Best Practice for Test Classes in Salesforce :

Test Interview Questions

Link : https://www.apexhours.com/apex-test-class-best-practices/

To deploy to production at least 75% code coverage is required, But your focus shouldn't be on the percentage of code that is covered. Instead, you should make sure that every use case of your application is covered, including positive and negative cases, as well as bulk and single records. This should lead to 75% or more of your code being covered by unit tests.
Test class and method default access are private, no need to add an access specifier.
Stating with salesforce API 28.0 test method can not reside inside non test classes.
Always use the @testSetup method dedicated to creating test records for that class. This is a newly added annotation and very powerful.
If possible Don't use seeAllData=true, Create your Own Test Data. SeeAllData=true will not work for API 23 version earlier.
User, profile, organization, AsyncApexjob, Corntrigger, RecordType, ApexClass, ApexComponent ,ApexPage we can access without (seeAllData=true).
Test method and test classes are not counted as a part of code limit.
Test method takes no argument, commit no data to database, send no email, flagged with testMethod keyword.
Use Test.startTest() to reset Governor limits in Test methods.
If you are doing any Asynchronous operation in code, then don't forget to call Test.stopTest() to make sure that operation is completed.
Use System.runAs() method to enforce OWD and Profile related testings. This is very important from Security point of View.
Use As much as Assertions like System.AssertEquals or System.AssertNotEquals.
Always test Batch Capabilities of your code by passing 20 to 100 records.
Always try to pass null values in every methods. This is the area where most of program fails, unknowingly.
Please use call-out mock to test web-service call-out.

==========

1. Test class must start with @isTest annotation if the class version is more than 25
2. Test environment support @testVisible, @testSetUp as well
3. Unit test is to test a particular piece of code working properly or not.
4. Unit test method takes no argument, commits no data to database, sends no email, flagged with testMethod keyword.
5. To deploy to production at least 75% code coverage is required

6. System. debug statements are not counted as a part of the apex code limit.
7. Test method and test classes are not counted as a part of the code limit
9. We should not focus on the percentage of code coverage, we should make sure that every use case should be covered including positive, negative, bulk, and single record.

Single Action -To verify that the single record produces the correct and expected result.

Bulk action -Any apex record trigger, class, or extension must be invoked for 1-200 records.

Positive behavior: Test every expected behavior that occurs through every expected permutation, l,e user filled out every correct data and not go past the limit.

Negative Testcase:-Not to add future date, Not to specify the negative amount.

Restricted User:-Test whether a user with restricted access used in your code.
10. Test class should be annotated with @isTest.
11. @isTest annotation with test method is equivalent to testMethod keyword.
12. Test method should static and no void return type .
13. Test class and method default access is private ,no matter to add access specifier .
14. classes with @isTest annotation can't be a interface or enum .
15. Test method code can't be invoked by non test request .
16. Stating with salesforce API 28.0 test method can not reside inside non test classes .
17. @Testvisible annotation to make visible private methods inside test classes.
18. Test method can not be used to test web-service call out . Please use call out mock .
19. You can't send email from test method.
20. User, profile, organization, AsyncApexjob, Corntrigger, RecordType, ApexClass, ApexComponent ,ApexPage we can access without (seeAllData=true) .
21. SeeAllData=true will not work for API 23 version eailer .
22. Accessing static resource test records in test class e,g List<Account> accList=Test.loadData(Account,SobjectType,'ResourceName').
23. Create TestFactory class with @isTest annotation to exclude from organization code size limit.
24. @testSetup to create test records once in a method and use in every test method in the test class .
25. We can run unit test by using Salesforce Standard UI,Force.com IDE ,Console ,API.
26. Maximum number of test classes run per 24 hour of period is not grater of 500 or 10 multiplication of test classes of your organization.
27. As apex runs in system mode so the permission and record sharing are not taken into account . So we need to use system.runAs to enforce record sharing .
28. System.runAs will not enforce user permission or field level permission .
29. Every test to runAs count against the total number of DML issued in the process .

