Super badge 1

1. Apex Triggers Badge
   a. Get started with Apex Triggers

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

b. Bulk Apex Triggers

```
trigger ClosedOpportunityTrigger on Opportunity (after insert,after update) {

    if(trigger.isInsert || trigger.isUpdate){
   OpptyHandlerclass.updatetask(trigger.new);
   }
}
```

2. Apex Testing
   a. Get started with Apex Unit tests

```
        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;
        }

}
```

b. Test Apex Triggers

```
      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');
                }

        }
```

```
}
```

c. Create Test data for Apex tests

```
    public class RandomContactFactory {
   public static List<Contact> generateRandomContacts(Decimal number1,String
lastname){
    List<Contact> con = new List<Contact>();

    For(integer i=1;i<= number1;i++ ){
       Contact con1 = new contact();
       con1.FirstName= 'Test'+i;
       con1.LastName =lastname+i;
       con.add(con1);
    }
     //insert con;
     return con;
   }

}
```

3. Asynchornous Apex
b. Use Future Methods

```
   public class AccountProcessor {
    @future(callout=true)
   public static void countContacts(Set<Id> accountID){
      List<Contact> contactList =new List<Contact>();
      List<aCCOUNT> accRecords = [SELECT id,(Select id From Contacts)
                                  From Account where id In : accountID];
      For(Account acc:accRecords){
        contactList.add(acc.Contacts);
        acc.Number_Of_Contacts__c = contactList.size();
      }
```

```
    update accRecords;

  }

}
```

```
AccountProcessorTest
@isTest
public class AccountProcessorTest {
    public static testmethod  void TestAccountProcessorTest(){
        Set<ID> idSet = new Set<ID>();
        Account acc= New Account();
        acc.Name ='Test';
        insert acc;
        Contact con= new contact();
        con.FirstName = 'Demo';
        con.LastName = 'Test Bansal';
        con.AccountId =acc.Id;
        insert con;
        //adding  id to set
        idSet.add(acc.Id);

        Test.startTest();
        AccountProcessor.countContacts(idSet);
        Test.stopTest();

    }

}
```

```
c. Use Batch Apex
lead processor
public class LeadProcessor implements
    Database.Batchable<sObject>, Database.Stateful {
        public Integer recordsProcessed = 0;
        public Database.QueryLocator
```

```
start(Database.BatchableContext bc) {
            return Database.getQueryLocator(
            'SELECT Id,LeadSource,Company,Name from Lead');
        }
        public void execute(Database.BatchableContext bc,
List<Lead> scope){
            // process each batch of records
            for (Lead leadRecords : scope) {
                leadRecords.LeadSource ='Dreamforce';



            }
            update scope;

}
        public void finish(Database.BatchableContext bc){
            System.debug(recordsProcessed + ' records processed.
Shazam!');
        AsyncApexJob job = [SELECT Id, Status, NumberOfErrors,
            JobItemsProcessed,
            TotalJobItems, CreatedBy.Email
            FROM AsyncApexJob
            WHERE Id = :bc.getJobId()];
        // call some utility to send email
        //EmailUtils.sendMessage(job, recordsProcessed);
        }
    }

lead processor test
@isTest
public class LeadProcessorTest {
     @testSetup
    static void setup() {
        List<Lead> insertLead =  new List<Lead>();
        for(Integer  i=0;i<200;i++){
            Lead leadRecord= new Lead();
```

```
            leadRecord.LastName ='Test'+i;
            leadRecord.Company ='Test Company'+i;
            insertLead.add(leadRecord);
        }
        insert insertLead;

    }

      @isTest static void test() {
        Test.startTest();
        LeadProcessor uca = new LeadProcessor();
        Id batchId = Database.executeBatch(uca);
        Test.stopTest();
        // after the testing stops, assert records were updated
properly
        //System.assertEquals(10, [select count() from contact
where MailingCity = 'New York']);
    }

}
```

