**Apex Triggers**

**Challenge-1**

[**Get Started with Apex Triggers**](https://trailhead.salesforce.com/content/learn/modules/apex_triggers/apex_triggers_intro?trailmix_creator_id=trailblazerconnect&trailmix_slug=salesforce-developer-catalyst)

**AccountAddressTrigger.apxc:**

trigger AccountAddressTrigger on Account (before insert,before update) {

for(Account account:Trigger.new){

if((account.Match\_Billing\_Address\_\_c == true) && (account.BillingPostalCode!=NULL))

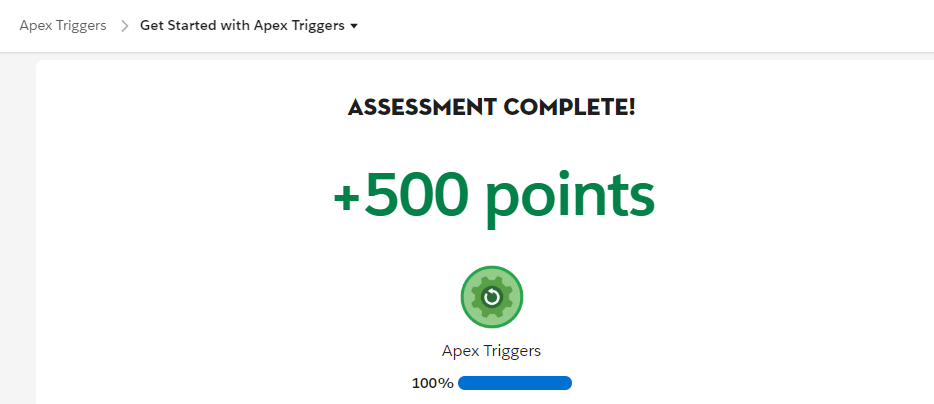
{

account.ShippingPostalCode=account.BillingPostalCode;

}

}

}

****

**Challenge-2**

**Bulk Apex Triggers**

**ClosedOpportunityTrigger.apxc:**

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));

}

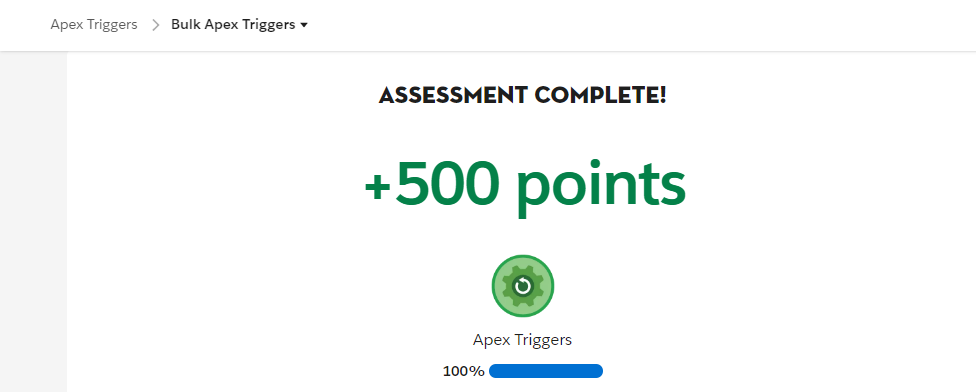
}

if(taskList.size()>0){

insert taskList;

}

}



**Apex Testing**

**Challenge-1**

**Get Started with Apex Unit Tests**

**VerifyDate.apxc:**

public class VerifyDate {

public static Date CheckDates(Date date1, Date date2) {

if(DateWithin30Days(date1,date2)) {

return date2;

}

else {

return SetEndOfMonthDate(date1);

}

}

private static Boolean DateWithin30Days(Date date1, Date date2) {

if( date2 < date1) { return false; }

Date date30Days = date1.addDays(30);

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.apxc:**

@isTest

public class TestVerifyDate {

@isTest static void test1(){

Date d=verifyDate.CheckDates(Date.parse('01/01/2021'),Date.parse('01/03/2021'));

