Module: Apex Triggers
Get Started with Apex Triggers:
AccountAddressTrigger.apxt

```
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 Triggers:
ClosedOpportunityTrigger.apxt

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

Module: Apex Testing
Get Started with Apex Unit Tests:
1) VerifyDate.apxc

```
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
@TestVisible 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
@TestVisible 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;
}
}
```

2) TestVerifyDate.apxc

```
@isTest
private class TestVerifyDate {
   @isTest static void Test_CheckDates_case1(){
      Date D =
VerifyDate.CheckDates(date.parse('01/01/2020'),date.parse('01/05/2020'));
      System.assertEquals(date.parse('01/05/2020'),D);
   }
@isTest static void Test_CheckDates_case2(){
      Date D =
VerifyDate.CheckDates(date.parse('01/01/2020'),date.parse('05/05/2020'));
      System.assertEquals(date.parse('01/31/2020'),D);
   }
   @isTest static void Test_DateWithin30Days_case1(){
      Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
date.parse('12/30/2019'));
      System.assertEquals(false, flag);
```

```
    }
     @isTest static void Test_DateWithin30Days_case2(){
        Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
date.parse('02/02/2019'));
        System.assertEquals(false, flag);
     }
     @isTest static void Test_DateWithin30Days_case3(){
        Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
date.parse('01/15/2020'));
        System.assertEquals(true, flag);
     }
    @isTest static void Test_setEndOfMonthDate(){
        Date returndate = VerifyDate.SetEndOfMonthDate(date.parse('01/01/2020'));
    }
}Test Apex Triggers:
1) RestrictContactByName.apxt
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');
}
}
}
2) TestRestrictContactByName.apxc
@isTest
public class TestRestrictContactByName {
    @isTest static void Test_insertupdateContact()
    {
        Contact cnt = new Contact();
        cnt.LastName = 'INVALIDNAME';
        Test.startTest();
        Database.SaveResult result = Database.insert(cnt, false);
        Test.stopTest();
        System.assert(!result.isSuccess());
        System.assert(result.getErrors().size()>0);
```

```
        System.assertEquals('The Last Name "INVALIDNAME" is not allowed for DML',
result.getErrors()[0].getMessage());
    }
}
```

Create Test Data for Apex Tests:

RandomContactFactory .apxc

```
public class RandomContactFactory {    public static List<Contact>
generateRandomContacts(Integer numcnt, string
lastname){
        List<Contact> contacts = new List<Contact>();
        for(Integer i=0;i<numcnt;i++)
        {
            Contact cnt = new Contact(FirstName = 'Test '+i, LastName = lastname);
            contacts.add(cnt);
        }
        return contacts;
    }
}
```

Module: Asynchronous Apex

Use Future Methods:

1) AccountProcessor.apxc

```
public class AccountProcessor {
    @future
    public static void countContacts(List<ID> accountIds) {
        List<Account> accountsToUpdate = new List<Account>();
        List<Account> accounts = [Select Id, Name, (Select Id from Contacts) from Account
Where Id in :accountIds];
        For(Account acc:accounts){
            List<Contact> contactList = acc.Contacts;
            acc.Number_Of_Contacts__c = contactList.size();
            accountsToUpdate.add(acc);
        }
        update accountsToUpdate;
    }
}
```

2) AccountProcessorTest.apxc

```
@IsTest
```

```
private class AccountProcessorTest {
   @IsTest
   private static void testCountContacts(){
      Account newAccount = new Account(Name='Test Account');
      insert newAccount;
      Contact newContact1 = new Contact(FirstName='John', LastName='Doe', AccountId
= newAccount.Id);
      insert newContact1;
       Contact newContact2 = new Contact(FirstName='Jane', LastName='Doe',
AccountId = newAccount.Id);
      insert newContact2;
      List<Id> accountIds = new List<ID>();
      accountIds.add(newAccount.Id);
      Test.startTest();
      AccountProcessor.countContacts(accountIds);
      Test.stopTest();
   }
}
```

Use Batch Apex:

1) LeadProcessor.apxc

```
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);
    }
}
```

2) LeadProcessorTest.apxc

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

Control Processes with Queueable Apex:

1) AddPrimaryContact.apxc