d. Control processes with Queueable Apex

AddPrimaryContact

```
public class AddPrimaryContact implements Queueable {
     private Contact contacts;
    private String state;
    public  AddPrimaryContact(Contact con, String
stateAbbreviation){
        this.contacts=con;
        this.state=stateAbbreviation;

    }
    public void execute(QueueableContext context) {
        List<Contact> contactRecords =new List<Contact>();
        List<Account> accRecord=[SELECT id,Name,(Select
id,FirstName,LastName from Contacts)
                          From Account where BillingState =
```

```
:state LIMIT 200];

        For(Account acc :accRecord ){

            Contact c = contacts.clone(false,false,false,false);
            c.AccountId= acc.Id;
            contactRecords.add(c);
        }
        System.debug('Print acc records'+ accRecord);
        insert contactRecords;

    }

}
```

AddPrimaryContactTest

```
@istest
public class AddPrimaryContactTest {
   @testSetup
    public static void  testSetup(){
        List<Account> accountRecordList = new List<Account>();
        List<Account> accountWithCAList = new List<Account>();
        for(Integer i=0;i< 50;i++){
            Account acc= new Account();
            acc.Name= 'Test NY'+i;
            acc.BillingState  ='NY';
            accountRecordList.add(acc);
        }
        insert accountRecordList;

        //insert account with CA
        for(Integer i=0;i< 50;i++){
            Account acc= new Account();
            acc.Name= 'Test CA'+i;
            acc.BillingState  ='CA';
            accountWithCAList.add(acc);
        }
```

```
        insert accountWithCAList;

    }
    static testmethod void testQueueable() {
        Contact co = new Contact();
         co.FirstName='demo';
         co.LastName ='demo';
         insert co;
         String state = 'CA';

          AddPrimaryContact apc = new AddPrimaryContact(co,
state);
          Test.startTest();
            System.enqueueJob(apc);
          Test.stopTest();
    }

}
```

e. Schedile Job using Apex Scheduler

DailyLeadProcessor

```
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
@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();
        // Schedule the test job
        String jobId = System.schedule('Update LeadSource to
DreamForce', CRON_EXP, new DailyLeadProcessor());

        // Stopping the test will run the job synchronously
        Test.stopTest();
    }
}
```

4. Apex Integration Services

b. Apex Rest Callouts

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 animals = '';

        if(response.getStatusCode() == 200) {
            Map<String, Object> result = (Map<String,
Object>)JSON.deserializeUntyped(response.getBody());
            Map<String, Object> animal = (Map<String,
Object>)result.get('animal');
            animals = string.valueOf(animal.get('name'));
        }
        return animals;
    }
}
```

AnimalLocatorTest

```
@isTest
private class AnimalLocatorTest {
    @isTest static void AnimalLocatorMock() {
        Test.setMock(HttpCalloutMock.class, new
AnimalLocatorMock());
        String actual = AnimalLocator.getAnimalNameById(1);
        String expected = 'chicken';
        System.assertEquals(actual, expected);
    }
}
```

c. Apex SOAP Callouts

ParkLocator

```
public class ParkLocator {
    public static String[] country(String country) {
        ParkService.ParksImplPort park = new
ParkService.ParksImplPort();
```

```
        return park.byCountry(country);
    }
}

ParkLocatorTest
@isTest
public class ParkLocatorTest {
@isTest static void testCallout() {
        // This causes a fake response to be generated
        Test.setMock(WebServiceMock.class, new
ParkServiceMock());
        // Call the method that invokes a callout
        String country = 'Germany';
        String[] result = ParkLocator.Country(country);


        // Verify that a fake result is returned
        System.assertEquals(new List<String>{'Hamburg Wadden Sea
National Park', 'Hainich National Park', 'Bavarian Forest
National Park'}, result);
    }
}
```

d. Apex Web Services

AccountManager

```
@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');
        system.debug(accountId);
        Account objAccount = [SELECT Id,Name,(SELECT Id,Name
FROM Contacts) FROM Account WHERE Id = :accountId LIMIT 1];
        return objAccount;
    }
```

