.

APEX SPECIALIST SUPER BADGE CODES

APEX TRIGGERS

AccountAddressTrigger.axpt:

```
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.axpt:

```
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

VerifyData.apxc:

```
public class VerifyDate {
public static Date CheckDates(Date date1, Date date2) {
if(DateWithin30Days(date1,date2)) { return date2;
} else {
return SetEndOfMonthDate(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;
}
}
```

TestVerifyData.apxc:

```
@isTest
private class TestVerifyDate {
@isTest static void Test_CheckDates_case1(){
Date D =
VerifyDate.CheckDates(date.parse('01/01/2022'),date.parse('01/05/
System.assertEquals(date.parse('01/05/2022'), D);
}
@isTest static void Test_CheckDates_case2(){ Date D =
VerifyDate.CheckDates(date.parse('01/01/2022'), date.parse('05/05/2022'));
System.assertEquals(date.parse('01/31/2022'), D);
}
@isTest static void Test_Within30Days_case1(){
Boolean flag =
VerifyDate.DateWithin30Days(date.parse('01/01/2022'),
date.parse('12/30/2021'));
System.assertEquals(false, flag);
}
@isTest static void Test_Within30Days_case2(){
Boolean flag =
VerifyDate.DateWithin30Days(date.parse('01/01/2022'), date.parse('02/02/2021'));
System.assertEquals(false, flag);
}
```

.

```
@isTest static void Test_Within30Days_case3(){

Boolean flag =
VerifyDate.DateWithin30Days(date.parse('01/01/2022'),
date.parse('01/15/2022'));
System.assertEquals(true, flag);
}
@isTest static void Test_SetEndOfMonthDate(){
Date returndate =
VerifyDate.SetEndOfMonthDate(date.parse('01/01/2022'));
}
}
```

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
}
}
}
```

TestRestrictContactByName.apxc:

```
@isTest
private 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
result.getErrors()[0].getMessage());
}
}
```

.

RandomContactFactory.apxc:

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

ASYNCHRONOUS APEX

AccountProcessor.apxc:

```
public class AccountProcessor {
   @future
   public static void countContacts(List<Id> accountId_lst) {
      Map<Id,Integer> account_cno = new Map<Id,Integer>();
      List<account> account_lst_all = new List<account>([select id, (select id from
contacts) from account]);
      for(account a:account_lst_all) {
         account_cno.put(a.id,a.contacts.size()); //populate the map
      }
      List<account> account_lst = new List<account>(); // list of account that we will
upsert
      for(Id accountId : accountId_lst) {
         if(account_cno.containsKey(accountId)) {
            account acc = new account();
            acc.Id = accountId;
            acc.Number_of_Contacts__c = account_cno.get(accountId);
            account_lst.add(acc);
         }
      }
      upsert account_lst;
   }
}
```

.

AccountProcessorTest.apxc:

```
@isTest
public class AccountProcessorTest {
   @isTest
   public static void testFunc() {
      account acc = new account();
      acc.name = 'MATW INC';
      insert acc;
      contact con = new contact();
      con.lastname = 'Mann1';
      con.AccountId = acc.Id;
      insert con;
      contact con1 = new contact();
      con1.lastname = 'Mann2';
      con1.AccountId = acc.Id;
      insert con1;
      List<Id> acc_list = new List<Id>();
      acc_list.add(acc.Id);
      Test.startTest();
AccountProcessor.countContacts(acc_list);
      Test.stopTest();
      List<account> acc1 = new List<account>([select
Number_of_Contacts__c from account where id = :acc.id]);
      system.assertEquals(2,acc1[0].Number_of_Contacts__c);
   }
}
```

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_lst) {
      List<lead> l_lst_new = new List<lead>();
      for(lead l : l_lst) {
         l.leadsource = 'Dreamforce';
         l_lst_new.add(l);
         count+=1;
      }
      update l_lst_new;
   }
   global void finish (Database.BatchableContext bc) {
      system.debug('count = '+count);
   }
}
```

LeadProcessorTest.apxc:

```
@isTest
public class LeadProcessorTest {
   @isTest
   public static void testit() {
      List<lead> l_lst = new List<lead>();
      for (Integer i = 0; i<200; i++) {
         Lead l = new lead();
         l.LastName = 'name'+i;
         l.company = 'company';
         l.Status =  'somestatus';
         l_lst.add(l);
      }
      insert l_lst;
      test.startTest();
      Leadprocessor lp = new Leadprocessor();
      Id batchId = Database.executeBatch(lp);
      Test.stopTest();

   }
}
```

.

