Apex Specialist-Superbadge (Codes):

Apex Trigger (Module)

Get Started with Apex

AccountAddressTrigger

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

Bulk Apex Trigger

ClosedOpportunityTrigger

```
trigger ClosedOpportunityTrigger on Opportunity (after insert,after update) {
List<Task> tasklist=new List<Task>();
for(Opportunity opp:Trigger.New){
if(opp.StageName=='Closed Won'){
tasklist.add(new Task(Subject='Follow Up Test Task',WhatId=opp.Id));
}
}
if(tasklist.size()>0){
insert tasklist;
}
}
```

Apex Testing (Module)

Get started with Apex Unit Test

VerifyDate

```
public class VerifyDate {
        public static Date CheckDates(Date date1, Date date2) {
                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; }
        }

        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;
        }
}
```

TestVerifyDate

```
@isTest
private class TestVerifyDate {
   @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);
   }
   @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);
   }
   @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);
   }
}
```

Test Apex Trigger

RestrictContactByName

```
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');
                }
        }
}
```

TestRestrictContactByName

```
@isTest
private class TestRestrictContactByName {
   @isTest static void testInvalidName() {
      //try inserting a Contact with INVALIDNAME
      Contact myConact = new Contact(LastName='INVALIDNAME');
      insert myConact;

      // Perform test
      Test.startTest();
      Database.SaveResult result = Database.insert(myConact, false);
      Test.stopTest();
      // Verify
      // In this case the creation should have been stopped by the trigger,
      // so verify that we got back an error.
      System.assert(!result.isSuccess());
      System.assert(result.getErrors().size() > 0);
      System.assertEquals('Cannot create contact with invalid last name.',
                          result.getErrors()[0].getMessage());

   }
}
```

Create Test Data For Apex Tests

RandomContactFactory

```
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;
   }

```

```
}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;
   }

}
```

Asynchronous Apex (Module)

Use Future Methods Unit

AccountProcessor

```
public class AccountProcessor
{
  @future
  public static void countContacts(List<Id> accountIds)
  {
      List<Account> accountToUpdate=new List<Account>();
      List<Account> account = [select id,Number_Of_Contacts__c, (select id from contacts ) from
Account Where id in :accountIds];
      For( Account acc:account)
      {
          acc.Number_Of_Contacts__c=acc.Contacts.size();
      }
      update account;
  }
}
```

AccountProcessorTest

```
@isTest
public class AccountProcessorTest {
    public static testmethod void testAccountProcessor(){
        Account a=new Account();
        a.Name = 'Test Account';
        insert a;

        Contact con = new Contact();
        con.FirstName='Binary';
        con.LastName='Programming';
        con.AccountId=a.Id;
        insert con;

        List<Id> accListId=new List<Id>();
        accListId.add(a.Id);
        Test.startTest();
        AccountProcessor.countContacts(accListId);
        Test.stopTest();
        Account acc=[Select Number_Of_Contacts__c from Account where Id=:a.Id];
        System.assertEquals(Integer.valueOf(acc.Number_Of_Contacts__c),1);
    }
}
```

Use Batch Apex Unit

LeadProcessor

```
global class LeadProcessor implements Database.Batchable<sObject> {
    global Integer count=0;
    global Database.QueryLocator start(Database.BatchableContext bc){
        return Database.getQueryLocator('SELECT ID,LeadSource FROM Lead');
    }
    global void execute(Database.BatchableContext bc,List<Lead> L_list){
        List<lead> L_list_new=new List<lead>();
        for(lead L:L_list){
            L.leadsource='Dreamforce';
            L_list_new.add(L);
            count+=1;
        }
        update L_list_new;
    }
    global void finish(Database.BatchableContext bc){
        system.debug('count= '+count);
    }
}
```

LeadProcessorTest

```
@isTest
public class LeadProcessorTest {
    @isTest
```

```
  public static void testit(){
    List<lead> L_list=new List<lead>();
    for(Integer i=0;i<200;i++){
      Lead L=new lead();
      L.LastName='name'+i;
      L.Company='Company';
      L.Status='Random status';
      L_list.add(L);
    }
    insert L_list;
    Test.startTest();
    LeadProcessor lp=new LeadProcessor();
    Id batchId=Database.executeBatch(lp);
    Test.stopTest();
  }
}
```

Schedule Jobs Using the Apex Scheduler

DailyLeadProcessor

```
global class DailyLeadProcessor implements Schedulable {
  global void execute(SchedulableContext ctx){
    List<lead> leads = [Select Id,LeadSource FROM lead WHERE LeadSource=''];
    if(leads.size()>0){
      List<Lead> newLeads=new List<Lead>();
    for(Lead lead:leads){
      lead.LeadSource='Dreamforce';
      newLeads.add(lead);
    }
    update newLeads;
  }
  }

}
```