```
}

AccountManagerTest
@isTest
private class AccountManagerTest{
    static testMethod void testMethod1(){
        Account objAccount = new Account(Name = 'test Account');
        insert objAccount;
        Contact objContact = new Contact(LastName = 'test
Contact',
                                         AccountId =
objAccount.Id);
        insert objContact;
        Id recordId = objAccount.Id;
        RestRequest request = new RestRequest();
        request.requestUri =
            'https://sandeepidentity-dev-
ed.my.salesforce.com/services/apexrest/Accounts/'
            + recordId +'/contacts';
        request.httpMethod = 'GET';
        RestContext.request = request;
        // Call the method to test
        Account thisAccount = AccountManager.getAccount();
        // Verify results
        System.assert(thisAccount!= null);
        System.assertEquals('test Account', thisAccount.Name);
    }
}


APEX SPECIALISTS

Challenge 1
public with sharing class MaintenanceRequestHelper {
public static void updateWorkOrders(List<Case> caseList) {
List<case> newCases = new List<Case>();
Map<String,Integer> result=getDueDate(caseList);
```

```
for(Case c : caseList){
if(c.status=='closed')
if(c.type=='Repair' || c.type=='Routine Maintenance'){
Case newCase = new Case();
newCase.Status='New';
newCase.Origin='web';
newCase.Type='Routine Maintenance';
newCase.Subject='Routine Maintenance of Vehicle';
newCase.Vehicle__c=c.Vehicle__c;
newCase.Equipment__c=c.Equipment__c;
newCase.Date_Reported__c=Date.today();
if(result.get(c.Id)!=null)
newCase.Date_Due__c=Date.today()+result.get(c.Id);
else
newCase.Date_Due__c=Date.today();
newCases.add(newCase);
}
}
insert newCases;
}
//
public static  Map<String,Integer> getDueDate(List<case>
CaseIDs){
Map<String,Integer> result = new Map<String,Integer>();
Map<Id, case> caseKeys = new Map<Id, case> (CaseIDs);
List<AggregateResult> wpc=[select Maintenance_Request__r.ID
cID,min(Equipment__r.Maintenance_Cycle__c)cycle
from Work_Part__c where  Maintenance_Request__r.ID in
:caseKeys.keySet() group by
Maintenance_Request__r.ID ];
for(AggregateResult res :wpc){
Integer addDays=0;
if(res.get('cycle')!=null)
addDays+=Integer.valueOf(res.get('cycle'));
result.put((String)res.get('cID'),addDays);
}
return result;
```

```
}
}


Triggers
trigger MaintenanceRequest on Case (before update, after update)
{
// ToDo: Call MaintenanceRequestHelper.updateWorkOrders
if(Trigger.isAfter)
MaintenanceRequestHelper.updateWorkOrders(Trigger.New);
}

Challenge 2

public with sharing class WarehouseCalloutService {
private static final String WAREHOUSE_URL = 'https://th-
superbadge-apex.herokuapp.com/equipment';
@future(callout=true)
public static void runWarehouseEquipmentSync() {
//ToDo: complete this method to make the callout (using @future)
to the
//      REST endpoint and update equipment on hand.
HttpResponse response = getResponse();
if(response.getStatusCode() == 200)
{
List<Product2> results = getProductList(response); //get list of
products from Http callout response
if(results.size() >0)
upsert results Warehouse_SKU__c; //Upsert the products in your
org based on the external ID SKU
}
}
//Get the product list from the external link
public static List<Product2> getProductList(HttpResponse
response)
{
List<Object> externalProducts = (List<Object>)
```