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<Account> acc_lst = new List<account>([select id, name,
BillingState from account where account.BillingState = :this.state limit 200]);
      List<contact> c_lst = new List<contact>();
      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<account> acc_lst = new List<account>();
      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<contact> c_lst = new List<contact>([select id from contact]);
        Integer size = c_lst.size();
        system.assertEquals(50, size);
    }
}
```

DailyLeadProcessor.apxc:

```
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, leadsource from lead where leadsource
= null]);
        for(lead l : l_lst) {
            l.leadsource = 'Dreamforce';
            l_lst_new.add(l);
        }
        update l_lst_new;
    }
}
```

.

DailyLeadProcessorTest.apxc:

```
@isTest
public class DailyLeadProcessorTest {

    @isTest
    public static void testing() {

        List<lead> l_lst = new List<lead>();
        for(Integer i=0;i<200;i++) {
            lead l = new lead();
            l.lastname = 'lastname'+i;
            l.Company = 'company'+i;
            l_lst.add(l);
        }
        insert l_lst;

        Test.startTest();
        DailyLeadProcessor dlp = new DailyLeadProcessor ();
        String jobId = System.Schedule('dailyleadprocessing','0 0 0 1 12 ?
2016',dlp);
        Test.stopTest();

        List<lead> l_lst_chk = new List<lead>([select id,leadsource from lead where
leadsource != 'Dreamforce']);
        System.assertequals(0,l_lst_chk.size());
    }
}
```

APEX INTEGRATION SERVICES

AnimalLocator.apxc:

```
public class AnimalLocator { public class cls_animal { public Integer id;        public String
name; public String eats;      public String says;
}
public class JSONOutput{ public cls_animal animal;
.
//public JSONOutput parse(String json){
//return (JSONOutput) System.JSON.deserialize(json,
JSONOutput.class);
//}
}

   public static String getAnimalNameById (Integer id) {
      Http http = new Http();
      HttpRequest request = new HttpRequest();       request.setEndpoint('https://th-
apex-httpcallout.herokuapp.com/animals/' + id);
      //request.setHeader('id', String.valueof(id)); -- cannot be used in this challenge :)
      request.setMethod('GET');
      HttpResponse response = http.send(request);
      system.debug('response: ' + response.getBody());
      //Map<String,Object> map_results = (Map<String,Object>)
JSON.deserializeUntyped(response.getBody());
      jsonOutput results = (jsonOutput)
JSON.deserialize(response.getBody(), jsonOutput.class);
      //Object results = (Object) map_results.get('animal'); system.debug('results= ' +
results.animal.name);
      return(results.animal.name);
   }
}
```

AnimalLocatorMock.apxc:

```
@IsTest
global class AnimalLocatorMock implements HttpCalloutMock {

   global HTTPresponse respond(HTTPrequest request) {
      Httpresponse response = new Httpresponse();
      response.setStatusCode(200);
      //-- directly output the JSON, instead of creating a logic
      //response.setHeader('key, value)
      //Integer id = Integer.valueof(request.getHeader('id'));
      //Integer id = 1;
      //List<String> lst_body = new List<String> {'majestic badger', 'fluffy bunny'};
.
        //system.debug('animal return value: ' + lst_body[id]);
        response.setBody('{"animal":{"id":1,"name":"chicken","eats":"chicken
food","says":"cluck cluck"}}');
        return response;
    }
}
```

AnimalLocatorTest.apxc:

```
@IsTest
public class AnimalLocatorTest {
    @isTest
    public static void testAnimalLocator() {
        Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
        //Httpresponse response = AnimalLocator.getAnimalNameById(1);
        String s =  AnimalLocator.getAnimalNameById(1);
        system.debug('string returned: ' + s);
    }
}
```

ParkService.apxc:

```
//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-
soapservice.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;
.
        }
    }
}
```

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:

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

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

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) {
        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);
    }
}
```

AccountManager.apxc:

```
@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];

    }
}
```

.

AccountManagerTest.apxc:

```
@Istest(SeeAllData=true) public class AccountManagerTest {
   @IsTest
   public static void testaccountmanager() {
      RestRequest request = new RestRequest();
      request.requestUri = 'https://mannharleen-dev-
ed.my.salesforce.com/services/apexrest/Accounts/00190000016cw4tAAA/c
ontacts';
      request.httpMethod = 'GET';
      RestContext.request = request;
system.debug('test account result = '+ AccountManager.getAccount());

   }
}
```

APEX SPECIALIST SUPER BADGE

Challenge 1

MaintenanceRequestHelper.apxc:

```
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;
}
}
```

MaintenanceRequest.apxt:

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

Challenge 2:

WarehouseCalloutService.apxt:

```
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<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;
}
}
```

.

Challenge 3:

WarehouseSyncSchedule.apxt

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

Challenge 4:

MaintenanceRequest.apxt:

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

InstallationTests.apxt:

```
@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;
}
}
```

MaintenanceRequestHelper.apxt:

```
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;
}
}
```

MaintenanceRequestTest.apxt:

```
@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:

WarehouseCalloutServiceTest.apxt:

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

WarehouseCalloutServiceMock.apxt:

```
@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","replacement":fals
e,"quantity":5,"name":"Generator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"},{"_id"
:"55d66226726b611100aaf742","replacement":true,"quantity":183,"name":"Co oling
Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"_id":"55d
66226726b611100aaf743","replacement":true,"quantity":143,"name":"Fuse
20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');
response.setStatusCode(200);
return response;
}
}
```

Challenge 6:

WarehouseSyncScheduleTest.apxt:

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