DailyLeadProcessorTest

```
@isTest
private class DailyLeadProcessorTest{
//Seconds Minutes Hours Day_of_month Month Day_of_week optional_year
public static String CRON_EXP = '0 0 0 2 6 ? 2022';

static testmethod void testScheduledJob(){
List<Lead> leads = new List<Lead>();

for(Integer i = 0; i < 200; i++){
Lead lead = new Lead(LastName = 'Test ' + i, LeadSource = '', Company = 'Test Company ' + i,
Status = 'Open - Not Contacted');
leads.add(lead);
}

insert leads;

Test.startTest();
// Schedule the test job
String jobId = System.schedule('Update LeadSource to DreamForce', CRON_EXP, new
DailyLeadProcessor());

// Stopping the test will run the job synchronously
Test.stopTest();
}
}
```

Control Processes with Queueable Apex

AddPrimaryContact

```
public class AddPrimaryContact implements Queueable {
   private Contact con;
   private String state;
   public AddPrimaryContact(Contact con,String state){
      this.con=con;
```

```
        this.state=state;
    }
    public void execute(QueueableContext context){
        List<Account> accounts=[Select Id,Name,(Select FirstName,LastName,Id from contacts)
                              from Account where BillingState= :state Limit 200];
        List<Contact> primaryContacts = new List<Contact>();
        for(Account acc:accounts){
            Contact c=con.clone();
            c.AccountId=acc.Id;
            primaryContacts.add(c);
        }
        if(primaryContacts.size()>0){
            insert primaryContacts;
        }
    }

}
```

AddPrimaryContactTest

```
@isTest
public class AddPrimaryContactTest {
    static testmethod void testQueueable(){
        List<Account> testAccounts=new List<Account>();
        for(Integer i=0;i<50;i++){
            testAccounts.add(new Account(Name='Account' +i,BillingState='CA'));
        }
         for(Integer j=0;j<50;j++){
            testAccounts.add(new Account(Name='Account' +j,BillingState='NY'));
        }
        insert testAccounts;
        Contact testContact = new Contact(FirstName='John',LastName='Doe');
        insert testContact;
        AddPrimaryContact addit=new addPrimaryContact(testContact,'CA');
        Test.startTest();
        system.enqueueJob(addit);
        Test.stopTest();
        system.assertEquals(50,[Select count() from Contact where accountId in (Select Id from
Account where BillingState='CA')]);
```

```
   }
}
```

<u>Apex Integration Services</u> (Module)

Apex REST Callouts

AnimalLocator

```
public class AnimalLocator
{

  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);
          String strResp = '';
           system.debug('******response '+response.getStatusCode());
           system.debug('******response '+response.getBody());
        // If the request is successful, parse the JSON response.
        if (response.getStatusCode() == 200)
        {
            // Deserializes the JSON string into collections of primitive data types.
           Map<String, Object> results = (Map<String, Object>)
JSON.deserializeUntyped(response.getBody());
            // Cast the values in the 'animals' key as a list
           Map<string,object> animals = (map<string,object>) results.get('animal');
            System.debug('Received the following animals:' + animals );
            strResp = string.valueof(animals.get('name'));
            System.debug('strResp >>>>>>' + strResp );
        }
        return strResp ;
   }

}
```

AnimalLocatorTest

```
@isTest
private class AnimalLocatorTest{
  @isTest static  void AnimalLocatorMock1() {
    Test.SetMock(HttpCallOutMock.class, new AnimalLocatorMock());
    String response=AnimalLocator.getAnimalNameById(1);
    System.assertEquals('chicken', response);
  }
}
```

Apex SOAP Callouts

parksServices

```
//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;
        }
    }
}
```

ParkLocator

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

ParkLocatorTest

```
@isTest
public class ParkLocatorTest {
@isTest static void testMock(){
      test.setMock(WebserviceMock.class, new ParkServiceMock());
      String[] parksName = ParkLocator.Country('India');
      List<String> country = new List<String>();
          country.add('Inamdar National Park');
            country.add('Riza National Park');
            country.add('Shilpa National Park');
      System.assertEquals(country, parksName, 'park names are not as expected');
   }

}
```

Apex Web Services

AccountManager

```
@RestResource (urlMapping = '/Account/*/contacts')
global with sharing class AccountManager
{
   @HttpGet
   global static Account getAccount ()
   {
      RestRequest request = RestContext.request;
      String accountId = request.requestURI.substringBetween('Accounts/','/contacts');
      Account result =  [SELECT Id, Name, (SELECT Id, Name FROM Contacts) FROM Account
WHERE Id = :accountId Limit 1];
      return result;
   }
}
```

AccountManagerTest

```
@isTest
private class AccountManagerTest
{
   @isTest static void testGetContactsByAccountId ()
```

```
  {
    Id recordId = createTestRecord ();
    RestRequest request = new RestRequest ();
    request.requestUri = 'https://yourInstance.salesforce.com/services/apexrest/Accounts/' +
recordId + '/contacts';
    request.httpMethod = 'GET';
    RestContext.request = request;
    Account thisAccount = AccountManager.getAccount();
    System.assert (thisAccount != null);
    System.assertEquals ('Test Record', thisAccount.Name);

  }

  static Id createTestRecord ()
  {
    Account accountTest = new Account (
      Name = 'Test Record');
    insert accountTest;
    Contact contactTest = new Contact (
      FirstName='John',
      LastName='Doe',
      AccountId =accountTest.Id);
    insert contactTest;
    return accountTest.Id;
  }

}
```