System.assertEquals(Date.parse('01/03/2021'),d);

}

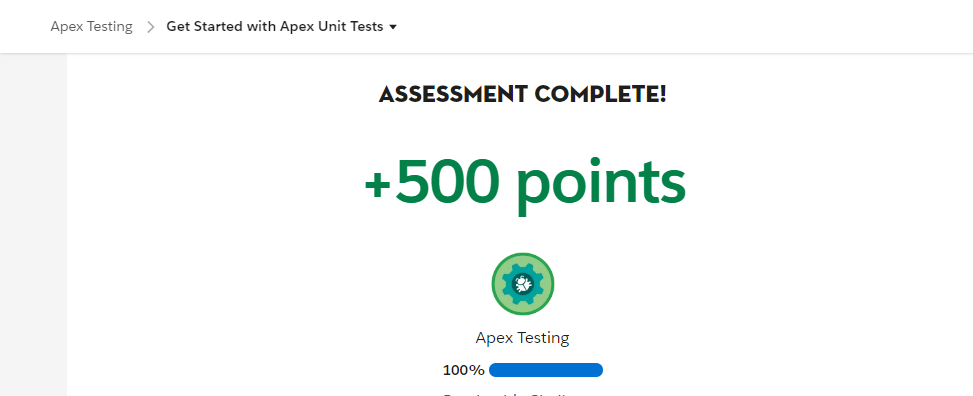
@isTest static void test2(){

Date d=verifyDate.CheckDates(Date.parse('01/01/2021'),Date.parse('03/03/2021'));

System.assertEquals(Date.parse('01/31/2021'),d);

}

}



Challenge-2

Test Apex Triggers

**RestrictContactByName.apxc:**

trigger RestrictContactByName on Contact (before insert, before update) {

For (Contact c :Trigger.New) {

if(c.LastName == 'INVALIDNAME') {

c.AddError('The Last Name "'+c.LastName+'" is not allowed for DML');

}

}

}

**TestRestrictContactByName.apxc:**

@isTest

public class TestRestrictContactByName {

@isTest static void createBadContact()

{

Contact c=new Contact(FirstName='John',LastName='INVALIDNAME');

Test.startTest();

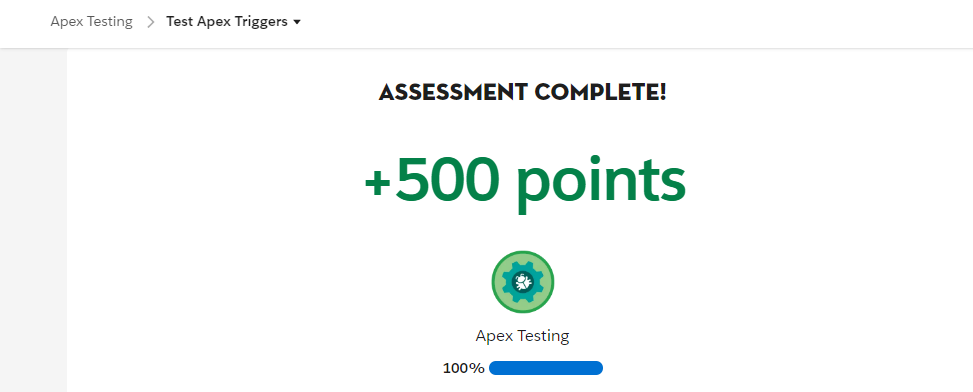
Database.SaveResult result = Database.insert(c, false);

Test.stopTest();

System.assert(!result.isSuccess());

}

}



**Challenge-3**

**Create Test Data for Apex Tests**

**RandomContactFactory.apxc:**

public class RandomContactFactory {

public static List<Contact>generateRandomContacts(Integer num,StringlastName){

List<Contact>contactList = new List<Contact>();

for(Integer i=1;i<=num;i++){

Contact ct=new Contact(FirstName ='Test'+i,LastName =lastname);

