1.Create an Apex trigger that sets an account's Shipping Postal Code to match the Billing Postal Code if the Match Billing Address option is selected. Fire the trigger before inserting an account or updating an account.

```
trigger AccountAddressTrigger on Account (before insert,before update) {
        for(Account a : Trigger.new) {
      if(a.Match_Billing_Address__c==True) {
        a.ShippingPostalCode = a.BillingPostalCode;
      }
   }
}
```

2.Create a Bulk Apex trigger
Create a bulkified Apex trigger that adds a follow-up task to an opportunity if its stage is Closed Won. Fire the Apex trigger after inserting or updating an opportunity.

```
trigger ClosedOppurtunityTrigger on Opportunity (after insert,after update) {
        List<Task> oppList = new List<Task>();
   for (Opportunity a : [SELECT Id,StageName,(SELECT WhatId,Subject FROM Tasks)
FROM Opportunity
                WHERE Id IN :Trigger.New AND StageName LIKE '%Closed Won%']) {
      oppList.add(new Task( WhatId=a.Id, Subject='Follow Up Test Task'));
   }
   if (oppList.size() > 0) {
      insert oppList;
   }
}
```

3.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.

```
public class VerifyDate {
        //method to handle potential checks against two dates
        public static Date CheckDates(Date date1, Date date2) {
                //if date2 is within the next 30 days of date1, use date2.  Otherwise use
```

```
the end of the month
                if(DateWithin30Days(date1,date2)) {
                        return date2;
                } else {
                        return SetEndOfMonthDate(date1);
                }
        }

        //method to check if date2 is within the next 30 days of date1
        private static Boolean DateWithin30Days(Date date1, Date date2) {
                //check for date2 being in the past
        if( date2 < date1) { return false; }
        //check that date2 is within (>=) 30 days of date1
        Date date30Days = date1.addDays(30); //create a date 30 days away from date1
                if( date2 >= date30Days ) { return false; }
                else { return true; }
        }

        //method to return the end of the month of a given date
        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;
        }
}


@isTest
public class TestVerifyDate {
        //testing that if date2 is within 30 days of date1, should return date 2
   @isTest static void testDate2within30daysofDate1() {
      Date date1 = date.newInstance(2018, 03, 20);
      Date date2 = date.newInstance(2018, 04, 11);
      Date resultDate = VerifyDate.CheckDates(date1,date2);
      Date testDate = Date.newInstance(2018, 04, 11);
      System.assertEquals(testDate,resultDate);
   }
```

```
    //testing that date2 is before date1. Should return "false"
    @isTest static void testDate2beforeDate1() {
        Date date1 = date.newInstance(2018, 03, 20);
        Date date2 = date.newInstance(2018, 02, 11);
        Date resultDate = VerifyDate.CheckDates(date1,date2);
        Date testDate = Date.newInstance(2018, 02, 11);
        System.assertNotEquals(testDate, resultDate);
    }
    //Test date2 is outside 30 days of date1. Should return end of month.
    @isTest static void testDate2outside30daysofDate1() {
        Date date1 = date.newInstance(2018, 03, 20);
        Date date2 = date.newInstance(2018, 04, 25);
        Date resultDate = VerifyDate.CheckDates(date1,date2);
        Date testDate = Date.newInstance(2018, 03, 31);
        System.assertEquals(testDate,resultDate);
    }
}
```

4.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.

```
trigger RestrictContactByName on Contact (before insert) {
        //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');
                }

        }
}

@isTest
public class TestRestrictContactByName {
         @isTest static void testInvalidName() {
```

```
        Contact myConact = new Contact(LastName='INVALIDNAME');
        insert myConact;
        Test.startTest();
        Database.SaveResult result = Database.insert(myConact, false);
        Test.stopTest();
        System.assert(!result.isSuccess());
        System.assert(result.getErrors().size() > 0);
        System.assertEquals('Cannot create contact with invalid last name.',
                            result.getErrors()[0].getMessage());
    }
}
```

5.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.

```
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;
    }

}
```

6.Create an Apex class that uses the @future annotation to update Account records. Create an Apex class with a future method that accepts a List of Account IDs and updates a custom field on the Account object with the number of contacts associated to the Account. Write unit tests that achieve 100% code coverage for the class. Every

hands-on challenge in this module asks you to create a test class.

```
public class AccountProcessor {
@future
  public static void countContacts(Set<id> setId)
  {
     List<Account> lstAccount = [select id,Number_of_Contacts__c , (select id from
contacts ) from account where id in :setId ];
     for( Account acc : lstAccount )
     {
        List<Contact> lstCont = acc.contacts ;
        acc.Number_of_Contacts__c = lstCont.size();
     }
     update lstAccount;
  }
}

@IsTest
public class AccountProcessorTest {
          public static testmethod void TestAccountProcessorTest(){
       Account a = new Account();
       a.Name = 'Test Account';
       Insert a;

       Contact cont = New Contact();
       cont.FirstName ='Bob';
       cont.LastName ='Masters';
       cont.AccountId = a.Id;
       Insert cont;
       set<Id> setAccId = new Set<ID>();
       setAccId.add(a.id);

       Test.startTest();
          AccountProcessor.countContacts(setAccId);
       Test.stopTest();
       Account ACC = [select Number_of_Contacts__c from Account where id = :a.id LIMIT
1];
```

```
        System.assertEquals ( Integer.valueOf(ACC.Number_of_Contacts__c) ,1);
          }
}
```

