

# Apex Triggers

## Get Started with Apex Triggers

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

    for(Account account:Trigger.New){
        if(account.Match_Billing_Address__c == True){
            account.ShippingPostalCode = account.BillingPostalCode;

        }
    }
}
```

## Bulk Apex Triggers

```
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

## 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
    @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;
    }
}

```

```

@Test
private class TestVerifyDate {

    @isTest static void Test_CheckDates_case1(){
        Date D =VerifyDate.CheckDates(date.parse('01/01/2020'),
date.parse('01/05/2020'));
        System.assertEquals(date.parse('01/05/2020'), D);
    }

    @isTest static void Test_CheckDates_case2(){
        Date D =VerifyDate.CheckDates(date.parse('01/01/2020'),
date.parse('05/05/2020'));
        System.assertEquals(date.parse('01/31/2020'), D);
    }

    @isTest static void Test_DateWithin30Days_case1(){
        Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
date.parse('12/30/2019'));
        System.assertEquals(false, flag);
    }

    @isTest static void Test_DateWithin30Days_case2(){
        Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
date.parse('02/02/2019'));
        System.assertEquals(false, flag);
    }

    @isTest static void Test_DateWithin30Days_case3(){
        Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
date.parse('01/15/2019'));
        System.assertEquals(true, flag);
    }

    @isTest static void Test_SetEndOfMonthDate(){
        Date returndate = VerifyDate.SetEndOfMonthDate(date.parse('01/01/2020'));
    }
}

```

# 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");
        }
    }
}

}

@isTest
public 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 for DML',
result.getErrors()[0].getMessage());
    }
}
```

## Create Test Data for Apex Tests

```
public class RandomContactFactory {
```

```

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

```

## Asynchronous Apex

### Use Future Methods

```

public class AccountProcessor {
    @future
    public static void countContacts(List<Id>accountIds){
        List<Account> accountsToUpdate = new List<Account>();

        List<Account> accounts = [Select Id, Name , (Select Id from Contacts) from
Account Where Id in :accountIds];

        For(Account acc:accounts){
            List<Contact> contactList = acc.Contacts;
            acc.Number_Of_Contacts__c = contactList.size();
            accountsToUpdate.add(acc);
        }
        update accountsToUpdate;
    }
}

@isTest
private class AccountProcessorTest {

```

```

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

        Contact newContact1 = new Contact(FirstName='Jhon',LastName =
'Deo',AccountId=newAccount.Id);
        insert newContact1;

        Contact newContact2 = new Contact(FirstName='Jane',LastName =
'Deo',AccountId=newAccount.Id);
        insert newContact2;

        List<Id>accountIds = new List<Id>();
        accountids.add(newAccount.Id);

        Test.startTest();
        Accountprocessor.countContacts(accountIds);
        Test.stopTest();

    }
}

```

## Use Batch Apex

```

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_list){
        List<lead> L_list_new= new List<lead>();
        for(lead L:L_list){
            L.leadsource='Dreamforce';
            L_list_new.add(L);
        }
    }
}

```

```

        count+=1;
    }
    update L_list_new;
}
global void finish(Database.BatchableContext bc){
    system.debug('count='+count);
}
}

```

```

@isTest
public class LeadProcessorTest {

    @isTest
    public static void testit(){
        List<lead> L_list = new List<lead>();

        for(Integer i=0;i<200;i++){
            Lead L = new lead();
            L.LastName = 'name' +1;
            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();

    }

}

```

# Control Processes with Queueable Apex

```
public class AddPrimaryContact implements Queueable{

    private Contact con;
    private String state;

    public AddPrimaryContact(Contact con,String state){
        this.con=con;
        this.state=state;
    }

    public void execute(QueueableContext context){
        List<Account> accounts=[Select Id , Name , (Select FirstName , LastName ,Id from
contacts)
                                from Account where BillingState = :state Limit 200];
        List<Contact> primaryContacts = new List<Contact>();

        for(Account acc:accounts){
            Contact c = con.clone();
            c.AccountId = acc.Id;
            primaryContacts.add(c);
        }

        if(primaryContacts.size() > 0){
            insert primaryContacts;
        }

    }

}

@isTest
public class AddPrimaryContactTest {
```



```

static testmethod void testQueueable(){
    List<Account> testAccounts = new List<Account>();
    for(Integer i=0;i<50;i++){
        testAccounts.add(new Account(Name='Account '+i,BillingState='CA'));
    }
    for(Integer j=0;j<50;j++){
        testAccounts.add(new Account(Name = 'Account '+j,BillingState='NY'));
    }
    insert testAccounts;

    Contact testContact = new Contact(FirstName = 'John',LastName='Doe');
    insert testContact;

    AddPrimaryContact addit = new addPrimaryContact(testContact, 'CA');

    Test.startTest();
    system.enqueueJob(addit);
    Test.stopTest();

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

