```
trigger AccountAddressTrigger on Account (before insert, before update) {
   For(Account accountAddress: Trigger.new){
      if(accountAddress.BillingPostalCode !=null &amp;&amp;
accountAddress.Match_Billing_Address__c ==true){
         accountAddress.ShippingPostalCode=accountAddress.BillingPostalCode;
      }
   }

}


trigger ClosedOpportunityTrigger on Opportunity (after insert, after update) {
   List&lt;Task&gt; newtsk = new List&lt;Task&gt;();
   if(trigger.IsAfter &amp;&amp; (trigger.IsInsert || trigger.IsUpdate)){
   for(Opportunity op:Trigger.New){
      if(op.StageName == &#39;Closed Won&#39;){
         Task tsk = new Task();
tsk.Subject = &#39;Follow Up Test Task&#39;;
tsk.WhatId = op.id;
newtsk.add(tsk);
}
}
   }
   if(newtsk.size()&gt;0){
      insert newtsk;
   }
}


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 &lt; date1) { return false; }

//check that date2 is within (&gt;=) 30 days of date1

Date date30Days = date1.addDays(30); //create a date 30 days away from
date1

if( date2 &gt;= 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;
}

}




@isTest
public class TestVerifyDate {
  @isTest static void testOldDate(){
    Date dateTest = VerifyDate.CheckDates(date.today(), date.today().addDays(-1));
```

```
        System.assertEquals(date.newInstance(2022, 4, 31), dateTest);
    }

    @isTest static void testLessThan30Days(){
        Date dateTest = VerifyDate.CheckDates(date.today(), date.today().addDays(20));
        System.assertEquals(date.today().addDays(20), dateTest);
    }

    @isTest static void testMoreThan30Days(){
        Date dateTest = VerifyDate.CheckDates(date.today(), date.today().addDays(31));
        System.assertEquals(date.newInstance(2022, 4, 31), dateTest);
    }

}
```

```
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 == &#39;INVALIDNAME&#39;) { //invalidname is invalid
c.AddError(&#39;The Last Name &quot;&#39;+c.LastName+&#39;&quot; is not allowed
for

DML&#39;);
}


}




}
```

```
@isTest
private class TestRestrictContactByName {

   @isTest static void testInvalidName() {
      //try inserting a Contact with INVALIDNAME
      Contact myConact = new Contact(LastName=&#39;INVALIDNAME&#39;);
      insert myConact;

      // Perform test
      Test.startTest();
      Database.SaveResult result = Database.insert(myConact, false);
      Test.stopTest();
      // Verify
      // In this case the creation should have been stopped by the trigger,
      // so verify that we got back an error.
      System.assert(!result.isSuccess());
      System.assert(result.getErrors().size() &gt; 0);
      System.assertEquals(&#39;Cannot create contact with invalid last name.&#39;,
                  result.getErrors()[0].getMessage());

   }
}



//@isTest
public class RandomContactFactory {
   public static List&lt;Contact&gt; generateRandomContacts(Integer
numContactsToGenerate, String FName) {
      List&lt;Contact&gt; contactList = new List&lt;Contact&gt;();

      for(Integer i=0;i&lt;numContactsToGenerate;i++) {
         Contact c = new Contact(FirstName=FName + &#39; &#39; + i, LastName =
&#39;Contact &#39;+i);
         contactList.add(c);
         System.debug(c);
      }
```

```
        //insert contactList;
        System.debug(contactList.size());
        return contactList;
    }




public class AccountProcessor {

    //Writting the countContacts method and marking it whit the @future label.
    @future
    public static void countContacts(Set&lt;Id&gt; accountIDs) {

        // Creating a list that will contain all those accounts that are referenced through
the accounIDs list.
        List&lt;Account&gt; accounts = [SELECT Id, Number_of_Contacts__c, (SELECT id
FROM Contacts) from Account where id in :accountIDs];

        //Assigment from the total contact number to the Number_of_Contacts__c field
for each account at accounts list.
        for( Account account : accounts ) {
          account.Number_of_Contacts__c = account.contacts.size();
        }

        //Updating all accounts in list
        update accounts;

    }

}




@isTest
public class AccountProcessorTest {
    @isTest
    public static void countContactsTest(){
        //Creating an account and inserting it
        Account account = New Account(Name = &#39;Account Number 1&#39;);
        insert account;

        //Creating some contacts related to the account and inserting them
        List&lt;Contact&gt; contacts = new List&lt;Contact&gt;();
        contacts.add(New Contact(lastname = &#39;Related Contact 1&#39;, AccountId =
account.Id));
        contacts.add(New Contact(lastname = &#39;Related Contact 2&#39;, AccountId =
account.Id));
        contacts.add(New Contact(lastname = &#39;Related Contact 3&#39;, AccountId =
account.Id));
        contacts.add(New Contact(lastname = &#39;Related Contact 4&#39;, AccountId =
account.Id));
        insert contacts;


        //Creating a List with account Ids to pass them throught the
