APEX TRIGGER:

Challenge 1:

Create an Apex trigger for Account that matches Shipping Address Postal Code with Billing Address Postal Code based on a custom field.

```
trigger AccountAddressTrigger on Account (before insert, before update) {
    for(Account a: Trigger.New){
        if(a.Match_Billing_Address__c == true && a.BillingPostalCode!= null){
            a.ShippingPostalCode=a.BillingPostalCode;
        }
    }

}
```

Challenge 2:

```
trigger ClosedOpportunityTrigger on Opportunity (after insert, after update) {
    List<Task> taskList = new List<Task>();
    //first way
    for(Opportunity opp : [SELECT Id, StageName FROM Opportunity WHERE StageName='Closed
Won' AND Id IN : Trigger.New]){
        taskList.add(new Task(Subject='Follow Up Test Task', WhatId = opp.Id));
    }

    //second way and we should use this
    /*
     for(opportunity opp: Trigger.New){

        if(opp.StageName!=trigger.oldMap.get(opp.id).stageName)
        {

            taskList.add(new Task(Subject = 'Follow Up Test Task',
                                  WhatId = opp.Id));
        }

    }
          */

    if(taskList.size()>0){
```

```
        insert tasklist;
    }

}
```

APEX TESTING:

Challange1:

```
@isTest
private 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);
    }
}
```

Challenge 2:

Create an Apex trigger for Opportunity that adds a task to any opportunity set to 'Closed Won'.

To complete this challenge, you need to add a trigger for Opportunity. The trigger will add a task to any opportunity inserted or updated with the stage of 'Closed Won'. The task's subject must be 'Follow Up Test Task'.

The Apex trigger must be called 'ClosedOpportunityTrigger'

With 'ClosedOpportunityTrigger' active, if an opportunity is inserted or updated with a stage of 'Closed Won', it will have a task created with the subject 'Follow Up Test Task'.

To associate the task with the opportunity, fill the 'WhatId' field with the opportunity ID.

This challenge specifically tests 200 records in one operation.

Solution:

```
trigger ClosedOpportunityTrigger on Opportunity (after insert, after update) {
    List<Task> taskList = new List<Task>();
    //first way
    for(Opportunity opp : [SELECT Id, StageName FROM Opportunity WHERE StageName='Closed Won' AND Id IN : Trigger.New]){
        taskList.add(new Task(Subject='Follow Up Test Task', WhatId = opp.Id));
    }
```

```
    //second way and we should use this
    /*
     for(opportunity opp: Trigger.New){

       if(opp.StageName!=trigger.oldMap.get(opp.id).stageName)
       {

          taskList.add(new Task(Subject = 'Follow Up Test Task',
                                WhatId = opp.Id));
       }

    }
           */

    if(taskList.size()>0){
       insert tasklist;
    }

}
```

Challenge 3:

```
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);
```

```
    }

    System.debug(contactList.size());
    return contactList;
  }

}
```

Asynchronous Apex:

AccountProcessor.cls:

```
public class AccountProcessor {
  @future
  public static void countContacts(List<Id> accountId_lst) {
    Map<Id,Integer> account_cno = new Map<Id,Integer>();
    List<account> account_lst_all = new List<account>([select id, (select id from contacts) from account]);
    for(account a:account_lst_all) {
      account_cno.put(a.id,a.contacts.size()); //populate the map
    }

    List<account> account_lst = new List<account>(); // list of account that we will upsert

    for(Id accountId : accountId_lst) {
      if(account_cno.containsKey(accountId)) {
        account acc = new account();
        acc.Id = accountId;
        acc.Number_of_Contacts__c = account_cno.get(accountId);
```

```
            account_lst.add(acc);
        }
    }
    upsert account_lst;
  }

}
```

AccountProcessorTest.cls:

```
@isTest
public class AccountProcessorTest {
  @isTest
  public static void testFunc() {
    account acc = new account();
    acc.name = 'MATW INC';
    insert acc;
    contact con = new contact();
    con.lastname = 'Mann1';
    con.AccountId = acc.Id;
    insert con;
    contact con1 = new contact();
    con1.lastname = 'Mann2';
    con1.AccountId = acc.Id;
    insert con1;
    List<Id> acc_list = new List<Id>();
    acc_list.add(acc.Id);
    Test.startTest();
      AccountProcessor.countContacts(acc_list);
```

```
        Test.stopTest();
        List<account> acc1 = new List<account>([select Number_of_Contacts__c from
account where id = :acc.id]);
        system.assertEquals(2,acc1[0].Number_of_Contacts__c);
    }

}
```

USing Batch Apex:

Apex Class:

```
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;
        }
    }
}
```

