal is to anticipate potential issues and failures and handle them gracefully, making the software more robust, reliable, and maintainable.

1. **Input Validation:**
   - Validate input data to ensure it meets expected criteria.
2. **Error Handling:**
   - Use try-except blocks to catch and handle exceptions.
   - Provide meaningful error messages to aid in debugging and troubleshooting.
   - Log errors to facilitate diagnosis and resolution.
3. **Code Reviews:**
   - Conduct thorough code reviews to identify potential issues.
   - Review code for correctness, clarity, and adherence to coding standards.
4. **Documentation:**
   - Document code clearly to help developers understand its purpose and behaviour.
   - Include information about potential pitfalls, edge cases, and error-handling strategies.
5. **Modularity and Encapsulation:**
   - Functions: Design code in a modular way, breaking it into smaller, manageable components.
   - Classes: Enclose functionality in the object to limit the impact of changes on other parts of the system.
6. **Testing:**
   - Write comprehensive unit tests to cover various scenarios and edge cases. It is always good to get someone else's input and not code for your assumptions.
7. **Graceful Exits:**
   - Design systems to gracefully handle unexpected situations and continue functioning even if in a degraded state.
8. **Consistent Coding Standards:**
   - Enforce consistent coding standards to improve code readability and maintainability.
   - Consistency helps prevent errors caused by misunderstandings or inconsistent practices.
9. **Resource Management:**
   - Properly manage resources (memory, file handles, database connections) and release them when no longer needed.