AccountProcessor.countContacts method
        Set&lt;Id&gt; accountIds = new Set&lt;Id&gt;();
        accountIds.add(account.id);

        //Starting Test:
        Test.startTest();

        //Calling the AccountProcessor.countContacts method
        AccountProcessor.countContacts(accountIds);

        //Finishing Test:
        Test.stopTest();
        Account ACC = [SELECT Number_of_Contacts__c FROM Account WHERE id =
:account.Id LIMIT 1];

        //Setting Assert (We have to parse the account.Number_of_Contacts__c
        //to integer to avoid some comparasion error between decimal and integer)
        System.assertEquals( Integer.valueOf(ACC.Number_of_Contacts__c) , 4);
    }


}




global class LeadProcessor implements
Database.Batchable&lt;sObject&gt;, Database.Stateful {

// instance member to retain state across transactions
global Integer recordsProcessed = 0;

global Database.QueryLocator start(Database.BatchableContext bc) {
return Database.getQueryLocator(&#39;SELECT Id, LeadSource FROM Lead&#39;);
}

global void execute(Database.BatchableContext bc, List&lt;Lead&gt; scope){
// process each batch of records
List&lt;Lead&gt; leads = new List&lt;Lead&gt;();
for (Lead lead : scope) {

lead.LeadSource = &#39;Dreamforce&#39;;
// increment the instance member counter
recordsProcessed = recordsProcessed + 1;

}
update leads;
}


global void finish(Database.BatchableContext bc){
System.debug(recordsProcessed + &#39; records processed. Shazam!&#39;);
```

```
}
}




@isTest
public class LeadProcessorTest {
@testSetup
static void setup() {
List&lt;Lead&gt; leads = new List&lt;Lead&gt;();
// insert 200 leads
for (Integer i=0;i&lt;200;i++) {
leads.add(new Lead(LastName=&#39;Lead &#39;+i,
Company=&#39;Lead&#39;, Status=&#39;Open - Not Contacted&#39;));
}
insert leads;
}

static testmethod void test() {
Test.startTest();
LeadProcessor lp = new LeadProcessor();
Id batchId = Database.executeBatch(lp, 200);
Test.stopTest();

// after the testing stops, assert records were updated properly
System.assertEquals(200, [select count() from lead where LeadSource =
&#39;Dreamforce&#39;]);
}
}




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(&#39;this.c = &#39;+this.c+&#39; this.state = &#39;+this.state);

        List&lt;Account&gt; acc_lst = new List&lt;account&gt;([select id, name, BillingState
from
account where account.BillingState = :this.state limit 200]);
        List&lt;contact&gt; c_lst = new List&lt;contact&gt;();
        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;
    }

}




@IsTest
public class AddPrimaryContactTest {

    @IsTest
    public static void testing() {
        List&lt;account&gt; acc_lst = new List&lt;account&gt;();
        for (Integer i=0; i&lt;50;i++) {
            account a = new account(name=string.valueOf(i),billingstate=&#39;NY&#39;);
            system.debug(&#39;account a = &#39;+a);
            acc_lst.add(a);
        }
        for (Integer i=0; i&lt;50;i++) {
            account a = new account(name=string.valueOf(50+i),billingstate=&#39;CA&#39;);
            system.debug(&#39;account a = &#39;+a);
            acc_lst.add(a);
        }
        insert acc_lst;
        Test.startTest();
        contact c = new contact(lastname=&#39;alex&#39;);
        AddPrimaryContact apc = new AddPrimaryContact(c,&#39;CA&#39;);
        system.debug(&#39;apc = &#39;+apc);
        System.enqueueJob(apc);
        Test.stopTest();
        List&lt;contact&gt; c_lst = new List&lt;contact&gt;([select id from contact]);
        Integer size = c_lst.size();
        system.assertEquals(50, size);
    }

}
```

```
global class DailyLeadProcessor implements Schedulable{
    global void execute(SchedulableContext ctx){
        List&lt;Lead&gt; leads = [SELECT Id, LeadSource FROM Lead WHERE LeadSource =
&#39;&#39;];

        if(leads.size() &gt; 0){
            List&lt;Lead&gt; newLeads = new List&lt;Lead&gt;();

            for(Lead lead : leads){
                lead.LeadSource = &#39;DreamForce&#39;;
                newLeads.add(lead);
            }

            update newLeads;
        }
    }
}




@isTest
private class DailyLeadProcessorTest{
    //Seconds Minutes Hours Day_of_month Month Day_of_week optional_year
    public static String CRON_EXP = &#39;0 0 0 2 6 ? 2022&#39;;

    static testmethod void testScheduledJob(){
        List&lt;Lead&gt; leads = new List&lt;Lead&gt;();

        for(Integer i = 0; i &lt; 200; i++){
            Lead lead = new Lead(LastName = &#39;Test &#39; + i, LeadSource =
&#39;&#39;, Company = &#39;Test
Company &#39; + i, Status = &#39;Open - Not Contacted&#39;);
            leads.add(lead);
        }

        insert leads;

        Test.startTest();
        // Schedule the test job
        String jobId = System.schedule(&#39;Update LeadSource to DreamForce&#39;,
CRON_EXP,
new DailyLeadProcessor());

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

