Apex Basics and Database AccountHandler.apxc

```
public class AccountHandler {     public static Account insertNewAccount(String
AccountName){        try {             Account newacct = new
Account(Name=AccountName);             insert newacct;             return newacct;          }
catch (DmlException e) {             System.debug('A DML exception has occurred: ' +
e.getMessage());             return null; }     } } ContactAndLeadSearch.apxc public class
ContactAndLeadSearch {     //a public static method that accepts an incoming string as
a parameter     public static List> searchContactsAndLeads (String incoming) {
//write a SOSQL query to search by lead or contact name fields for the incoming string.
List> searchList = [FIND :incoming IN NAME FIELDS                 RETURNING
Contact(FirstName,LastName),Lead(FirstName,LastName)];           //return the list of
the same kind           return searchList;     }       } ContactSearch.apxc public class
ContactSearch{     public static list searchForContacts(string name1, string name2){
List  con = new List();        con = [SELECT ID,FirstName from Contact where LastName
=:name1  and MailingPostalCode=:name2];        return con;     } } StringArrayTest.apxc
public class StringArrayTest {     public static List generateStringArray(Integer N){
List TestList = new List();        for(Integer i=0;i results = (Map)
JSON.deserializeUntyped(response.getBody());          // Cast the values in the 'animals'
key as a list          Map animals = (map) 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 {     // 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('{"animal":{"id":1,"name":"chicken","eats":"chicken food","says":"cluck
cluck"}}');        response.setStatusCode(200);        return response;     } }
AnimalLocatorTest.apxc @isTestpublic class AnimalLocatorTest {  @isTest public static
void AnimalLocatorMock() {       Test.setMock(HttpCalloutMock.class, new
AnimalLocatorMock());       string result = AnimalLocator.getAnimalNameById(1);
system.debug(result);       String expectedResult = 'chicken';
System.assertEquals(result,expectedResult );    }} AsyncParksService.apxc public class
AsyncParksService {     public class byCountryResponseFuture extends
System.WebServiceCalloutFuture {        public String[] getValue() {
parksService.byCountryResponse response =
(parksService.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 inputHttpHeaders_x;        public String clientCertName_x;        public Integer
timeout_x;        private String[] ns_map_type_info = new String[]{'http://parks.services/',
'parksService'};        public AsyncParksService.byCountryResponseFuture
beginByCountry(System.Continuation continuation,String arg0) {
parksService.byCountry request_x = new parksService.byCountry();
request_x.arg0 = arg0;            return (AsyncParksService.byCountryResponseFuture)
System.WebServiceCallout.beginInvoke(             this,             request_x,
AsyncParksService.byCountryResponseFuture.class,             continuation,             new
String[]{endpoint_x,             '',             'http://parks.services/',             'byCountry',
'http://parks.services/',             'byCountryResponse',
'parksService.byCountryResponse'}            );        }     } } ParkLocator.apxc public class
ParkLocator {    public static String[] country(String country){
ParkService.ParksImplPort parks = new ParkService.ParksImplPort();       String[]
parksname = parks.byCountry(country);       return parksname;    }}
ParkLocatorTest.apxc @isTestprivate class ParkLocatorTest{    @isTest     static void
testParkLocator() {       Test.setMock(WebServiceMock.class, new ParkServiceMock());
String[] arrayOfParks = ParkLocator.country('India');       System.assertEquals('Park1',
arrayOfParks[0]);    }} 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 inputHttpHeaders_x;       public Map 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 response_map_x =
