Apex Testing

(1) Get Started with Apex Unit Tests.

Create a Unit Test for a Simple Apex Class

Create and install a simple Apex class to test if a date is within a proper range, and if not, returns a date that occurs at the end of the month within the range. You'll copy the code for the class from GitHub. Then write unit tests that achieve 100% code coverage.

- Create an Apex class:
  - Name: `VerifyDate`
  - Code: Copy from GitHub
- Place the unit tests in a separate test class:
  - Name: `TestVerifyDate`
  - Goal: 100% code coverage
- Run your test class at least once

Steps taken to complete this challenge:

(1) Created an Apex class named as "VerifyDate.apxc".
(2) Code copied from GitHub link provided and written inside VerifyDate.apxc file:

```
public class VerifyDate {
    public static Date CheckDates(Date date1, Date date2) {
        if(DateWithin30Days(date1,date2)) {
            return date2;
        } else {
            return SetEndOfMonthDate(date1);
        }
    }
    private static Boolean DateWithin30Days(Date date1, Date date2) {
        if( date2 < date1 ) { return false; }
        Date date30Days = date1.addDays(30);
        if( date2 >= date30Days ) { return false; }
        else { return true; }
    }
     private static Date SetEndOfMonthDate(Date date1) {
        Integer totalDays = Date.daysInMonth(date1.year(), date1.month());
        Date lastDay = Date.newInstance(date1.year(), date1.month(), totalDays);
```

```
            return lastDay;
        }
    }
```

(3) Created a new Unit Test Apex class named as “TestVerifyDate.apxc”.
(4) Code written inside TestVerifyDate.apxc file:

```
@isTest
public class TestVerifyDate
{
   static testMethod void testMethod1()
   {
      Date d = VerifyDate.CheckDates(System.today(),System.today()+1);
      Date d1 = VerifyDate.CheckDates(System.today(),System.today()+60);
   }
}
```

(5) Save both the code in the Apex class files and run both of the Apex class files to obtained the result.

(2) Test Apex Triggers.

Create a Unit Test for a Simple Apex Trigger

Create and install a simple Apex trigger which blocks inserts and updates to any contact with a last name of 'INVALIDNAME'. You'll copy the code for the class from GitHub. Then write unit tests that achieve 100% code coverage.

- Create an Apex trigger on the Contact object
  - Name: `RestrictContactByName`
  - Code: **Copy from GitHub**
- Place the unit tests in a separate test class
  - Name: `TestRestrictContactByName`
  - Goal: 100% test coverage
- Run your test class at least once

Steps taken to complete this challenge:

(1) Created an Apex Trigger on Contact Object with a file name as “RestrictContactByName.apxt”.
(2) Code copied from GitHub link provided and written inside RestrictContactByName.apxt file:

```
trigger RestrictContactByName on Contact (before insert, before update) {
```

```
    //check contacts prior to insert or update for invalid data
    For (Contact c : Trigger.New) {
            if(c.LastName == 'INVALIDNAME') {        //invalidname is invalid
                    c.AddError('The Last Name "'+c.LastName+'" is not allowed for
DML');
            }
    }
}
```

(3) Created a new Unit Test Apex class named as “TestRestrictContactByName.apxc”.
(4) Code written inside TestRestrictContactByName.apxc file:

```
@isTest
private class TestRestrictContactByName {
   static testMethod void  metodoTest()
   {
      List<Contact> listContact= new List<Contact>();
      Contact c1 = new Contact(FirstName='Francesco', LastName='Riggio' ,
email='Test@test.com');
      Contact c2 = new Contact(FirstName='Francesco1', LastName =
'INVALIDNAME',email='Test@test.com');
      listContact.add(c1);
      listContact.add(c2);
      Test.startTest();
         try
         {
            insert listContact;
         }
         catch(Exception ee)
         {
         }
      Test.stopTest();
   }
}
```

(5) Save both the Apex code files and run both of the Apex files to obtained the result.

(3) Create Test Data for Apex Tests.

Create a Contact Test Factory

Create an Apex class that returns a list of contacts based on two incoming parameters: the number of contacts to generate and the last name. Do not insert the generated contact records into the database.

NOTE: For the purposes of verifying this hands-on challenge, don't specify the @isTest annotation for either the class or the method, even though it's usually required.

- Create an Apex class in the `public` scope
  - Name: `RandomContactFactory` (without the @isTest annotation)
- Use a Public Static Method to consistently generate contacts with unique first names based on the iterated number in the format Test 1, Test 2 and so on.
  - Method Name: `generateRandomContacts` (without the @isTest annotation)
  - Parameter 1: An integer that controls the number of contacts being generated with unique first names
  - Parameter 2: A string containing the last name of the contacts
  - Return Type: `List < Contact >`

Steps taken to complete this challenge:

(1) Created an Apex class named as “RandomContactFactory”.
(2) Code written inside RandomContactFactory.apxc file:

```
public class RandomContactFactory {

    public static List<Contact> generateRandomContacts(Integer numContactsToGenerate,
String FName) {

        List<Contact> contactList = new List<Contact>();

    for(Integer i=0;i<numContactsToGenerate;i++) {

            Contact c = new Contact(FirstName=FName + ' ' + i, LastName = 'Contact '+i);

            contactList.add(c);

            System.debug(c);

        }

        //insert contactList;

        System.debug(contactList.size());

        return contactList;

    }

}
```

(3) Save the above code written in the RandomContactFactory.apxc file and run the Apex code file two times to obtained the result.