```

## Schedule Jobs Using the Apex Scheduler

```

global class DailyLeadProcessor implements Schedulable {
    global void execute(SchedulableContext ctx){
        List<lead> leadstoupdate = new List<lead>();
        List<Lead> leads=[Select id From Lead where LeadSource=NULL Limit 200];

        for(Lead l:leads){
            l.LeadSource= 'Dreamforce';
            leadstoupdate.add(l);
        }
    }
}

```

```

        update leadstoupdate;
    }

}

```

```

@Test
private class DailyLeadProcessorTest {

    public static String CRON_EXP = '0 0 0 15 3 7 2022';
    static testmethod void testScheduledJob(){
        List<Lead> leads = new List<lead>();
        for(Integer i=0;i<200;i++){
            Lead l =new Lead(
                FirstName = 'First ' +i,
                LastName ='LastName',
                Company ='The Inc'
            );
            leads.add(l);
        }
        insert leads;

        Test.startTest();

        String jobId = System.schedule('ScheduledApexTest',CRON_EXP,new
DailyLeadProcessor());
        Test.stopTest();

        List<Lead> checkleads = new List<Lead>();
        checkleads = [Select Id From Lead Where LeadSource = 'Dreamforce' and Company
= 'The Inc'];

        System.assertEquals(200,checkleads.size(), 'Leads were not created');
    }

}

```

# Apex Integration Services

## Apex REST Callouts

```
public class AnimalLocator{
    public static String getAnimalNameById(Integer x){
        Http http = new Http();
        HttpRequest req = new HttpRequest();
        req.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/' + x);
        req.setMethod('GET');
        Map<String, Object> animal= new Map<String, Object>();
        HttpResponse res = http.send(req);
        if (res.getStatusCode() == 200) {
            Map<String, Object> results = (Map<String,
Object>)JSON.deserializeUntyped(res.getBody());
            animal = (Map<String, Object>) results.get('animal');
        }
        return (String)animal.get('name');
    }
}
```

```
@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 );
    }
}
```

```
@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('{"animals": ["majestic badger", "fluffy bunny", "scary bear",
"chicken", "mighty moose"]}');
    response.setStatusCode(200);
    return response;
}
}

```

## Apex SOAP Callouts

```

public class ParkLocator {
    public static String[] country(String theCountry) {
        ParkService.ParksImplPort parkSvc = new ParkService.ParksImplPort(); // remove
space
        return parkSvc.byCountry(theCountry);
    }
}

```

```

@Test
private class ParkLocatorTest {
    @Test static void testCallout() {
        Test.setMock(WebServiceMock.class, new ParkServiceMock ());
        String country = 'United States';
        List<String> result = ParkLocator.country(country);
        List<String> parks = new List<String>{'Yellowstone', 'Mackinac National Park',
'Yosemite'};
        System.assertEquals(parks, result);
    }
}

```

```

@Test
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) {
    // start - specify the response you want to send
    ParkService.byCountryResponse response_x = new
ParkService.byCountryResponse();
    response_x.return_x = new List<String>{'Yellowstone', 'Mackinac National Park',
'Yosemite'};
    // end
    response.put('response_x', response_x);
}
}

```

## Apex Web Services

```

@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');
        Account result = [SELECT Id, Name, (Select Id, Name from Contacts) from Account
where Id=:accountId Limit 1];
        return result;
    }
}

```

```

@IsTest

```

```

private class AccountManagerTest {
    @isTest static void testGetContactsByAccountId(){
        Id recordId = createTestRecord();
        RestRequest request = new RestRequest();
        request.requestUri =
'https://yourInstance.my.salesforce.com/services/apexrest/Accounts/'
            + recordId+'/contacts';
        request.httpMethod = 'GET';
        RestContext.request = request;
        Account thisAccount = AccountManager.getAccount();
        System.assert(thisAccount != null);
        System.assertEquals('Test record', thisAccount.Name);
    }

    static Id createTestRecord(){
        Account accountTest = new Account(
            Name ='Test record');
        insert accountTest;

        Contact contactTest = new Contact(
            FirstName='John',
            LastName = 'Doe',
            AccountId = accountTest.Id
        );
        insert contactTest;

        return accountTest.Id;
    }
}