==========

1. Test class must start with @isTest annotation if class class version is more than 25
2. Test environment support @testVisible, @testSetup as well
3. Unit test is to test particular piece of code working properly or not .
4. Unit test method takes no argument, send no email , commit no data to database and flagged with testMethod keyword .
5. To deploy to production at least 75% code coverage is required
6. Test method and test classes are not counted as a part of code limit7. System.debug statement are not counted as a part of apex code limit.8. We should not focus on the percentage of code coverage ,we should make sure that every use case should covered including positive, negative,bulk and single record .

Single Action -To verify that the the single record produces the correct an expected result .

Bulk action -Any apex record trigger ,class or extension must be invoked for 1-200 records .

Positive behavior : Test every expected behavior occurs through every expected permutation , i,e user filled out every correctly data and not go past the limit .

Negative Testcase :-Not to add future date , Not to specify negative amount.

Restricted User :-Test whether a user with restricted access used in your code .
9. Test class should be annotated with @isTest .
10. @isTest annotation with test method is equivalent to testMethod keyword .
11. Test method should static and no void return type .
12. Test class and method default access is private ,no matter to add access specifier .
13. Classes with @isTest annotation can't be a interface or enum .
14. Test method code can't be invoked by non test request .
15. Stating with salesforce API 28.0 test method can not reside inside non test classes .
16. @Testvisible annotation to make visible private methods inside test classes.
17. Test method can't be used to test web-service call out . Please use call out mock .
18. You can't send email from test method.
19.User, profile, organization, AsyncApexjob, Corntrigger, RecordType, ApexClass, ApexComponent ,ApexPage we can access without (seeAllData=true) .
20. SeeAllData=true will not work for API 23 version eailer .
21. Accessing static resource test records in test class e,g List<Account> accList=Test.loadData(Account,SobjectType,'ResourceName').
22. Create TestFactory class with @isTest annotation to exclude from organization code size limit .
23. @testSetup to create test records once in a method and use in every test method in the test class .
24. We can run unit test by using Salesforce Standard UI,Force.com IDE ,Console ,API.
25. As apex runs in system mode so the permission and record sharing are not taken into account . So we need to use system.runAs to enforce record sharing .
26. System.runAs will not enforce user permission or field level permission .
27. Every test to runAs count against the total number of DML issued in the process .
==========

Unit Testing

Use a consistent naming convention including "Test" and the name of the class being tested (e.g., Test_AccountTrigger)
Test classes should use the @isTest annotation
Test methods should craete all data needed for the method and not rely on data currently in the Org.
Use System.assert liberally to prove that code behaves as expected.
Test each branch of conditional logic
Write test methods that both pass and fail for certain conditions and test for boundary conditions.
Test triggers to process 200 records - make sure your code is "bulkified" for 200 records and doesn't throw the dreaded "Too many SOQL queries: 21" exception.
When testing for governor limits, use Test.startTest and Test.stopTest and the Limit class instead of hard-coding governor limits.
Use System.runAs() to execute code as a specific user to test for sharing rules (but not CRUD or FLS permissions)
Execute tests with the Force.com IDE and not the salesforce.com UI. We've seen misleading code coverage results when running from the salesforce.com UI.
Run the Force.com Security Source Scanner to test your Org for a number of security and code quality issues (e.g., Cross Site Scripting, Access Control Issues, Frame Spoofing)

==========

TESTing:

- **Sandbox : Sample data**
- **Production : Live Data :**

Salesforce Testing Frameworking

- **Apex code : >= 75% code coverage**
- **Mock Data :**
  - **Excel Sheet**
  - **Mockaroo**
  - **Coding**
  - **....**
- **Define a class (Test Class)**
  - **Apex class :**
    - **Methods**

Salesforce options to run a particular method selected methods Entire class :

TEst class can be private even methods can be defined as private

Governor limit :

Apex Code

Testing 3MB

Annotations:

Apex annotations

```
//@isTest
public class MyContactTest{
    @testSetup static void prepareMock(){
        List<Contact> lstContacts = new List<Contact>();
        lstContacts.add( new Contact());
        lstContact.add(new Contact(lastName='sgdjsadgjsahsjdhjhakaj'));
    }

    testMethod static void  testMethod1 (){
        Database.insert(lstContacts, false);
    }
    testMethod static void  testMethod2 (){
    }
    testMethod static void  testMethod3 (){

    }
    testMethod static void  testMethod4 (){
    }
}
```