```
public class AnimalLocator{
   public static String getAnimalNameById(Integer x){
      Http http = new Http();
      HttpRequest req = new HttpRequest();
      req.setEndpoint(&#39;https://th-apex-http-callout.herokuapp.com/animals/&#39;
+ x);
      req.setMethod(&#39;GET&#39;);
      Map&lt;String, Object&gt; animal= new Map&lt;String, Object&gt;();
      HttpResponse res = http.send(req);
         if (res.getStatusCode() == 200) {
      Map&lt;String, Object&gt; results = (Map&lt;String,
Object&gt;)JSON.deserializeUntyped(res.getBody());
     animal = (Map&lt;String, Object&gt;) results.get(&#39;animal&#39;);
      }
return (String)animal.get(&#39;name&#39;);
   }
}
```

```
@isTest
private class AnimalLocatorTest{
   @isTest static void AnimalLocatorMock1() {
      Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
      string result = AnimalLocator.getAnimalNameById(3);
      String expectedResult = &#39;chicken&#39;;
      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(&#39;Content-Type&#39;, &#39;application/json&#39;);
        response.setBody(&#39;{&quot;animals&quot;: [&quot;majestic badger&quot;,
&quot;fluffy bunny&quot;, &quot;scary bear&quot;,
&quot;chicken&quot;, &quot;mighty moose&quot;]}&#39;);
        response.setStatusCode(200);
        return response;
    }
}
```