```

## Apex Specialist

### challenge 2

```

trigger MaintenanceRequest on Case (before update, after update) {

```

```

//ToDo: Call MaintenanceRequestHelper.updateWorkOrders
if(trigger.isAfter){
    MaintenanceRequestHelper.updateWorkOrders();
}
}

```

```

public with sharing class MaintenanceRequestHelper {
    public static void updateworkOrders(List<Case> updWorkOrders, Map<Id,Case>
nonUpdCaseMap) {
        Set<Id> validIds = new Set<Id>();
        For (Case c : updWorkOrders){
            if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){
                if (c.Type == 'Repair' || c.Type == 'Routine Maintenance'){
                    validIds.add(c.Id);
                }
            }
        }
    }
}

```

//When an existing maintenance request of type Repair or Routine Maintenance is closed,

```

//create a new maintenance request for a future routine checkup.
if (!validIds.isEmpty()){
    Map<Id,Case> closedCases = new Map<Id,Case>([SELECT Id, Vehicle__c,
Equipment__c, Equipment__r.Maintenance_Cycle__c,
(SELECT Id,Equipment__c,Quantity__c FROM
Equipment_Maintenance_Items__r)
FROM Case WHERE Id IN :validIds]);
    Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();

```

//calculate the maintenance request due dates by using the maintenance cycle defined on the related equipment records.

```

AggregateResult[] results = [SELECT Maintenance_Request__c,
MIN(Equipment__r.Maintenance_Cycle__c)cycle
FROM Equipment_Maintenance_Item__c
WHERE Maintenance_Request__c IN :ValidIds GROUP BY
Maintenance_Request__c];

```

```

        for (AggregateResult ar : results){
            maintenanceCycles.put((Id) ar.get('Maintenance_Request__c'), (Decimal)
ar.get('cycle'));
        }

        List<Case> newCases = new List<Case>();
        for(Case cc : closedCases.values()){
            Case nc = new Case (
                ParentId = cc.Id,
                Status = 'New',
                Subject = 'Routine Maintenance',
                Type = 'Routine Maintenance',
                Vehicle__c = cc.Vehicle__c,
                Equipment__c =cc.Equipment__c,
                Origin = 'Web',
                Date_Reported__c = Date.Today()
            );

            //If multiple pieces of equipment are used in the maintenance request,
            //define the due date by applying the shortest maintenance cycle to today's
            date.
            If (maintenanceCycles.containsKey(cc.Id)){
                nc.Date_Due__c = Date.today().addDays((Integer)
maintenanceCycles.get(cc.Id));
            } else {
                nc.Date_Due__c = Date.today().addDays((Integer)
cc.Equipment__r.maintenance_Cycle__c);
            }

            newCases.add(nc);
        }

        insert newCases;

        List<Equipment_Maintenance_Item__c> clonedList = new
List<Equipment_Maintenance_Item__c>();

```



```

        for (Case nc : newCases){
            for (Equipment_Maintenance_Item__c clonedListItem :
closedCases.get(nc.ParentId).Equipment_Maintenance_Items__r){
                Equipment_Maintenance_Item__c item = clonedListItem.clone();
                item.Maintenance_Request__c = nc.Id;
                clonedList.add(item);
            }
        }
        insert clonedList;
    }
}
}

```

## challenge 3

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

//Write a 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(){
        System.debug('go into runWarehouseEquipmentSync');
        Http http = new Http();
        HttpRequest request = new HttpRequest();

        request.setEndpoint(WAREHOUSE_URL);
        request.setMethod('GET');
        HttpResponse response = http.send(request);

        List<Product2> product2List = new List<Product2>();
        System.debug(response.getStatusCode());
        if (response.getStatusCode() == 200){

```

