Apex triggers ;

1).**Create an Apex trigger**

- Name: **AccountAddressTrigger**

```
trigger AccountAddressTrigger on Account (before insert, before update) {
   for(Account a cc : trigger.new){
      if(a cc.Match_Billing_Address__c == true){
          a cc.ShippingPostalCode = a cc.BillingPostalCode;
      }

   }
}
```

2). Bulk Apex trigger

- Name: **ClosedOpportunityTrigger**

```
trigger ClosedOpportunityTrigger on Opportunity (after insert, after update) {
    List<Task> taskList = new List<Task>();

   for(opportunity opp : Trigger.new){
      if(opp.stage-name == 'Closed Won'){
         taskList.add(new Task(Subject = 'Follow Up Test Task',
                        WhatId=opp.Id));
      }
   }
   if(taskList.size()>0){
      insert taskList;
   }
}
```

Apex Testing;

Apex test

- Name:**Test VerifyDate**

```
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;
	}

}
```

- Name: **RestrictContactByName**

```
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') {        //invalid name is invalid
                        c.AddError('The Last Name "'+c.LastName+'" is not allowed for
DML');
                }

        }

}
```

TestRestrictContactByName

```
 @isTest
public class TestRestrictContactByName {
       @isTest static void Test_insertupdateContact(){
            Contact cent = new Contact();
            cent.LastName = 'INVALIDNAME';

             Test.startTest();
              Database.SaveResult result = Database.insert(cent,
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());
  }

}
```

Apex class in the public scope

- Name: **RandomContactFactory** (without the @isTest annotation)

```
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);
      }



     // insert con;

      Return con;
  }
}
```

Asynchronous Apex;

1)Use Future Methods

- Name: **AccountProcessor**

```
public class AccountProcessor {
   @future
   public static void count contacts(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;

    }
}
```

- Name: **AccountProcessorTest**

```
@isTest
Private class AccountProcessorTest {
    @isTest
    private static void testCountContacts(){
        Account newAccount = new Account(Name='Test Account');
        insert newAccount;
```

uses Batch Apex

- Name: **LeadProcessor**

```
global class LeadProcessor implements Database.Batch-able <SObject> {
//Start Method
global Database.QueryLocator Start(Database.BatchableContext bc) {
String Query = 'Select Id, LeadSource from Lead';
return Database.getQueryLocator(Query);
}
//Execute Method
global void execute(Database.BatchableContext bc, List<Lead> Scope) {
if(Scope != null && !Scope.isEmpty()) {
for(Lead L : Scope) {
L.LeadSource = 'DreamForce';
}
update Scope;
}
}
//Finish Method
```

```
global void finish(Database.BatchableContext bc) {
Id BatchId = bc.getJobId();
system.debug('BatchId::'+ BatchId);
}

}
```

- Name: **LeadProcessorTest**

```
@isTest
public class LeadProcessorTest {
@testSetup
static void setup() {
List<Lead> Leads = new List<Lead>();
for(Integer i=0; i <100; i++) {
Leads.add(new lead(LastName = 'LastName'+i, FirstName='FirstName'+i,
Email='test'+i+'@theblogreaders.com', LeadSource='Web', Company='TRP'));
}
insert Leads;
}

static test-method void test() {
Test.startTest();
LeadProcessor lp =new LeadProcessor();
```

*Control Processes with Queueable Apex

- Name: **AddPrimaryContact**

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

public AddPrimaryContact(Contact c, String state) {
```

```
this.c = c;
this.state = state;
}
```

- Name: **AddPrimaryContactTest**

```
@isTest
public class AddPrimaryContactTest {

@testSetup
static void setup() {
List<Account> insertAccount = new List<Account>();
for(integer i=0; i<=100; i++) {
if(i <=50) {
insertAccount.add(new Account(Name='Acc'+i, BillingState = 'NY'));
} else {
insertAccount.add(new Account(Name='Acc'+i, BillingState = 'CA'));
}
}
insert insertAccount;
}

static testMethod void testAddPrimaryContact() {
Contact con = new Contact(LastName = 'LastName');
AddPrimaryContact addPC = new AddPrimaryContact(con, 'CA');
Test.startTest();
system.enqueueJob(addPC);
Test.stopTest();

system.assertEquals(50, [select count() from Contact]);
}
}
```

*Schedule Jobs Using the Apex Scheduler

- Name: **DailyLeadProcessor**

