CODES FOR APEX TRIGGER SUPERBADGE:-

1. Create an Apex trigger

- **Name: Account Address Trigger**

```
trigger AccountAddressTrigger on Account (before insert, before update) {
for(Account account : Trigger.new){

      if((account.Match_Billing_Address__c == true) &&
(account.BillingPostalCode != NULL)){

          account.ShippingPostalCode = account.BillingPostalCode;

      }

   }

}
```

2. Create a Bulk Apex trigger

- **Name: Closed Opportunity Trigger**

```
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;

       }

}
```

3. Create a Unit Test for a Simple Apex Class

- **Name: Verify Date**

```
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;

        }

}
```

4. Create a Unit Test for a Simple Apex Trigger

- **Name: Restrict Contact By Name**

```
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');

                }

        }

}
```

5. Create a Contact Test Factory

```
public class RandomContactFactory

{

        public static List<Contact> generateRandomContacts(integer
numofContacts,string LastNameGen)

   {

        List<Contact> con= new List<Contact>();
```

```
            for(Integer i=0;i<numofContacts;i++)

            {

            LastNameGen='Test'+ i;

            Contact a=new
Contact(FirstName=LastNameGen,LastName=LastNameGen);

             con.add(a);

          }

          return con;

       }

}
```

6.. Create an Apex class that uses the @future annotation to update Account records.

- **Name: Account Processor**

```
public class AccountProcessor {

   @future

   public static void countContacts(List<Id> accountIds){

      List<Account> accounts = [Select Id, Name from Account Where Id IN :
accountIds];

      List<Account> updateAccounts = new List<Account>();

      for(Account account : accounts){

         account.Number_of_Contacts__c = [Select count() from Contact Where
AccountId =: account.Id];

         System.debug('No of Contacts = ' + account.Number_of_Contacts__c);

         updateAccounts.add(account);

      }
```

```
        update updateAccounts;

    }

}
```

7. Create an Apex test class:

- **Name: Account Processor Test**

```
@isTest

public class AccountProcessorTest {



    @isTest

    public static void testNoOfContacts(){

        Account a = new Account();

        a.Name = 'Test Account';

        Insert a;
```

```
Contact c = new Contact();

c.FirstName = 'Bob';

c.LastName = 'Willie';

c.AccountId = a.Id;



Contact c2 = new Contact();

c2.FirstName = 'Tom';

c2.LastName = 'Cruise';

c2.AccountId = a.Id;



List<Id> acctIds = new List<Id>();

acctIds.add(a.Id);



Test.startTest();

AccountProcessor.countContacts(acctIds);
```

```
        Test.stopTest();

           }

}
```

8. Create an Apex class that uses Batch Apex to update Lead records.

- **Name: Lead Processor**

```
global class LeadProcessor implements

Database.Batchable<sObject>, Database.Stateful {



    // instance member to retain state across transactions

    global Integer recordsProcessed = 0;



    global Database.QueryLocator start(Database.BatchableContext bc) {

        return Database.getQueryLocator('SELECT Id, LeadSource FROM Lead');

    }
```

```
global void execute(Database.BatchableContext bc, List<Lead> scope){

    // process each batch of records

    List<Lead> leads = new List<Lead>();

    for (Lead lead : scope) {



            lead.LeadSource = 'Dreamforce';

            // increment the instance member counter

            recordsProcessed = recordsProcessed + 1;



    }

    update leads;

}



global void finish(Database.BatchableContext bc){
        System.debug(recordsProcessed + ' records processed. Shazam!');
           }
}
```

- <u>**Name: Lead Processor Test**</u>

```
@isTest
public class LeadProcessorTest {
 @testSetup
    static void setup() {
        List<Lead> leads = new List<Lead>();
        // insert 200 leads
        for (Integer i=0;i<200;i++) {
            leads.add(new Lead(LastName='Lead '+i,
                Company='Lead', Status='Open - Not Contacted'));
        }
        insert leads;

    }



    static testmethod void test() {

        Test.startTest();

        LeadProcessor lp = new LeadProcessor();

        Id batchId = Database.executeBatch(lp, 200);

        Test.stopTest();



        // after the testing stops, assert records were updated properly

        System.assertEquals(200, [select count() from lead where LeadSource =
'Dreamforce']);

    }

}
```

9. Create a Queueable Apex class that inserts Contacts for Accounts.

- **Name: Add Primary Contact**

```
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;

         }

   }
```

```
}
```

- <u>**Name: Add Primary Contact Test**</u>

```
@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();

}
```

```
 }
```