7.Create an Apex class that uses Batch Apex to update Lead records.
Create an Apex class that implements the Database.Batchable interface to update all Lead records in the org with a specific LeadSource.

```
global class LeadProcessor implements  Database.Batchable<Sobject> {
          global Database.QueryLocator start(Database.BatchableContext bc)
   {
      return Database.getQueryLocator([Select LeadSource From Lead ]);
   }

   global void execute(Database.BatchableContext bc, List<Lead> scope)
   {
          for (Lead Leads : scope)
          {
             Leads.LeadSource = 'Dreamforce';
          }
      update scope;
   }

   global void finish(Database.BatchableContext bc){   }
}

@isTest
public class LeadProcessorTest {
          static testMethod void testMethod1()
   {
      List<Lead> lstLead = new List<Lead>();
      for(Integer i=0 ;i <200;i++)
      {
         Lead led = new Lead();
         led.FirstName ='FirstName';
         led.LastName ='LastName'+i;
         led.Company ='demo'+i;
```

```
            lstLead.add(led);
        }
        insert lstLead;
        Test.startTest();

            LeadProcessor obj = new LeadProcessor();
            DataBase.executeBatch(obj);
        Test.stopTest();
    }
}
```

8.Create a Queueable Apex class that inserts Contacts for Accounts.
Create a Queueable Apex class that inserts the same Contact for each Account for a specific state.

```
public class AddPrimaryContact implements Queueable {
            private Contact c;
    private String state;
    public  AddPrimaryContact(Contact c, String state)
    {
        this.c = c;
        this.state = state;
    }
    public void execute(QueueableContext context)
    {
         List<Account> ListAccount = [SELECT ID, Name ,(Select id,FirstName,LastName
from contacts ) FROM ACCOUNT WHERE BillingState = :state LIMIT 200];
         List<Contact> lstContact = new List<Contact>();
         for (Account acc:ListAccount)
         {
                Contact cont = c.clone(false,false,false,false);
                cont.AccountId =  acc.id;
                lstContact.add( cont );
         }
         if(lstContact.size() >0 )
         {
            insert lstContact;
```

```
        }
    }
}

@isTest
public class AddPrimaryContactTest {
          @isTest static void TestList()
    {
        List<Account> Teste = new List <Account>();
        for(Integer i=0;i<50;i++)
        {
            Teste.add(new Account(BillingState = 'CA', name = 'Test'+i));
        }
        for(Integer j=0;j<50;j++)
        {
            Teste.add(new Account(BillingState = 'NY', name = 'Test'+j));
        }
        insert Teste;

        Contact co = new Contact();
        co.FirstName='demo';
        co.LastName ='demo';
        insert co;
        String state = 'CA';
         AddPrimaryContact apc = new AddPrimaryContact(co, state);
         Test.startTest();
           System.enqueueJob(apc);
         Test.stopTest();
     }
}
```

9.Create an Apex class that uses Scheduled Apex to update Lead records.
Create an Apex class that implements the Schedulable interface to update Lead records with a specific LeadSource.

```
global class DailyLeadProcessor implements Schedulable {
          global void execute(SchedulableContext ctx) {
```

```
    List<Lead> lList = [Select Id, LeadSource from Lead where LeadSource = null];
    if(!lList.isEmpty()) {
                    for(Lead l: lList) {
                        l.LeadSource = 'Dreamforce';
                    }
                    update lList;
                }
  }
}

@isTest
private class DailyLeadProcessorTest {
        static testMethod void testDailyLeadProcessor() {
                String CRON_EXP = '0 0 1 * * ?';
                List<Lead> lList = new List<Lead>();
          for (Integer i = 0; i < 200; i++) {
                    lList.add(new Lead(LastName='Dreamforce'+i, Company='Test1
Inc.', Status='Open - Not Contacted'));
                }
                insert lList;

                Test.startTest();
                String jobId = System.schedule('DailyLeadProcessor', CRON_EXP, new
DailyLeadProcessor());
        }
}
```

10.Create an Apex class that calls a REST endpoint and write a test class.
Create an Apex class that calls a REST endpoint to return the name of an animal, write unit tests that achieve 100% code coverage for the class using a mock response, and run your Apex tests.

```
public class AnimalLocator {
  public class Animal{
    public Integer id;
    public String name;
```

```
        public String eats;
        public String says;
    }
    public class AnimalResult{
        public Animal animal;
    }
            public static String getAnimalNameById(Integer id) {
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/' + id);
        request.setMethod('GET');
        HttpResponse response = http.send(request);
        AnimalResult result =
(AnimalResult)JSON.deserialize(response.getBody(),AnimalResult.class);
        return result.animal.name;
    }
}

@IsTest
public class AnimalLocatorTest {
            @isTest static void testPostCallout() {
    Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
     String expectedValue = AnimalLocator.getAnimalNameById(1);
      System.debug('expectedValue=' + expectedValue);

  }
}

@isTest
global class AnimalLocatorMock implements HttpCalloutMock {
   global HTTPResponse respond(HTTPRequest request) {
    HttpResponse response = new HttpResponse();
    response.setHeader('Content-Type', 'application/json');
    response.setBody('{ "animal": { "id": "5", "name": "Tiger",  "eats": "meat", "says": "roar"
}}');
    response.setStatusCode(200);
    return response;
```

```
 }
}
```