```
JSON.deserializeUntyped(response.getBody()); //desrialize the
json response
List<Product2> newProducts = new List<Product2>();
for(Object p : externalProducts)
{
Map<String, Object> productMap = (Map<String, Object>) p;
Product2 pr = new Product2();
//Map the fields in the response to the appropriate fields in the
Equipment object
pr.Replacement_Part__c = (Boolean)productMap.get('replacement');
pr.Cost__c = (Integer)productMap.get('cost');
pr.Current_Inventory__c = (Integer)productMap.get('quantity');
pr.Lifespan_Months__c = (Integer)productMap.get('lifespan') ;
pr.Maintenance_Cycle__c =
(Integer)productMap.get('maintenanceperiod');
pr.Warehouse_SKU__c = (String)productMap.get('sku');
pr.ProductCode = (String)productMap.get('_id');
pr.Name = (String)productMap.get('name');
newProducts.add(pr);
}
return newProducts;
}
// Send Http GET request and receive Http response
public static HttpResponse getResponse() {
Http http = new Http();
HttpRequest request = new HttpRequest();
request.setEndpoint(WAREHOUSE_URL);
request.setMethod('GET');
HttpResponse response = http.send(request);
return response;
}
}
```

Execute

```
WarehouseCalloutService.runWarehouseEquipmentSync();
```

Challenge 3

```
global  class WarehouseSyncSchedule implements Schedulable{
// implement scheduled code here
global  void execute (SchedulableContext sc){
WarehouseCalloutService.runWarehouseEquipmentSync();
//optional this can be done by debug mode
String sch = '00 00 01 * * ?';//on 1 pm
System.schedule('WarehouseSyncScheduleTest', sch, new
WarehouseSyncSchedule());
}
}
Execute
WarehouseSyncSchedule scheduleInventoryCheck();

Challenge 4
trigger MaintenanceRequest on Case (before update, after update)
{
if(Trigger.isUpdate && Trigger.isAfter)
MaintenanceRequestHelper.updateWorkOrders(Trigger.New);
}

@IsTest
private class InstallationTests {
private static final String STRING_TEST = 'TEST';
private static final String NEW_STATUS = 'New';
private static final String WORKING = 'Working';
private static final String CLOSED = 'Closed';
private static final String REPAIR = 'Repair';
private static final String REQUEST_ORIGIN = 'Web';
private static final String REQUEST_TYPE = 'Routine Maintenance';
private static final String REQUEST_SUBJECT = 'AMC Spirit';
public static String CRON_EXP = '0 0 1 * * ?';
static testmethod void testMaintenanceRequestNegative() {
Vehicle__c vehicle = createVehicle();
insert vehicle;
Id vehicleId = vehicle.Id;
Product2 equipment = createEquipment();
```

```
insert equipment;
Id equipmentId = equipment.Id;
Case r = createMaintenanceRequest(vehicleId, equipmentId);
insert r;
Work_Part__c w = createWorkPart(equipmentId, r.Id);
insert w;
Test.startTest();
r.Status = WORKING;
update r;
Test.stopTest();
List<case> allRequest = [SELECT Id
FROM Case];
Work_Part__c workPart = [SELECT Id
FROM Work_Part__c
WHERE Maintenance_Request__c =: r.Id];
System.assert(workPart != null);
System.assert(allRequest.size() == 1);
}
static testmethod void testWarehouseSync() {
Test.setMock(HttpCalloutMock.class, new
WarehouseCalloutServiceMock());
Test.startTest();
String jobId = System.schedule('WarehouseSyncSchedule',
CRON_EXP,
new WarehouseSyncSchedule());
CronTrigger ct = [SELECT Id, CronExpression, TimesTriggered,
NextFireTime
FROM CronTrigger
WHERE id = :jobId];
System.assertEquals(CRON_EXP, ct.CronExpression);
System.assertEquals(0, ct.TimesTriggered);
Test.stopTest();
}
private static Vehicle__c createVehicle() {
Vehicle__c v = new Vehicle__c(Name = STRING_TEST);
return v;
}
```

