```
#trigger  RestrictContac
trigger RestrictContactByName on Contact (before insert) {

}
#trigger ClosedOpportunityTrigge

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

   List<Task> taskListToInsert = new List<Task>();

   for(opportunity opp:Trigger.new)
   {
      if(opp.stageName == 'Closed Won')
      {
         Task t = new Task();
         t.Subject = 'follow up Test Task';
         t.WhatId = opp.Id;
         taskListToInsert.add(t);
      }
   }

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

#trigger AccountAddressTrigger

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

   for(Account acct:Trigger.new)
   {
      if(acct.Match_Billing_Address__c == True)
         acct.Shippingpostalcode = acct.Billingpostalcode;
      }
}
```

#DailyLeadProcessor

```
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;
   }

}
```

#class DailyLeadProcessorTest

```
@isTest
private class DailyLeadProcessorTest{
   // Dummy CRON expression: midnight on March 15.
   // Because this is a test, job executes
   // immediately after Test.stopTest().
   public static String CRON_EXP = '0 0 0 15 3 ? 2022';
   static testmethod void testScheduledJob() {
      // Create some out of date Opportunity records
      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();
    // Schedule the test job
    String jobId = System.schedule('ScheduledApexTest',
        CRON_EXP,
        new DailyLeadProcessor());
    Test.stopTest();

    // Now that the scheduled job has executed,
    // check that we have 200 Leads with dewamforce
    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');
  }
}


#AnimalLocator

public class AnimalLocatorById{
  public static String getAnimalNameById(){
    Http http = new Http();
    HttpRequest req = new HttpRequest();
    req.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/');
    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');
   }
}
```

#AnimalLocatorTest

```
@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 );
   }
}
```

#AnimalLocatorMock

```
@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;
   }
}
```