```

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

        //class maps the following fields:
        //warehouse SKU will be external ID for identifying which equipment records to
update within Salesforce
        for (Object jR : jsonResponse){
            Map<String,Object> mapJson = (Map<String,Object>)jR;
            Product2 product2 = new Product2();
            //replacement part (always true),
            product2.Replacement_Part__c = (Boolean) mapJson.get('replacement');
            //cost
            product2.Cost__c = (Integer) mapJson.get('cost');
            //current inventory
            product2.Current_Inventory__c = (Double) mapJson.get('quantity');
            //lifespan
            product2.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
            //maintenance cycle
            product2.Maintenance_Cycle__c = (Integer)
mapJson.get('maintenanceperiod');
            //warehouse SKU
            product2.Warehouse_SKU__c = (String) mapJson.get('sku');

            product2.Name = (String) mapJson.get('name');
            product2.ProductCode = (String) mapJson.get('_id');
            product2List.add(product2);
        }

        if (product2List.size() > 0){
            upsert product2List;
            System.debug('Your equipment was synced with the warehouse one');
        }
    }
}

public static void execute (QueueableContext context){
    System.debug('start runWarehouseEquipmentSync');

```

```

    runWarehouseEquipmentSync();
    System.debug('end runWarehouseEquipmentSync');
}

}

```

## challenge 4

```

global with sharing class WarehouseSyncSchedule implements Schedulable{
    global void execute(SchedulableContext ctx){
        System.enqueueJob(new WarehouseCalloutService());
    }
}

```

## challenge 5

```

@isTest
public with sharing class MaintenanceRequestHelperTest {

    // createVehicle
    private static Vehicle__c createVehicle(){
        Vehicle__c vehicle = new Vehicle__C(name = 'Testing Vehicle');
        return vehicle;
    }

    // createEquipment
    private static Product2 createEquipment(){
        product2 equipment = new product2(name = 'Testing equipment',
                                            lifespan_months__c = 10,
                                            maintenance_cycle__c = 10,
                                            replacement_part__c = true);
        return equipment;
    }

    // createMaintenanceRequest

```

```

private static Case createMaintenanceRequest(id vehicleId, id equipmentId){
    case cse = new case(Type='Repair',
        Status='New',
        Origin='Web',
        Subject='Testing subject',
        Equipment__c=equipmentId,
        Vehicle__c=vehicleId);
    return cse;
}

// createEquipmentMaintenanceItem
private static Equipment_Maintenance_Item__c createEquipmentMaintenanceItem(id
equipmentId,id requestId){
    Equipment_Maintenance_Item__c equipmentMaintenanceItem = new
Equipment_Maintenance_Item__c(
    Equipment__c = equipmentId,
    Maintenance_Request__c = requestId);
    return equipmentMaintenanceItem;
}

@isTest
private static void testPositive(){
    Vehicle__c vehicle = createVehicle();
    insert vehicle;
    id vehicleId = vehicle.Id;

    Product2 equipment = createEquipment();
    insert equipment;
    id equipmentId = equipment.Id;

    case createdCase = createMaintenanceRequest(vehicleId,equipmentId);
    insert createdCase;

    Equipment_Maintenance_Item__c equipmentMaintenanceItem =
createEquipmentMaintenanceItem(equipmentId,createdCase.id);
    insert equipmentMaintenanceItem;

```

```
test.startTest();
createdCase.status = 'Closed';
update createdCase;
test.stopTest();
```

```
Case newCase = [Select id,
                  subject,
                  type,
                  Equipment__c,
                  Date_Reported__c,
                  Vehicle__c,
                  Date_Due__c
                from case
                where status ='New'];
```

```
Equipment_Maintenance_Item__c workPart = [select id
                                           from Equipment_Maintenance_Item__c
                                           where Maintenance_Request__c =:newCase.Id];
```

```
list<case> allCase = [select id from case];
system.assert(allCase.size() == 2);
```

```
system.assert(newCase != null);
system.assert(newCase.Subject != null);
system.assertEquals(newCase.Type, 'Routine Maintenance');
SYSTEM.assertEquals(newCase.Equipment__c, equipmentId);
SYSTEM.assertEquals(newCase.Vehicle__c, vehicleId);
SYSTEM.assertEquals(newCase.Date_Reported__c, system.today());
}
```

```
@isTest
```

```
private static void testNegative(){
    Vehicle__C vehicle = createVehicle();
    insert vehicle;
    id vehicleId = vehicle.Id;
```

```
product2 equipment = createEquipment();
insert equipment;
```

```
id equipmentId = equipment.Id;
```

```
case createdCase = createMaintenanceRequest(vehicleId,equipmentId);  
insert createdCase;
```

```
Equipment_Maintenance_Item__c workP =  
createEquipmentMaintenanceItem(equipmentId, createdCase.Id);  
insert workP;
```

```
test.startTest();  
createdCase.Status = 'Working';  
update createdCase;  
test.stopTest();
```