```
//Generated by wsdl2apex

public class ParkService {
    public class byCountryResponse {
        public String[] return_x;
        private String[] return_x_type_info = new
String[]{&#39;return&#39;,&#39;http://parks.services/&#39;,null,&#39;0&#39;,&#39;-
1&#39;,&#39;false&#39;};
        private String[] apex_schema_type_info = new
String[]{&#39;http://parks.services/&#39;,&#39;false&#39;,&#39;false&#39;};
        private String[] field_order_type_info = new String[]{&#39;return_x&#39;};
    }
    public class byCountry {
        public String arg0;
        private String[] arg0_type_info = new
String[]{&#39;arg0&#39;,&#39;http://parks.services/&#39;,null,&#39;0&#39;,&#39;1&#39
;,&#39;false&#39;};
        private String[] apex_schema_type_info = new
String[]{&#39;http://parks.services/&#39;,&#39;false&#39;,&#39;false&#39;};
        private String[] field_order_type_info = new String[]{&#39;arg0&#39;};
    }
    public class ParksImplPort {
        public String endpoint_x = &#39;https://th-apex-soap-
service.herokuapp.com/service/parks&#39;;
        public Map&lt;String,String&gt; inputHttpHeaders_x;
        public Map&lt;String,String&gt; 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[]{&#39;http://parks.services/&#39;,
&#39;ParkService&#39;};
        public String[] byCountry(String arg0) {
            ParkService.byCountry request_x = new ParkService.byCountry();
            request_x.arg0 = arg0;
            ParkService.byCountryResponse response_x;
            Map&lt;String, ParkService.byCountryResponse&gt; response_map_x = new
Map&lt;String, ParkService.byCountryResponse&gt;();
            response_map_x.put(&#39;response_x&#39;, response_x);
            WebServiceCallout.invoke(
              this,
              request_x,
              response_map_x,
              new String[]{endpoint_x,
              &#39;&#39;,
              &#39;http://parks.services/&#39;,
              &#39;byCountry&#39;,
              &#39;http://parks.services/&#39;,
              &#39;byCountryResponse&#39;,

              &#39;ParkService.byCountryResponse&#39;}
            );
            response_x = response_map_x.get(&#39;response_x&#39;);
            return response_x.return_x;
        }
    }
}
```

```
@isTest
global class ParkServiceMock implements WebServiceMock {
  global void doInvoke(
         Object stub,
         Object request,
         Map&lt;String, Object&gt; 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&lt;String&gt;{&#39;Yellowstone&#39;,
&#39;Mackinac National Park&#39;,
&#39;Yosemite&#39;};
      // end
      response.put(&#39;response_x&#39;, response_x);
  }
}
```

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

```
@isTest
private class ParkLocatorTest {
   @isTest static void testCallout() {

      Test.setMock(WebServiceMock.class, new ParkServiceMock ());
      String country = &#39;United States&#39;;
      List&lt;String&gt; result = ParkLocator.country(country);
      List&lt;String&gt; parks = new List&lt;String&gt;{&#39;Yellowstone&#39;,
&#39;Mackinac National Park&#39;,
&#39;Yosemite&#39;};
       System.assertEquals(parks, result);
   }
}
```

```
@RestResource(urlMapping=&#39;/Accounts/*/contacts&#39;)
global class AccountManager {
   @HttpGet
   global static Account getAccount() {
      RestRequest req = RestContext.request;
      String accId = req.requestURI.substringBetween(&#39;Accounts/&#39;,
&#39;/contacts&#39;);
      Account acc = [SELECT Id, Name, (SELECT Id, Name FROM Contacts)
                     FROM Account WHERE Id = :accId];
      return acc;
   }
}
```

```
@isTest
```

```
private class AccountManagerTest {

    private static testMethod void getAccountTest1() {
        Id recordId = createTestRecord();
        // Set up a test request
        RestRequest request = new RestRequest();
        request.requestUri =
&#39;https://na1.salesforce.com/services/apexrest/Accounts/&#39;+
recordId +&#39;/contacts&#39; ;
        request.httpMethod = &#39;GET&#39;;
        RestContext.request = request;
        // Call the method to test
        Account thisAccount = AccountManager.getAccount();
        // Verify results
        System.assert(thisAccount != null);
        System.assertEquals(&#39;Test record&#39;, thisAccount.Name);

    }

    // Helper method
        static Id createTestRecord() {

        // Create test record
        Account TestAcc = new Account(
          Name=&#39;Test record&#39;);
        insert TestAcc;
        Contact TestCon= new Contact(
        LastName=&#39;Test&#39;,
        AccountId = TestAcc.id);
        return TestAcc.Id
;
    }
}
```