Apex Test Class:

```
@isTest
private class DailyLeadProcessorTest{

   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();

      String jobId = System.schedule('Update LeadSource to DreamForce', CRON_EXP, new DailyLeadProcessor());
      Test.stopTest();
   }
}
```

Control Processes With Queueable Apex:

1.AddPrimaryContact.Apxc

```
public class AddPrimaryContact implements Queueable {
    public contact c;
    public String state;

    public AddPrimaryContact(Contact c, String state) {
        this.c = c;
        this.state = state;
    }

    public void execute(QueueableContext qc) {
        system.debug('this.c = '+this.c+' this.state = '+this.state);
        List<Account> acc_lst = new List<account>([select id, name, BillingState from
account where account.BillingState = :this.state limit 200]);
        List<contact> c_lst = new List<contact>();
        for(account a: acc_lst) {
            contact c = new contact();
            c = this.c.clone(false, false, false, false);
            c.AccountId = a.Id;
            c_lst.add(c);
        }
        insert c_lst;
    }

}
```

2.AddPrimaryContactTest.apxc

```
@isTest
```

```
public class AddPrimaryContactTest {

    @testSetup
    public static void setup(){
        List<account> acc_lst = new List<account>();
        for (Integer i=0; i<50;i++) {
            account a = new account(name=string.valueOf(i),billingstate='NY');
            system.debug('account a = '+a);
            acc_lst.add(a);
        }
        for (Integer i=0; i<50;i++) {
            account a = new account(name=string.valueOf(50+i),billingstate='CA');
            system.debug('account a = '+a);
            acc_lst.add(a);
        }
        insert acc_lst;
    }

    public static testMethod void TestQueueable(){
        List<Account> ac_ca=[select id from Account where billingstate='CA'];


        contact c = new contact(lastname='bhau');
        AddPrimaryContact apc = new AddPrimaryContact(c,'CA');

        Test.startTest();
        System.enqueueJob(apc);
```

```
        Test.stopTest();



        system.assertEquals(50, [select count() from contact where AccountId IN :ac_ca]);
    }
}
```

Schedule Job Using the Apex Scheduler:

Apex Class:

```
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;
        }
    }
}
```

```
```

Apex Test Class

```
@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();

      String jobId = System.schedule('Update LeadSource to DreamForce', CRON_EXP, new DailyLeadProcessor());

      Test.stopTest();
   }
```

```
}
```

Apex Integration Service:

AnimalLocator.cls:

```
public class AnimalLocator {
        public class cls_animal {
                public Integer id;
                public String name;
                public String eats;
                public String says;
        }
public class JSONOutput{
        public cls_animal animal;

        //public JSONOutput parse(String json){
        //return (JSONOutput) System.JSON.deserialize(json, JSONOutput.class);
        //}
}
   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.setHeader('id', String.valueof(id)); -- cannot be used in this challenge :)
      request.setMethod('GET');
      HttpResponse response = http.send(request);
      system.debug('response: ' + response.getBody());
      //Map<String,Object> map_results = (Map<String,Object>)
```

```
JSON.deserializeUntyped(response.getBody());
      jsonOutput results = (jsonOutput) JSON.deserialize(response.getBody(),
jsonOutput.class);
      //Object results = (Object) map_results.get('animal');
                  system.debug('results= ' + results.animal.name);
      return(results.animal.name);
   }

}
```

AnimalLocatorTest.cls:

```
@IsTest
public class AnimalLocatorTest {
   @isTest
   public static void testAnimalLocator() {
      Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
      //Httpresponse response = AnimalLocator.getAnimalNameById(1);
      String s =  AnimalLocator.getAnimalNameById(1);
      system.debug('string returned: ' + s);
   }

}
```

AnimalLocatorMock:

```
@IsTest
global class AnimalLocatorMock implements HttpCalloutMock {
   global HTTPresponse respond(HTTPrequest request) {
      Httpresponse response = new Httpresponse();
      response.setStatusCode(200);
```

```
        //-- directly output the JSON, instead of creating a logic
        //response.setHeader('key, value)
        //Integer id = Integer.valueof(request.getHeader('id'));
        //Integer id = 1;
        //List<String> lst_body = new List<String> {'majestic badger', 'fluffy bunny'};
        //system.debug('animal return value: ' + lst_body[id]);
        response.setBody('{"animal":{"id":1,"name":"chicken","eats":"chicken food","says":"cluck cluck"}}');
        return response;
    }

}
```

Apex Soap Callouts:

Apex Class

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

Apex Test Class

```

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

        System.assertEquals('Park1', arrayOfParks[0]);
    }
}

```

Apex Mock Test Class

```

@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);
  }
}
```

Apex Web Services:

Apex Class

```
@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;
    }
}
```

Apex Test Class

```
@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;
    }
}
```