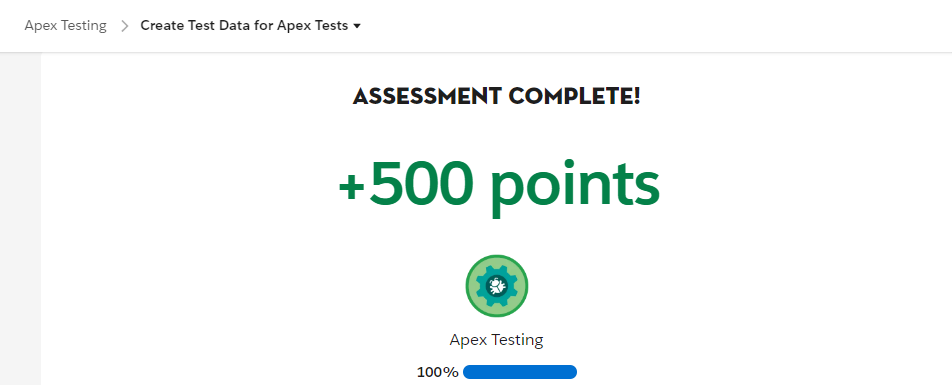
contactList.add(ct);

}

return contactList;

}

}



# Asynchronous Apex

# Challenge-1

# Use Future Methods

# 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;

# }

# }

# 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();

# }

# }

# 

Challenge-2

Use Batch Apex

**LeadProcessor.apxc:**

global class LeadProcessor implements Database.Batchable<sObject> {

global Integer count = 0;

global Database.QueryLocatorstart(Database.BatchableContextbc) {

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

}

global void execute(Database.BatchableContextbc, 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.BatchableContextbc){

system.debug('count = ' + count);

}

}

**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();

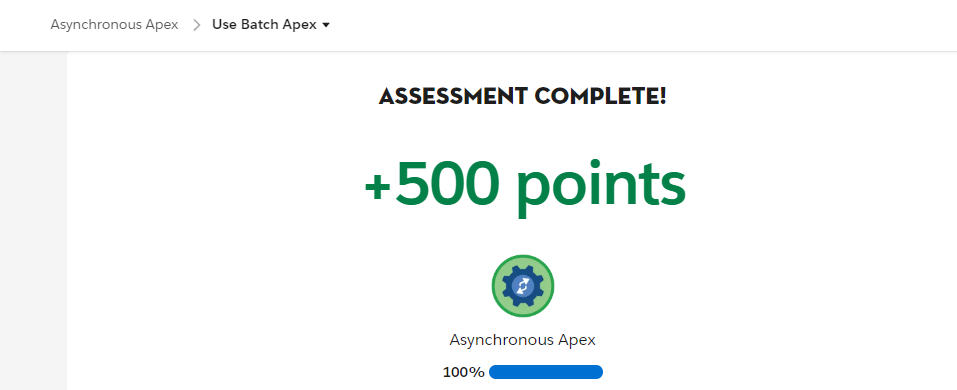
LeadProcessorlp = new LeadProcessor();

Id batchId= Database.executeBatch(lp);

Test.stopTest();

}

}



Challenge-3

[Control Processes with Queueable Apex](https://trailhead.salesforce.com/content/learn/modules/asynchronous_apex/async_apex_queueable?trailmix_creator_id=trailblazerconnect&trailmix_slug=salesforce-developer-catalyst)

**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;

}

}

}

**AddPrimaryContactTest.apxc:**

@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;

AddPrimaryContactaddit = new addPrimaryContact(testContact, 'CA');

Test.startTest();

