1) **Create an Apex trigger that sets an account's Shipping Postal Code to match the Billing Postal Code if the Match Billing Address option is selected. Fire the trigger before inserting an account or updating an account.**

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

    for(Account account :Trigger.New){

        if(account.Match_Billing_Address__c ==
True){

            account.ShippingPostalCode =
account.BillingPostalCode;

        }

    }

}
```

2) **Create a bulkified Apex trigger that adds a follow-up task to an opportunity if its stage is Closed Won. Fire the Apex trigger after inserting or updating an opportunity**.

```
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 Code

1) For Verify Date Class

```
@isTest
public class TestVerifyDate {

   @isTest static void test1(){
      Date d =
VerifyDate.CheckDates(Date.parse('01/06/2022'),Date.parse(
'01/08/2022'));
      System.assertEquals(Date.parse('01/08/2022'), d);
   }

   @isTest static void test2(){
      Date d =
VerifyDate.CheckDates(Date.parse('01/06/2022'),Date.parse(
'03/08/2022'));
      System.assertEquals(Date.parse('01/31/2022'), d);
   }
}
```

2) RestrictContactByName- Trigger

```
@isTest
public class TestRestrictContactByName {

   @isTest
   public 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());
   }
}
```

3) Create a Contact Test Factory

```
public class RandomContactFactory {
  ublic static List<Contact> generateRandomContacts(Integer
num,String lastName){
      List<Contact> contactList = new List<Contact>();
      for(Integer i = 1;i<=num; i++){
        Contact ct = new Contact(FirstName = 'Test' +i,
LastName = lastName);
        contactList.add(ct);
      }
      return contactList;
   }
}
```

4) Create an Apex class that uses the @future annotation to update Account records

A) AccountProcessor

```
public class AccountProcessor {
  @future
  public static void countContacts(List<id> accountIds)
  {
     List<Account> accList = [select id,Number_of_Contacts__c ,
(select id from contacts ) from account where id in :accountIds
];
     for( Account acc : accList )
     {
          acc.Number_of_Contacts__c = acc.Contacts.size();
     }
     update accList;
  }
}
```

B) AccountProcessorTest

```
@IsTest
public class AccountProcessorTest {
   public static testmethod void TestAccountProcessorTest()
   {
      Account a = new Account();
      a.Name = 'Test Account';
      Insert a;

      Contact cont = New Contact();
      cont.FirstName ='Binary';
      cont.LastName ='Programming';
      cont.AccountId = a.Id;
      insert cont;

      List<Id> accListId = new List<ID>();
```

```
        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);
  }
  }
```

Create an Apex class that uses Batch Apex to update Lead records.

a)LeadProcessor

```
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);
    }
}
```

b)LeadProcessorTest

```
@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'+ 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();
   }
}
```

c) Add Primary Contact

```
public class AddPrimaryContact implements Queueable {
   public contact con;
   public 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;
      }

   }

}
```

d) Add Primary Contact Test

```
@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);

}

}
```

e) 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;
      }
}
```

f)DailyLeadProcessorTest

```
@isTest
private class DailyLeadProcessorTest{
   public static String CRON_EXP = '0 0 0 2 6 ? 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');

    }
    }
```

g) AnimalLocator

```
public class AnimalLocator {
   public static String getAnimalNameById(Integer id)    {

      Http http = new Http();
      HttpRequest request = new HttpRequest();
      request.setEndpoint('https://th-apex-http-
callout.herokuapp.com/animals/'+id);
      request.setMethod('GET');       HttpResponse response =
http.send(request);
      String strResp = '';
      system.debug('******response
'+response.getStatusCode());
      system.debug('******response '+response.getBody());
            if (response.getStatusCode() == 200)        {
               Map<String, Object> results = (Map<String, Object>)
JSON.deserializeUntyped(response.getBody());
                Map<string,object> animals = (map<string,object>)
results.get('animal');
               System.debug('Received the following animals:' +
animals );

               strResp = string.valueof(animals.get('name'));
               System.debug('strResp >>>>>>' + strResp );
            }
      return strResp ;
   }
 }
```

h) AnimalLocator Test

```
@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);
   }
}
```

i) AnimalLocator MockTest

```
@isTest
 global class AnimalLocatorMock implements HttpCalloutMock {
   global HTTPResponse respond(HTTPRequest request) {

         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;
   }
}
```

j) ParkLocator

```
public class ParkLocator {

   public static String [] country (String x) {

      String parks = x;
      ParkService.ParksImplPort findCountries = new
ParkService.ParksImplPort ();
      return findCountries.byCountry (parks);
   }
}
```

k) ParkLocatorTest

```
@isTest

public class ParkLocatorTest {
   @isTest
static void testCallout () {
      Test.setMock (WebServiceMock.class, new
ParkServiceMock ());
      String x ='Yellowstone';
      List <String> result = ParkLocator.country(x);

      string resultstring = string.join (result,',');
      System.assertEquals ('USA', resultstring);
   }
}
```

l) ParkServiceMock

```
@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 ();

            response_x.return_x = new List <String> {'USA'};
       response.put ('response_x', response_x);
  }

}
```

m) AccountManager

```
@RestResource(urlMapping='/Accounts/*/contacts')
global with sharing class AccountManager{

   @Http
Getglobal static Account getAccount(){

      RestRequest req = RestContext.request;

      String accId =
req.requestURI.substringBetween('Accounts/', '/contacts');

      Account acc = [SELECT Id, Name, (SELECT Id, Name FROM
Contacts)
                  FROM Account WHERE Id = :accId];
      return acc;

   }

}
```

m)AccountManagerTest

```
@IsTest
private class AccountManagerTest{

   @isTest static void testAccountManager(){

      Id recordId = getTestAccountId();
      RestRequest request = new RestRequest();
      request.requestUri =
      request.httpMethod = 'GET';
      RestContext.request = request;

      Account  acc = AccountManager.getAccount();

      System.assert(acc != null);
   }

   private static Id getTestAccountId(){
      Account acc = new Account(Name = 'TestAcc2');
      Insert acc;

      Contact con = new Contact(LastName = 'TestCont2',
AccountId = acc.Id);
      Insert con;
       return acc.Id;
   }
}
```