```
private static Product2 createEquipment() {
Product2 p = new Product2(Name = STRING_TEST,
Lifespan_Months__c = 10,
Maintenance_Cycle__c = 10,
Replacement_Part__c = true);
return p;
}
private static Case createMaintenanceRequest(Id vehicleId, Id
equipmentId) {
Case c = new Case(Type = REPAIR,
Status = NEW_STATUS,
Origin = REQUEST_ORIGIN,
Subject = REQUEST_SUBJECT,
Equipment__c = equipmentId,
Vehicle__c = vehicleId);
return c;
}
private static Work_Part__c createWorkPart(Id equipmentId, Id
requestId) {
Work_Part__c wp = new Work_Part__c(Equipment__c = equipmentId,
Maintenance_Request__c = requestId);
return wp;
}
}

public with sharing class MaintenanceRequestHelper {
public static void updateWorkOrders(List<case> caseList) {
List<case> newCases = new List<case>();
Map<String,Integer> result=getDueDate(caseList);
for(Case c : caseList){
if(c.status=='closed')
if(c.type=='Repair' || c.type=='Routine Maintenance'){
Case newCase = new Case();
newCase.Status='New';
newCase.Origin='web';
newCase.Type='Routine Maintenance';
newCase.Subject='Routine Maintenance of Vehicle';
```

```
newCase.Vehicle__c=c.Vehicle__c;
newCase.Equipment__c=c.Equipment__c;
newCase.Date_Reported__c=Date.today();
if(result.get(c.Id)!=null)
newCase.Date_Due__c=Date.today()+result.get(c.Id);
else
newCase.Date_Due__c=Date.today();
newCases.add(newCase);
}
}
insert newCases;
}
//
public static  Map<String,Integer> getDueDate(List<case>
CaseIDs){
Map<String,Integer> result = new Map<String,Integer>();
Map<Id, case> caseKeys = new Map<Id, case> (CaseIDs);
List<aggregateresult> wpc=[select Maintenance_Request__r.ID
cID,min(Equipment__r.Maintenance_Cycle__c)cycle
from Work_Part__c where  Maintenance_Request__r.ID in
:caseKeys.keySet() group by
Maintenance_Request__r.ID ];
for(AggregateResult res :wpc){
Integer addDays=0;
if(res.get('cycle')!=null)
addDays+=Integer.valueOf(res.get('cycle'));
result.put((String)res.get('cID'),addDays);
}
return result;
}
}

@isTest
public  class MaintenanceRequestTest {
static  List<case> caseList1 = new List<case>();
static List<product2> prodList = new List<product2>();
static List<work_part__c> wpList = new List<work_part__c>();
```

```
@testSetup
static void getData(){
caseList1= CreateData( 300,3,3,'Repair');
}
public static List<case>   CreateData( Integer numOfcase,
Integer numofProd, Integer numofVehicle,
String type){
List<case> caseList = new List<case>();
//Create Vehicle
Vehicle__c vc = new Vehicle__c();
vc.name='Test Vehicle';
upsert vc;
//Create Equiment
for(Integer i=0;i<numofProd;i++){
Product2 prod = new Product2();
prod.Name='Test Product'+i;
if(i!=0)
prod.Maintenance_Cycle__c=i;
prod.Replacement_Part__c=true;
prodList.add(prod);
}
upsert  prodlist;
//Create Case
for(Integer i=0;i< numOfcase;i++){
Case newCase = new Case();
newCase.Status='New';
newCase.Origin='web';
if( math.mod(i, 2) ==0)
newCase.Type='Routine Maintenance';
else
newCase.Type='Repair';
newCase.Subject='Routine Maintenance of Vehicle' +i;
newCase.Vehicle__c=vc.Id;
if(i<numofProd)
newCase.Equipment__c=prodList.get(i).ID;
else
newCase.Equipment__c=prodList.get(0).ID;
```

```
caseList.add(newCase);
}
upsert caseList;
for(Integer i=0;i<numofProd;i++){
Work_Part__c wp = new Work_Part__c();
wp.Equipment__c   =prodlist.get(i).Id   ;
wp.Maintenance_Request__c=caseList.get(i).id;
wplist.add(wp) ;
}
upsert wplist;
return caseList;
}
public static testmethod void testMaintenanceHelper(){
Test.startTest();
getData();
for(Case cas: caseList1)
cas.Status ='Closed';
update caseList1;
Test.stopTest();
}
}
```

Challenge 5