10. Create an Apex class that uses Scheduled Apex to update Lead records.

- **Name: Daily Lead Processor**

```
global class DailyLeadProcessor implements Schedulable {



    global void execute(SchedulableContext ctx) {

        List<Lead> lList = [Select Id, LeadSource from Lead where LeadSource = null
limit 200];

        list<lead> led = new list<lead>();

        if(!lList.isEmpty()) {

            for(Lead l: lList) {

                l.LeadSource = 'Dreamforce';

                led.add(l);

            }

            update led;
        }
    }
}
```

- **Name: Daily Lead Processor Test**

```
@isTest
public class DailyLeadProcessorTest{


    static testMethod void testMethod1()
    {
 Test.startTest
 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;

            DailyLeadProcessor ab = new DailyLeadProcessor();

         String jobId = System.schedule('jobName','0 5 * * * ? ' ,ab) ;

            Test.stopTest();

      }

   }
```

11. Create an Apex class that calls a REST endpoint and write a test class.

- **Name: Animal Locator**

```
public class AnimalLocator{

    public static String getAnimalNameById(Integer x){

        Http http = new Http();

        HttpRequest req = new HttpRequest();

        req.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/' + x);

        req.setMethod('GET');

        Map<String, Object> animal= new Map<String, Object>();

        HttpResponse res = http.send(req);

        if(res.getStatusCode() == 200) {

            Map<String, Object> results = (Map<String,
Object>)JSON.deserializeUntyped(res.getBody());

            animal = (Map<String, Object>) results.get('animal');

        }
        return (String)animal.get('name');

        }

    }
```

- **Name: Animal Locator Test**

```
@isTest

private class AnimalLocatorTest{

    @isTest static void AnimalLocatorMock1() {

        Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());

        string result = AnimalLocator.getAnimalNameById(3);

        String expectedResult = 'chicken';

        System.assertEquals(result,expectedResult);

    }

}
```

<u>12. Generate an Apex class using WSDL2Apex and write a test class.</u>

- **<u>Name: Park Locator</u>**

```
public class ParkLocator {

    public static string[] country(String country) {

        parkService.parksImplPort park = new parkService.parksImplPort();

        return park.byCountry(country);

    }

}
```

- **<u>Name: Park Locator Test</u>**

```
@isTest

private class ParkLocatorTest {

    @isTest static void testCallout() {

        // This causes a fake response to be generated

        Test.setMock(WebServiceMock.class, new ParkServiceMock());
        // Call the method that invokes a callout

        //Double x = 1.0;

        //Double result = AwesomeCalculator.add(x, y);



        String country = 'Germany';

        String[] result = ParkLocator.Country(country);



            // Verify that a fake result is returned

        System.assertEquals(new List<String>{'Hamburg Wadden Sea National
Park', 'Hainich National Park', 'Bavarian Forest National Park'}, result);

    }

}
```

13. Create an Apex REST service that returns an account and its contacts.

- **Name: Account Manager**

```
@RestResource(urlMapping='/Accounts/*/contacts')

global with sharing class AccountManager{

   @HttpGet

   global static Account getAccount(){

      RestRequest req = RestContext.request;

      String accId = req.requestURI.substringBetween('Accounts/', '/contacts');

      Account acc = [SELECT Id, Name, (SELECT Id, Name FROM Contacts)

                  FROM Account WHERE Id = :accId];



      return acc;

   }
}
```

- <u>AccountManagerTest</u>

```
@IsTest

private class AccountManagerTest{

    @isTest static void testAccountManager(){

        Id recordId = getTestAccountId();

        // Set up a test request

        RestRequest request = new RestRequest();

        request.requestUri =

            'https://ap5.salesforce.com/services/apexrest/Accounts/'+ recordId
+'/contacts';

        request.httpMethod = 'GET';

        RestContext.request = request;



        // Call the method to test
    }

        Account  acc = AccountManager.getAccount();



        // Verify results

        System.assert(acc != null);

    }



    private static Id getTestAccountId(){

        Account acc = new Account(Name = 'TestAcc2');

        Insert acc;



        Contact con = new Contact(LastName = 'TestCont2', AccountId = acc.Id);

        Insert con;



        return acc.Id;

    }
```

```
  // Helper method

    static Id createTestRecord() {

        // Create test record

        Account TestAcc = new Account(

          Name='Test record');

        insert TestAcc;

        Contact TestCon= new Contact(

        LastName= 'Test',

        AccountId = TestAcc.id);

        return TestAcc.Id;

    }

}
```