new Map();          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 @isTestglobal class ParkServiceMock implements WebServiceMock {   global void doInvoke(Object stub,   Object request,   Map 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();   List myStrings = new List {'Park1','Park2','Park3'};   response_x.return_x = myStrings;   // end   response.put('response_x', response_x);   }} Apex Testing RandomContactFactory.apxc public class RandomContactFactory {   public static List generateRandomContacts(Integer num, String lastName){   List contactList = new List();   for(Integer i = 1; i <= num; i++){   Contact ct = new Contact(FirstName = 'Test '+i, LastName = lastname);   contactList.add(ct);   }   return contactList;   } } 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');   }   } } TestRestrictContactByName.apxc @isTest public class TestRestrictContactByName {   @isTest static void testContact(){   Contact ct = new Contact();   ct.LastName = 'INVALIDNAME';   Database.SaveResult res = Database.insert(ct, false);   System.assertEquals('The Last Name "INVALIDNAME" is not allowed for DML', res.getErrors()[0].getMessage());   } } TestVerifyDate.apxc @isTest public class TestVerifyDate {   @isTest static void Test_CheckDates_case1(){   Date d = VerifyDate.CheckDates(Date.parse('01/01/2020'), Date.parse('01/03/2020'));   System.assertEquals(Date.parse('01/03/2020'), d);   }   @isTest static void Test_CheckDates_case2(){   Date d = VerifyDate.CheckDates(Date.parse('01/01/2020'), Date.parse('03/03/2020'));   System.assertEquals(Date.parse('01/31/2020'), d);   } } 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    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;    } }
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;        }     } } ClosedOpportunityTrigger.apxt trigger
ClosedOpportunityTrigger on Opportunity (after insert, after update) {     List tasklist =
new List();     for(Opportunity op: Trigger.New){        if(op.StageName == 'Closed Won'){
tasklist.add(new Task(Subject = 'Follow Up Test Task', WhatId = op.Id));        }     }
if(tasklist.size() > 0){        insert tasklist;     } } Asynchronous Apex
AccountProcessor.apxc public class AccountProcessor {     @future     public static void
countContacts(List accountsIds){        List accList = [Select Id, Number_Of_Contacts__c,
(Select Id from Contacts) from Account where Id in :accountsIds];        for(Account acc:
accList){            acc.Number_Of_Contacts__c = acc.Contacts.size();        }        update
accList;    } } AccountProcessorTest.apxc @isTest public class AccountProcessorTest {
public static testmethod void testAccountProcessor(){        Account a = new Account();
a.Name = 'Test Account';        insert a;        Contact con = new Contact();
con.FirstName = 'Yash';        con.LastName = 'Kalola';        con.AccountId = a.Id;
insert con;        List accListId = new List();        accListId.add(a.Id);        Test.startTest();
AccountProcessor.countContacts(accListId);        Test.stopTest();        Account acc =
[Select Number_Of_Contacts__c from Account where Id =: a.Id];
System.assertEquals(Integer.valueOf(acc.Number_Of_Contacts__c), 1);     } }
AddPrimaryContact.apxc public class AddPrimaryContact implements Queueable {
public contact c;     public String state;     public AddPrimaryContact(Contact c, String
state) {        this.c = c;        this.state = state;     }     public void
execute(QueueableContext qc) {        system.debug('this.c = '+this.c+' this.state =
'+this.state);        List acc_lst = new List([select id, name, BillingState from account
where account.BillingState = :this.state limit 200]);        List c_lst = new List();
for(account a: acc_lst) {            contact c = new contact();            c = this.c.clone(false,
false, false, false);            c.AccountId = a.Id;            c_lst.add(c);        }        insert c_lst;
} }
```

```
AddPrimaryContactTest.apxc @IsTest public class AddPrimaryContactTest {     @IsTest
public static void testing() {         List acc_lst = new List();         for (Integer i=0; i<50;i++) {
account a = new account(name=string.valueOf(i),billingstate='NY');
system.debug('account a = '+a);             acc_lst.add(a);         }         for (Integer i=0;
i<50;i++) {             account a = new account(name=string.valueOf(50+i),billingstate='CA');
system.debug('account a = '+a);             acc_lst.add(a);         }             insert acc_lst;
Test.startTest();         contact c = new contact(lastname='alex');         AddPrimaryContact
apc = new AddPrimaryContact(c,'CA');         system.debug('apc = '+apc);
System.enqueueJob(apc);         Test.stopTest();         List c_lst = new List([select id from
contact]);         Integer size = c_lst.size();         system.assertEquals(50, size);     } }
DailyLeadProcessor.apxc global class DailyLeadProcessor implements Schedulable {
global void execute(SchedulableContext ctx) {         //Retrieving the 200 first leads where
lead source is in blank.         List leads = [SELECT ID, LeadSource FROM Lead where
LeadSource = '' LIMIT 200];         //Setting the LeadSource field the 'Dreamforce' value.