```
public class DailyLeadProcessor implements schedulable{

   public void execute(schedulableContext sc) {
      List<lead> l_lst_new = new List<lead>();
      List<lead> l_lst = new List<lead>([select id, lead-source from lead where leadsource
= null]);
      for(lead l : l_lst) {
         l.leadsource = 'Dreamforce';
         l_lst_new.add(l);
      }
      update l_lst_new;
   }

}
```

- Name: **DailyLeadProcessorTest**

```
@isTest
public class DailyLeadProcessorTest {
   @testSetup
   static void setup(){
      List<Lead> lstOfLead = new List<Lead>();
      for(Integer i = 1; i <= 200; i++){
         Lead ld = new Lead(Company = 'Comp' + i ,LastName = 'LN'+i, Status = 'Working -
Contacted');
         lstOfLead.add(ld);
      }
      Insert lstOfLead;
   }
   static test method void testDailyLeadProcessorScheduledJob(){
      String sch = '0 5 12 * * ?';
```

```
        Test.startTest();
        String jobId = System.schedule('ScheduledApexTest', sch, new
DailyLeadProcessor());

        List<Lead> lstOfLead = [SELECT Id FROM Lead WHERE LeadSource = null LIMIT
200];
        System.assertEquals(200, lstOfLead.size());

        Test.stopTest();
    }
}
```

Apex Integration Services

Apex REST Callouts

Name: **AnimalLocator**

```
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 the request is successful, parse the JSON response.
      if (response.getStatusCode() == 200)
      {
          // Deserializes the JSON string into collections of primitive data types.
         Map<String, Object> results = (Map<String, Object>)
JSON.deserializeUntyped(response.getBody());
          // Cast the values in the 'animals' key as a list
         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 ;
    }

}
```

- Name: **AnimalLocatorTest**

```
@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);
    }
}
```

AnimalLocatorMock

```
@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;
    }
}
```

ParkLocator

```
public class ParkLocator{
```

```
        public static List country(String country){
            ParkService.ParksImplPort parkservice =
                new parkService.parksImplPort();
             return parkservice.byCountry(country);
    }
}
```

1. **Apex SOAP Callouts**

- Name: **ParkService** (Tip: After you click the **Parse WSDL** button, change the Apex class name from **parksServices** to `ParkService`)

```
//Generated by wsdl2apex

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;
        }
    }
}
```

- Name: **ParkLocatorTest**

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

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

Name: **ParkServiceMock to mock**;

```
@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

- Name: **AccountManager**

```
@RestResource(urlMapping='/Accounts/*/contacts')
global with sharing class AccountManager {


   @HttpGet
   global static account getAccount() {

      RestRequest request = RestContext.request;

      String accountId = request.requestURI.substring(request.requestURI.lastIndexOf('/')-18,
        request.requestURI.lastIndexOf('/'));
```

```
        List<Account> a = [select id, name, (select id, name from contacts) from account where id =
:accountId];
        List<contact> co = [select id, name from contact where account.id = :accountId];
        system.debug('** a[0]= '+ a[0]);
        return a[0];

    }

}
```

- Name: **AccountManager Test**

```
@istest
public class AccountManagerTest {
@istest static void testGetContactsByAccountId() {
Id recordId = createTestRecord();
// Set up a test request
RestRequest request = new RestRequest();
request.requestUri =
'https://yourInstance.salesforce.com/services/apexrest/Accounts/'+
recordId+'/Contacts';
request.httpMethod = 'GET';
RestContext.request = request;

Account thisAccount = AccountManager.getAccount();
```

