Salesforce Test Classes Best Practices

- **Use Descriptive Names**: Name test classes clearly to indicate what functionality they are testing.
- **Isolate Test Data**: Create test data within the test class to ensure tests are independent of data created in other tests.
- **Use @TestSetup**: Use @TestSetup methods to create common test data for multiple test methods, improving efficiency.
- **Use System.assert()**: Assert expected outcomes using System.assert() methods to validate behavior.
- **Test Positive and Negative Cases**: Cover both positive (expected) and negative (unexpected) scenarios in your test methods.
- **Avoid Hardcoding IDs**: Use Salesforce's built-in Id creation methods or query for record IDs dynamically to avoid hardcoding IDs.
- **Test Governor Limits**: Ensure tests don't exceed Salesforce's governor limits to prevent deployment failures.

- **Test Bulk Operations**: Test the behavior of your code with bulk data to ensure it performs well under various loads.

- **Use Test.startTest() and Test.stopTest():** Use these methods to separate setup and execution phases, ensuring accurate governor limit counting.

- **Use System.runAs():** Test code behavior under different user contexts using System.runAs() to cover sharing rules and permissions.

- **Test Exception Handling**: Cover exception handling scenarios to ensure your code behaves as expected when errors occur.

- **Test Asynchronous Code**: Use Test.startTest() and Test.stopTest() to test asynchronous code like future methods and queueable jobs.

- **Keep Test Classes Up-to-date**: Regularly review and update test classes as your codebase evolves to maintain effective test coverage.

- **Use Test.isRunningTest():** Conditionally execute code blocks within your classes using Test.isRunningTest() to handle test-specific logic.

- **Document Your Tests**: Add comments to your test methods explaining what they are testing and why, aiding in understanding and maintenance