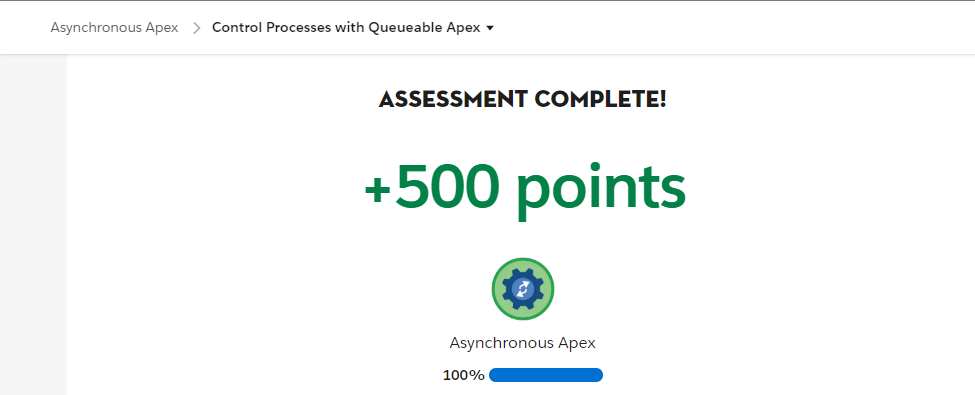
system.enqueueJob(addit);

Test.stopTest();

System.assertEquals(50, [Select count() from Contact where accountId in (Select Id from Account where BillingState='CA')]);

}

}



Challenge-4

[Schedule Jobs Using the Apex Scheduler](https://trailhead.salesforce.com/content/learn/modules/asynchronous_apex/async_apex_scheduled?trailmix_creator_id=trailblazerconnect&trailmix_slug=salesforce-developer-catalyst)

**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;

}

}

}

**DailyLeadProcessorTest.apxc:**

@isTest

private class DailyLeadProcessorTest {

static testmethod void testDailyLeadProcessor() {

String CRON\_EXP = '0 0 1 \* \* ?';

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

for(Integer i=0; i<200; i++){

lead.add(new Lead(LastName ='Dreamforce'+i, Company ='Test Inc.', Status = 'Open. Not Contacted'));

}

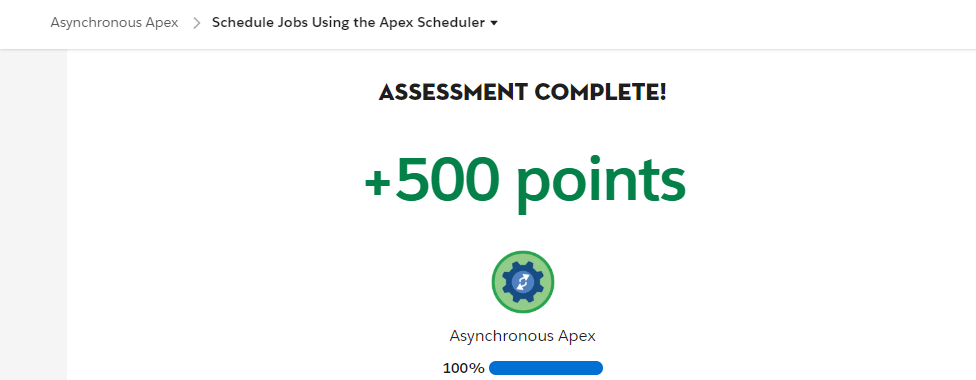
insert lead;

Test.startTest();

String jobId = System.schedule('DailyLeadProcessor', CRON\_EXP, new DailyLeadProcessor());

}

}



**APEX INTEGRATION SERVICES**

**Challenge-1**

**Apex REST Callouts**

**AnimalLocator.apxc:**

public class AnimalLocator {

public static String getAnimalNameById(Integer animalId){

String animalName;

Http http = new Http();

HttpRequest request = new HttpRequest();

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

request.setMethod('GET');

HttpResponse response = http.send(request);

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

Map<String , Object> r = (Map<String, Object>)

JSON.deserializeUntyped(response.getBody());

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

animalName = string.valueOf(animal.get('name'));

}

return animalName;

}

}

**AnimalLocatorMock.apxc:**

@isTest

global class AnimalLocatorMock implements HttpCalloutMock {

global HttpResponserespond(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 getAnimalNameByIdTest() {