Apex Specialist

Named: **MaintenanceRequestHelper**

```
public with sharing class MaintenanceRequestHelper {
    public static void updateworkOrders(List<Case> updWorkOrders, Map<Id,Case>
nonUpdCaseMap) {
```

```
    Set<Id> validIds = new Set<Id>();
```

Named: **MaintenanceRequestHelperTest**

```
@istest
public with sharing class MaintenanceRequestHelperTest {

  private static final string STATUS_NEW = 'New';
  private static final string WORKING = 'Working';
```

Named: **MaintenanceRequest**

```
trigger MaintenanceRequest on Case (before update, after update) {
  if(Trigger.isUpdate && Trigger.isAfter){
    MaintenanceRequestHelper.updateWorkOrders(Trigger.New, Trigger.OldMap);
  }
}
```

WarehouseCalloutService:-

```
public with sharing class WarehouseCalloutService implements Queueable {
  private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';

  //class that makes a REST callout to an external warehouse system to get a list of equipment
that needs to be updated.
  //The callout's JSON response returns the equipment records that you upsert in Salesforce.

  @future(callout=true)
  public static void runWarehouseEquipmentSync(){
    Http http = new Http();
    HttpRequest request = new HttpRequest();

    request.setEndpoint(WAREHOUSE_URL);
    request.setMethod('GET');
```

```
        HttpResponse response = http.send(request);

        List<Product2> warehouseEq = new List<Product2>();

        if (response.getStatusCode() == 200){
            List<Object> jsonResponse =
(List<Object>)JSON.deserializeUntyped(response.getBody());
            System.debug(response.getBody());

            //class maps the following fields: replacement part (always true), cost, current inventory,
lifespan, maintenance cycle, and warehouse SKU
            //warehouse SKU will be external ID for identifying which equipment records to update
within Salesforce
            for (Object eq : jsonResponse){
                Map<String,Object> mapJson = (Map<String,Object>)eq;
                Product2 myEq = new Product2();
                myEq.Replacement_Part__c = (Boolean) mapJson.get('replacement');
                myEq.Name = (String) mapJson.get('name');
                myEq.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');
                myEq.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
                myEq.Cost__c = (Integer) mapJson.get('cost');
                myEq.Warehouse_SKU__c = (String) mapJson.get('sku');
                myEq.Current_Inventory__c = (Double) mapJson.get('quantity');
                myEq.ProductCode = (String) mapJson.get('_id');
                warehouseEq.add(myEq);
            }

            if (warehouseEq.size() > 0){
                upsert warehouseEq;
                System.debug('Your equipment was synced with the warehouse one');
            }
        }
    }

    public static void execute (QueueableContext context){
        runWarehouseEquipmentSync();
    }

}
```

WarehouseCalloutServiceMock:-

```
@isTest
global class WarehouseCalloutServiceMock implements HttpCalloutMock {
    // implement http mock callout
    global static HttpResponse respond(HttpRequest request){

        System.assertEquals('https://th-superbadge-apex.herokuapp.com/equipment',
request.getEndpoint());
        System.assertEquals('GET', request.getMethod());

        // Create a fake response
        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type', 'application/json');

response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":false,"quantity":5,"name":
"Generator 1000 kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"}]');
        response.setStatusCode(200);
        return response;
    }
}
```

WarehouseCalloutServiceTest

```
@isTest
private class WarehouseCalloutServiceTest {
    @isTest
    static void testWareHouseCallout(){
        Test.startTest();
        // implement mock callout test here
        Test.setMock(HTTPCalloutMock.class, new WarehouseCalloutServiceMock());
        WarehouseCalloutService.runWarehouseEquipmentSync();
        Test.stopTest();
        System.assertEquals(1, [SELECT count() FROM Product2]);
    }
}
```