Apex Triggers

```
//AccountAddressTrigger.aptx
trigger AccountAddressTrigger on Account (before insert, before update) {

   for(Account a : Trigger.new) {

      if(a.Match_Billing_Address__c && a.BillingPostalCode != null) {

         a.ShippingPostalCode = a.BillingPostalCode;
      }
   }
}
//ClosedOpportunityTrigger.aptx
trigger ClosedOpportunityTrigger 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;
   }
}
```

Apex Testing

```
//TestRestrictContactByName.apxc
@isTest
private 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());

    }
}
//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);
    }
}
//RandomContactFactory.apxc
public class RandomContactFactory {


   Public Static List<Contact> generateRandomContacts(integer noOfContact, String lastName)
   {
       List<Contact> con=New list<Contact>();
       for(Integer i=0;i<noOfContact;i++)
       {
            Contact c = new Contact(FirstName='Ank' + i,LastName=lastName);
            Con.add(c);
        }
        Return con;
    }


}
```

Asynchronous Apex

```
//AccountProcessor.apxc

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;

  }

}

//AccountProcessorTest.apxc

@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);
  }

}
//LeadProcessor.apxc
global class LeadProcessor implements
Database.Batchable<sObject>, Database.Stateful {

    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){
        List<Lead> leads = new List<Lead>();
        for (Lead lead : scope) {

                lead.LeadSource = 'Dreamforce';
                recordsProcessed = recordsProcessed + 1;

        }
        update leads;
    }

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

    }
}
//LeadProcessorTest.apxc
global class LeadProcessor implements
Database.Batchable<sObject>, Database.Stateful {

    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){
        List<Lead> leads = new List<Lead>();
        for (Lead lead : scope) {

                lead.LeadSource = 'Dreamforce';
```

```
            recordsProcessed = recordsProcessed + 1;

        }
        update leads;
    }

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

    }
}
//AddPrimaryContact.apxc
public class AddPrimaryContact implements Queueable{
    Contact con;
    String state;

    public AddPrimaryContact(Contact con, String state){
        this.con = con;
        this.state = state;
    }
    public void execute(QueueableContext qc){
        List<Account> lstOfAccs = [SELECT Id FROM Account WHERE BillingState = :state LIMIT 200];

        List<Contact> lstOfConts = new List<Contact>();
        for(Account acc : lstOfAccs){
            Contact conInst = con.clone(false,false,false,false);
            conInst.AccountId = acc.Id;

            lstOfConts.add(conInst);
        }

        INSERT lstOfConts;
    }
}
//AddPrimaryContactTest.apxc
@isTest
public class AddPrimaryContactTest{
    @testSetup
    static void setup(){
        List<Account> lstOfAcc = new List<Account>();
        for(Integer i = 1; i <= 100; i++){
            if(i <= 50)
                lstOfAcc.add(new Account(name='AC'+i, BillingState = 'NY'));
            else
                lstOfAcc.add(new Account(name='AC'+i, BillingState = 'CA'));
        }

        INSERT lstOfAcc;
    }

    static testmethod void testAddPrimaryContact(){
        Contact con = new Contact(LastName = 'TestCont');
        AddPrimaryContact addPCIns = new AddPrimaryContact(CON ,'CA');

        Test.startTest();
        System.enqueueJob(addPCIns);
        Test.stopTest();
        System.assertEquals(50, [select count() from Contact]);
    }
}
```

//DailyLeadProcessor.apxc

```
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.apxc

```
@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();
    }
}
```

Apex Integration Services

```
//AnimalLocator.apxc

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 (response.getStatusCode() == 200)

        {.

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

           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 ;

   }



}
```

```
//AnimalLocatorMock.apxc
@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":1,"name":"chicken","eats":"chicken food","says":"cluck cluck"}}');
        response.setStatusCode(200);
        return response;
    }
}
//AnimalLocatorTest.apxc
@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);
    }
}
//ParkLocator.apxc
public class ParkLocator {
    public static String[] country(String country){
        ParkService.ParksImplPort parks = new ParkService.ParksImplPort();
        String[] parksname = parks.byCountry(country);
        return parksname;
    }
}
```

//ParkService.apxc

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


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