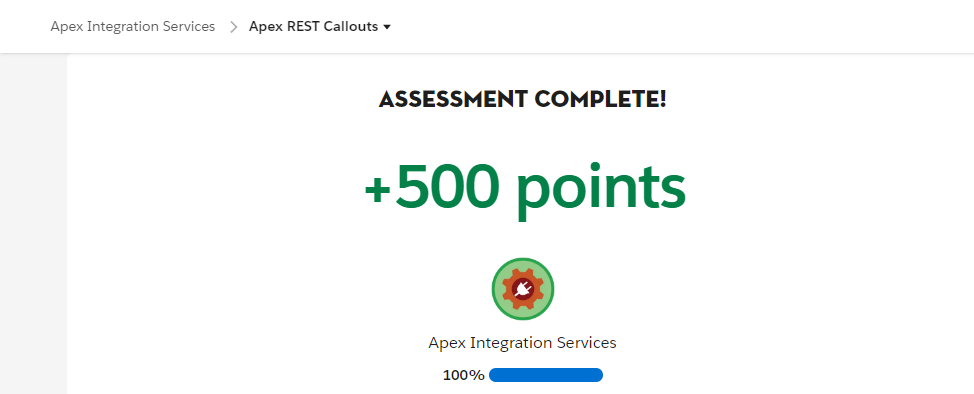
Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());

String response = AnimalLocator.getAnimalNameById(1);

System.assertEquals('chicken', response);

}

}



**Challenge-2**

**Apex SOAP Callouts**

**ParkLocator.apxc:**

public class ParkLocator {

public static List<String>country(String country){

ParkService.ParksImplPortparkservice = new ParkService.ParksImplPort();

return parkservice.byCountry(country);

}

}

**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>();

parks.add('Yosemite');

parks.add('Yellowstone');

parks.add('Another Park');

System.assertEquals(parks, result);

}

}

**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) {

List<String> parks = new List<string>();

parks.add('Yosemite');

parks.add('Yellowstone');

parks.add('Another Park');

ParkService.byCountryResponseresponse\_x =

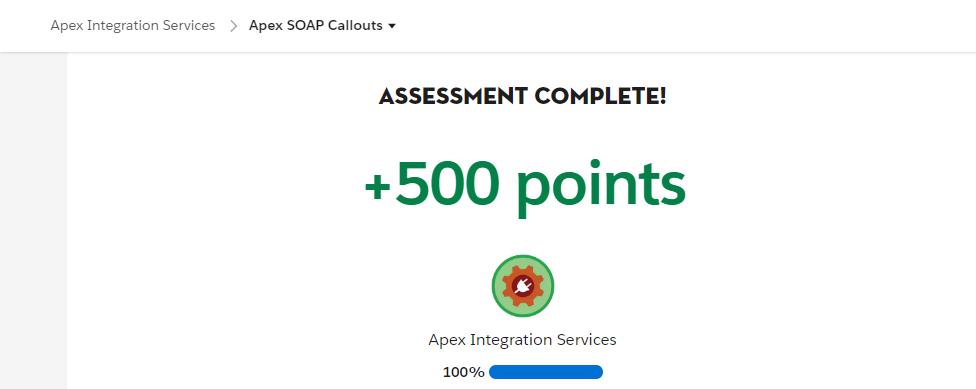
new ParkService.byCountryResponse();

response\_x.return\_x = parks;

response.put('response\_x', response\_x);

}

}



**Challenge-3**

**Apex Web Services**

**AccountManager.apxc:**

@RestResource(urlMapping='/Accounts/\*')

global with sharing class AccountManager {

@HttpGet

global static Account getAccount() {

RestRequest request = RestContext.request;

// grab the caseId from the end of the URL

String AccountId = request.requestURI.substringBetween('Accounts/','/contacts');

Account result = [SELECT Id, Name, (Select Id, Name from Contacts) FROM Account WHERE Id = :accountId];

return result;

}

}

**AccountManagerTest.apxc:**

@IsTest

private class AccountManagerTest {

@isTest static void testGetContactsByAccountId() {

Id recordId = createTestRecord();

// Set up a test request

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;

}

}