```
@IsTest
private class WarehouseCalloutServiceTest {
// implement your mock callout test here
@isTest
static void testWareHouseCallout(){
Test.setMock(HttpCalloutMock.class, new
WarehouseCalloutServiceMock());
WarehouseCalloutService.runWarehouseEquipmentSync();
}
}

@isTest
public class WarehouseCalloutServiceMock implements
HTTPCalloutMock {
```

```
// implement http mock callout
public HTTPResponse respond (HttpRequest request){
HttpResponse response = new HTTPResponse();
response.setHeader('Content-type','application/json');
response.setBody('[{"_id":"55d66226726b611100aaf741","replacemen
t":false,"quantity":5,"name":"Generator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"10
0003"},{"_id":"55d66226726b611100aaf742","replacement":true,"qua
ntity":183,"name":"Cooling
Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004
"},{"_id":"55d66226726b611100aaf743","replacement":true,"quantit
y":143,"name":"Fuse
20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"
}]');
response.setStatusCode(200);
return response;
}
}
```

Challenge 6

```
@isTest
private class WarehouseSyncScheduleTest {
public static String CRON_EXP = '0 0 0 15 3 ? 2022';
static testmethod void testjob(){
MaintenanceRequestTest.CreateData( 5,2,2,'Repair');
Test.startTest();
Test.setMock(HttpCalloutMock.class, new
WarehouseCalloutServiceMock());
String joBID= System.schedule('TestScheduleJob', CRON_EXP, new
WarehouseSyncSchedule());
// List<Case> caselist = [Select count(id) from case where case]
Test.stopTest();
}
}
```

Super Badges 2

Challenge 1

```
OR(AND(LEN(State) > 2,
NOT(CONTAINS("AL:AK:AZ:AR:CA:CO:CT:DE:DC:FL:GA:HI:ID:IL:IN:IA:KS
:KY:LA:ME:MD:MA:MI:MN:MS:MO:MT:NE:NV:NH:NJ:NM:NY:NC:ND:OH:OK:OR:
PA:RI:SC:SD:TN:TX:UT:VT:VA:WA:WV:WI:WY", State )) ),
NOT(OR(Country ="US",Country ="USA",Country ="United States",
ISBLANK(Country))))
```

Challenge 2

```
OR(AND(LEN(BillingState) > 2,
NOT(CONTAINS("AL:AK:AZ:AR:CA:CO:CT:DE:DC:FL:GA:HI:ID:IL:IN:IA:KS
:KY:LA:ME:MD:MA:MI:MN:MS:MO:MT:NE:NV:NH:NJ:NM:NY:NC:ND:OH:OK:OR:
PA:RI:SC:SD:TN:TX:UT:VT:VA:WA:WV:WI:WY", BillingState ))
),AND(LEN(ShippingState) > 2,
NOT(CONTAINS("AL:AK:AZ:AR:CA:CO:CT:DE:DC:FL:GA:HI:ID:IL:IN:IA:KS
:KY:LA:ME:MD:MA:MI:MN:MS:MO:MT:NE:NV:NH:NJ:NM:NY:NC:ND:OH:OK:OR:
PA:RI:SC:SD:TN:TX:UT:VT:VA:WA:WV:WI:WY", ShippingState))
),NOT(OR(BillingCountry ="US",BillingCountry
="USA",BillingCountry ="United States",
ISBLANK(BillingCountry))),
NOT(OR(ShippingCountry ="US",ShippingCountry
="USA",ShippingCountry ="United States",
ISBLANK(ShippingCountry))))
```

Validation rule 2

error condition formula

```
ISCHANGED( Name ) && ( OR( ISPICKVAL( Type ,'Customer - Direct')
,ISPICKVAL( Type ,'Customer - Channel') ))
```

Challenge 4

Opportunity Validation Rule

IF((Amount > 100000 && Approved__c <> True && ISPICKVAL(StageName,'Closed Won')),True,False)

Challenge 7
Formula
Case (WEEKDAY(Date__c),
1,"Sunday",
2,"Monday",
3,"Tuesday",
4,"Wednesday",
5,"Thursday",
6,"Friday",
7,"Saturday",
Text(WEEKDay(Date__c)))