```
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;
      }
   }
      }
```

2)AddPrimaryContactTest.apxc

```
@isTest
public class AddPrimaryContactTest {
static testmethod void testQueuable()
{
   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')]);
}
}
```

Schedule Jobs Using the Apex Scheduler:

1) DailyLeadProcessor.apxc

```
public class DailyLeadProcessor implements Schedulable  {
    Public void execute(SchedulableContext SC){
      List<Lead> LeadObj=[SELECT Id from Lead where LeadSource=null limit 200];
       for(Lead l:LeadObj){
          l.LeadSource='Dreamforce';
          update l;
       }
    }
}
}
```

2) DailyLeadProcessorTest.apxc

```
@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('DailyLeadProcessorTest',
CRON_EXP, new DailyLeadProcessor());
}
}
```

Module: Apex Integration Services

Apex REST Callouts:

1) AnimalLocator.apxc

```
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');
   }
}
2) 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 );
   }
}3) AnimalLocatorMock.apxc
@isTest
global class AnimalLocatorMock implements HttpCalloutMock {
// Implement this interface method
global HTTPResponse respond(HTTPRequest request) {
// Create a fake response
HttpResponse response = new HttpResponse();
response.setHeader('Content-Type', 'application/json');
response.setBody('{"animals": ["majestic badger", "fluffy bunny", "scary bear", "chicken",
"mighty moose"]}');
response.setStatusCode(200);
return response;
}
```

```
}
```
Apex SOAP Callouts:
1)ParkLocator.apxc
```
public class ParkLocator {
public static string[] country(string theCountry) {
ParkService.ParksImplPort parkSvc = new ParkService.ParksImplPort(); // remove
space
return parkSvc.byCountry(theCountry);
}
}
```
2)ParkLocatorTest.apxc
```
@isTest
private class ParkLocatorTest {
@isTest static void testCallout() {
Test.setMock(WebServiceMock.class, new ParkServiceMock ());
String country = 'United States';
List<String> result = ParkLocator.country(country);
List<String> parks = new List<String>{'Yellowstone', 'Mackinac National Park',
'Yosemite'};
System.assertEquals(parks, result);
}
}3)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) {
// start - specify the response you want to send
ParkService.byCountryResponse response_x = new ParkService.byCountryResponse();
response_x.return_x = new List<String>{'Yellowstone', 'Mackinac National Park',
```

```
'Yosemite'};
// end
response.put('response_x', response_x);
}
}
4)AsyncParkService.apxc
public class AsyncParkService {
public class byCountryResponseFuture extends System.WebServiceCalloutFuture {
public String[] getValue() {
ParkService.byCountryResponse response =
(ParkService.byCountryResponse)System.WebServiceCallout.endInvoke(this);
return response.return_x;
}
}
public class AsyncParksImplPort {
public String endpoint_x = 'https://th-apex-soap-service.herokuapp.com/service/parks';
public Map<String,String> inputHttpHeaders_x;
public String clientCertName_x;
public Integer timeout_x;
private String[] ns_map_type_info = new String[]{'http://parks.services/', 'ParkService'};
public AsyncParkService.byCountryResponseFuture
beginByCountry(System.Continuation
continuation,String arg0) {
ParkService.byCountry request_x = new ParkService.byCountry();request_x.arg0 = arg0;
return (AsyncParkService.byCountryResponseFuture)
System.WebServiceCallout.beginInvoke(
this,
request_x,
AsyncParkService.byCountryResponseFuture.class,
continuation,
new String[]{endpoint_x,
'',
'http://parks.services/',
'byCountry',
'http://parks.services/',
'byCountryResponse',
'ParkService.byCountryResponse'}
```

```
);
}
}
}
```
Apex Web Services:
1)AccountManager.apxc
```
@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');
Account result = [SELECT Id, Name, (Select Id, Name from Contacts) from Account
where
Id=:accountId Limit 1];
return result;
}
```
}2)AccountManagerTest.apxc
```
@IsTest
private class AccountManagerTest {
@isTest static void testGetContactsByAccountId(){
Id recordId = createTestRecord();
RestRequest request = new RestRequest();
request.requestUri =
'https://yourInstance.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 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;
}
```