11.Generate an Apex class using WSDL2Apex for a SOAP web service, write unit tests that achieve 100% code coverage for the class using a mock response, and run your Apex tests.

```
//Generated by wsdl2apex

public class ParkService {
   public class byCountryResponse {
      public String[] return_x;
      private String[] return_x_type_info = new
String[]{'return','http://parks.services/',null,'0','-1','false'};
      private String[] apex_schema_type_info = new
String[]{'http://parks.services/','false','false'};
      private String[] field_order_type_info = new String[]{'return_x'};
   }
   public class byCountry {
      public String arg0;
      private String[] arg0_type_info = new
String[]{'arg0','http://parks.services/',null,'0','1','false'};
      private String[] apex_schema_type_info = new
String[]{'http://parks.services/','false','false'};
      private String[] field_order_type_info = new String[]{'arg0'};
   }
   public class ParksImplPort {
      public String endpoint_x = 'https://th-apex-soap-
service.herokuapp.com/service/parks';
      public Map<String,String> inputHttpHeaders_x;
      public Map<String,String> outputHttpHeaders_x;
      public String clientCertName_x;
      public String clientCert_x;
      public String clientCertPasswd_x;
      public Integer timeout_x;
      private String[] ns_map_type_info = new String[]{'http://parks.services/',
```

```
'ParkService'};
        public String[] byCountry(String arg0) {
            ParkService.byCountry request_x = new ParkService.byCountry();
            request_x.arg0 = arg0;
            ParkService.byCountryResponse response_x;
            Map<String, ParkService.byCountryResponse> response_map_x = new
Map<String, ParkService.byCountryResponse>();
            response_map_x.put('response_x', response_x);
            WebServiceCallout.invoke(
              this,
              request_x,
              response_map_x,
              new String[]{endpoint_x,
              '',
              'http://parks.services/',
              'byCountry',
              'http://parks.services/',
              'byCountryResponse',
              'ParkService.byCountryResponse'}
            );
            response_x = response_map_x.get('response_x');
            return response_x.return_x;
        }
    }
}

@isTest
global class ParkServiceMock implements WebServiceMock {
    global void doInvoke(
            Object stub,
            Object request,
            Map<String, Object> response,
            String endpoint,
            String soapAction,
            String requestName,
            String responseNS,
            String responseName,
```

```
        String responseType) {
      ParkService.byCountryResponse response_x = new
ParkService.byCountryResponse();
      List<String> lstOfDummyParks = new List<String> {'Park1','Park2','Park3'};
      response_x.return_x = lstOfDummyParks;
      response.put('response_x', response_x);
   }
}

public class ParkLocator {
   public static String[] country(String country){
      ParkService.ParksImplPort parks = new ParkService.ParksImplPort();
      String[] parksname = parks.byCountry(country);
      return parksname;
   }
}

@isTest
private class ParkLocatorTest{
   @isTest
   static void testParkLocator() {
      Test.setMock(WebServiceMock.class, new ParkServiceMock());
      String[] arrayOfParks = ParkLocator.country('India');
      System.assertEquals('Park1', arrayOfParks[0]);
   }
}
```

12.Create an Apex REST class that is accessible at /Accounts/<Account_ID>/contacts. The service will return the account's ID and name plus the ID and name of all contacts associated with the account. Write unit tests that achieve 100% code coverage for the class and run your Apex tests.

```
@RestResource(urlMapping='/Accounts/*/contacts')
global with sharing class AccountManager{
   @HttpGet
   global static Account getAccount(){
```

```
        RestRequest request = RestContext.request;
        String accountId = request.requestURI.substringBetween('Accounts/','/contacts');
        system.debug(accountId);
        Account objAccount = [SELECT Id,Name,(SELECT Id,Name FROM Contacts) FROM
Account WHERE Id = :accountId LIMIT 1];
        return objAccount;
    }
}

@isTest
private class AccountManagerTest{
    static testMethod void testMethod1(){
        Account objAccount = new Account(Name = 'test Account');
        insert objAccount;
        Contact objContact = new Contact(LastName = 'test Contact',
                                         AccountId = objAccount.Id);
        insert objContact;
        Id recordId = objAccount.Id;
        RestRequest request = new RestRequest();
        request.requestUri =
            'https://sandeepidentity-dev-ed.my.salesforce.com/services/apexrest/Accounts/'
            + recordId +'/contacts';
        request.httpMethod = 'GET';
        RestContext.request = request;
        Account thisAccount = AccountManager.getAccount();
        System.assert(thisAccount!= null);
        System.assertEquals('test Account', thisAccount.Name);
    }
}
```