```
list<case> allCase = [select id from case];
```

```
Equipment_Maintenance_Item__c equipmentMaintenanceItem = [select id  
                                                             from Equipment_Maintenance_Item__c  
                                                             where Maintenance_Request__c = :createdCase.Id];
```

```
system.assert(equipmentMaintenanceItem != null);  
system.assert(allCase.size() == 1);  
}
```

```
@isTest  
private static void testBulk(){  
    list<Vehicle__C> vehicleList = new list<Vehicle__C>();  
    list<Product2> equipmentList = new list<Product2>();  
    list<Equipment_Maintenance_Item__c> equipmentMaintenanceItemList = new  
list<Equipment_Maintenance_Item__c>();  
    list<case> caseList = new list<case>();  
    list<id> oldCaseIds = new list<id>();  
  
    for(integer i = 0; i < 300; i++){  
        vehicleList.add(createVehicle());  
        equipmentList.add(createEquipment());  
    }  
}
```

```
insert vehicleList;  
insert equipmentList;
```

```
for(integer i = 0; i < 300; i++){  
    caseList.add(createMaintenanceRequest(vehicleList.get(i).id,  
equipmentList.get(i).id));  
}  
insert caseList;
```

```
for(integer i = 0; i < 300; i++){  
  
equipmentMaintenanceItem1List.add(createEquipmentMaintenanceItem(equipmentList.  
get(i).id, caseList.get(i).id));  
}  
insert equipmentMaintenanceItem1List;
```

```
test.startTest();  
for(case cs : caseList){  
    cs.Status = 'Closed';  
    oldCaseIds.add(cs.Id);  
}  
update caseList;  
test.stopTest();
```

```
list<case> newCase = [select id  
                      from case  
                      where status = 'New'];
```

```
list<Equipment_Maintenance_Item__c> workParts = [select id  
                                                  from Equipment_Maintenance_Item__c  
                                                  where Maintenance_Request__c in: oldCaseIds];
```

```
system.assert(newCase.size() == 300);
```

```
list<case> allCase = [select id from case];
```

```

        system.assert(allCase.size() == 600);
    }
}

```

## challenge 6

```

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

        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"},{"_id":"55d66226
726b611100aaf742","replacement":true,"quantity":183,"name":"Cooling
Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"_id":"55d66226726b6
11100aaf743","replacement":true,"quantity":143,"name":"Fuse
20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]]');
        response.setStatusCode(200);

        return response;
    }
}

```

```

@Test
private class WarehouseCalloutServiceTest {
    // implement your mock callout test here
    @Test
    static void testWarehouseCallout() {
        test.startTest();
        test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
    }
}

```



```

WarehouseCalloutService.execute(null);
test.stopTest();

List<Product2> product2List = new List<Product2>();
product2List = [SELECT ProductCode FROM Product2];

System.assertEquals(3, product2List.size());
System.assertEquals('55d66226726b611100aaf741',
product2List.get(0).ProductCode);
System.assertEquals('55d66226726b611100aaf742',
product2List.get(1).ProductCode);
System.assertEquals('55d66226726b611100aaf743',
product2List.get(2).ProductCode);
}
}

```

## challenge 7

```

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

        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"}, { "_id": "55d66226
726b611100aaf742", "replacement": true, "quantity": 183, "name": "Cooling
Fan", "maintenanceperiod": 0, "lifespan": 0, "cost": 300, "sku": "100004"}, { "_id": "55d66226726b6
11100aaf743", "replacement": true, "quantity": 143, "name": "Fuse
20A", "maintenanceperiod": 0, "lifespan": 0, "cost": 22, "sku": "100005"}]');
    }
}

```

```

        response.setStatusCode(200);

    }
    return response;
}

```

```

global with sharing class WarehouseSyncSchedule implements Schedulable{
    global void execute(SchedulableContext ctx){
        System.enqueueJob(new WarehouseCalloutService());
    }
}

```

```

@isTest
public with sharing class WarehouseSyncScheduleTest {
    // implement scheduled code here
    //
    @isTest static void test() {
        String scheduleTime = '00 00 00 * * ? *';
        Test.startTest();
        Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
        String jobId = System.schedule('Warehouse Time to Schedule to test', scheduleTime, new
WarehouseSyncSchedule());
        CronTrigger c = [SELECT State FROM CronTrigger WHERE Id =: jobId];
        System.assertEquals('WAITING', String.valueOf(c.State), 'JobId does not match');

        Test.stopTest();
    }
}

```