for (Lead lead : leads) {             lead.LeadSource = 'Dreamforce';         }         //Updating all
elements in the list.         update leads;     } } DailyLeadProcessorTest.apxc @isTest
private class DailyLeadProcessorTest {     @isTest     public static void
testDailyLeadProcessor(){         //Creating new 200 Leads and inserting them.         List
leads = new List();         for (Integer x = 0; x < 200; x++) {             leads.add(new
Lead(lastname='lead number ' + x, company='company number ' + x));         }         insert
leads;         //Starting test. Putting in the schedule and running the DailyLeadProcessor
execute method.         Test.startTest();         String jobId =
System.schedule('DailyLeadProcessor', '0 0 12 * * ?', new DailyLeadProcessor());
Test.stopTest();         //Once the job has finished, retrieve all modified leads.         List
listResult = [SELECT ID, LeadSource FROM Lead where LeadSource = 'Dreamforce'
LIMIT 200];         //Checking if the modified leads are the same size number that we
created in the start of this method.         System.assertEquals(200, listResult.size());     } }
LeadProcessor.apxc global class LeadProcessor implements Database.Batchable {
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 L_list){         List L_list_new = new List();         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);     } } LeadProcessorTest.apxc @isTest public class LeadProcessorTest {
@isTest     public static void testit(){         List L_list = new List();         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();   } }   Superbadge Apex Specialist Challenge 1 : Automate Record Creation MaintenanceRequest.apxt trigger MaintenanceRequest on Case (before update, after update) {// ToDo: Call MaintenanceRequestHelper.updateWorkOrdersif(Trigger.isAfter)MaintenanceRequestHelper.upd ateWorkOrders(Trigger.New);} MaintenanceRequestHelper.apxc public with sharing class MaintenanceRequestHelper {public static void updateWorkOrders(List caseList) {List newCases = new List();Map 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()+r esult.get(c.Id);elsenewCase.Date_Due__c=Date.today();newCases.add(newCase);}}insert newCases;}//public static Map getDueDate(List CaseIDs){Map result = new Map();Map caseKeys = new Map (CaseIDs);List wpc=[select Maintenance_Request__r.ID cID,min(Equipment__r.Maintenance_Cycle__c)cyclefrom 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;}} Challenge - 2 : Synchronize Salesforce data with an external system Anonymous Window Code: WarehouseCalloutService.runWarehouseEquipmentSync(); WarehouseCalloutService.apxc public with sharing class WarehouseCalloutService {private static final String WAREHOUSE_URL = 'https://th-superbadgeapex.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 results = getProductList(response); //get list of products from Http callout responseif(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 linkpublic static List getProductList(HttpResponse response){List externalProducts = (List)

JSON.deserializeUntyped(response.getBody()); //desrialize the json responseList
newProducts = new List();for(Object p : externalProducts){Map productMap = (Map)
p;Product2 pr = new Product2();//Map the fields in the response to the appropriate fields
in the Equipment objectpr.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 responsepublic static HttpResponse getResponse() {Http
http = new Http();HttpRequest request = new
HttpRequest();request.setEndpoint(WAREHOUSE_URL);request.setMethod('GET');HttpR
espon se response = http.send(request);return response;}} Challenge - 3 : Schedule
Synchronization Anonymous Window Code WarehouseSyncSchedule
scheduleInventoryCheck(); WarehouseSyncSchedule.apxc global class
WarehouseSyncSchedule implements Schedulable{// implement scheduled code
hereglobal void execute (SchedulableContext
sc){WarehouseCalloutService.runWarehouseEquipmentSync();//optional this can be
done by debug modeString sch = '00 00 01 * * ?';//on 1
pmSystem.schedule('WarehouseSyncScheduleTest', sch, new
WarehouseSyncSchedule());}} Challenge - 4 : Test automation logic
InstallationTests.apxc @IsTestprivate 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 allRequest = [SELECT IdFROM Case];Work_Part__c workPart =
[SELECT IdFROM Work_Part__cWHERE 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, NextFireTimeFROM CronTriggerWHERE 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;}}
MaintenanceRequest.apxt trigger MaintenanceRequest on Case (before update, after
update) {if(Trigger.isUpdate &&
Trigger.isAfter)MaintenanceRequestHelper.updateWorkOrders(Trigger.New);}
MaintenanceRequestHelper.apxc public with sharing class MaintenanceRequestHelper
{public static void updateWorkOrders(List caseList) {List newCases = new List();Map
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;new
Case.
Date_Reported__c=Date.today();if(result.get(c.Id)!=null)newCase.Date_Due__c=Date.tod
ay()+r
esult.get(c.Id);elsenewCase.Date_Due__c=Date.today();newCases.add(newCase);}}inse
rt newCases;}//public static  Map getDueDate(List CaseIDs){Map result = new
Map();Map caseKeys = new Map (CaseIDs);List wpc=[select Maintenance_Request__r.ID
cID,min(Equipment__r.Maintenance_Cycle__c)cyclefrom 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;}} MaintenanceRequestTest.apxc
@isTestpublic  class MaintenanceRequestTest {static  List caseList1 = new List();static
List prodList = new List();static List wpList = new List();@testSetupstatic void
getData(){caseList1= CreateData( 300,3,3,'Repair');}public static List   CreateData(
Integer numOfcase, Integer numofProd, Integer numofVehicle,String type){List caseList
= new List();//Create VehicleVehicle__c vc = new Vehicle__c();vc.name='Test
Vehicle';upsert vc;//Create Equimentfor(Integer i=0;i caselist = [Select count(id) from
case where case] Test.stopTest(); } }
```