Part 2:

Coding Test #1:

Apex Fix all the issues you find with the trigger:

Question 1

```
For(Opportunity p : Trigger.New){
 List acc = [Select Id, CustomField__c FROM Account where id=:p.AccountId];
 if(p.CloseDate < TODAY()){
p.StageName='Closed';
 p.RecordType='renewal';
 }
p.CustomField__c= acc.CustomField__c;
}
```

Answer:

Here the trigger looks like we are trying to close all the related old opportunities of an account whenever a new opportunity is updated. So the code should look like following:

```
trigger OpptyTrigger on Opportunity (before update) {

  set <Id> accountIds = new set<Id>(); //get all account ids
  Map <Id, Account>accountIdsMap = new Map<Id,Account>();
   for(Opportunity oppty  : trigger.new ){

      accountIds.add(oppty.AccountId);
                          }

   Date todaysDate =   System.today(); // This gives today's date
   List<Opportunity> oppListToClose = [SELECT Id FROM Opportunity WHERE AccountId IN: accountIds
                              AND StageName !='Closed' AND CloseDate <: todaysDate];
```

```
    for(Opportunity opty : oppListtoClose){
       opty.StageName='Closed'; //  standard values like Closed/Won or Closed/Lost should be used
      opty.RecordTypeId= ' 0125e000000DLNq ';
      opty.Shipping_Address__c= accountIdsMap.get(opty.AccountId).Shipping_Address__c;

}
}
```

Question 2

The below trigger should fire on record update and prevent a duplicate lead to be entered based on the lead email. Make the necessary changes and follow the coding best practices to make it work

```
Trigger PreventDupLead on Lead(before update) {
 for(Lead aLead : Trigger.new) {
// Find dups based on the email
 List leadList = [SELECT Id FROM Lead WHERE Email =: aLead.Email];
if(leadList.size() > 0) {
 aLead.addError('Duplicate Lead!');
}
 }
 }
```

Don’t forget your test class!

Answer:

1.Trigger

```
trigger PreventDupLead on Lead (before update) {

   //create set to store email ids of all records that are being created.
   set <String> leadEmailSet = new set<String>();
```

```
   // Here we are iterating through each lead record, then we will add email id of each lead to the set.
   for(Lead aLead : trigger.new)
   {
      leadEmailSet.add(aLead.Email);

    }
   // here the blank list will hold LeadId of existing leads.
   List <Lead> leadList = new List <Lead>();

   /** Using SOQL, above list will now contain ids of existing records whose email matches with the
email of the records being created.  **/
   leadList = [SELECT Email FROM Lead WHERE Email IN : leadEmailSet];

   // Iterate through each record that's being created and display error. If the size of the list is greater
than 0 then it's the duplicate record.
   for (Lead lLead : trigger.new)
   {

      if(leadList.size()> 0)
      {
         lLead.addError('Duplicate Lead');

      }

   }

}
```

Test Class:

```
@isTest
public class test_PreventDupLead {
   public static void testLeadDupe()
   {
      Lead myLead = new Lead (LastName = 'Test', Email = 'testLead@gmail.com',Company = 'TestL company');
      insert myLead;
            Lead myDupeLead = new Lead (LastName = 'Test data', Email = 'testLead@gmail.com',Company = 'TestL company');
      insert myDupeLead;

                  Test.startTest();
                  myDupeLead.Email = myLead.Email;
                  myDupeLead.mobilephone = myLead.mobilephone;
                  Database.SaveResult result = Database.update(myDupeLead, false);
                  System.assert(result.getErrors()[0].getMessage().contains('Duplicate Lead'));
                  Test.stopTest();
   }
}
```

Coding Test #2: **Apex**

Write a trigger that will prevent each user from creating more than 100 Events a month.

● The maximum number of Events per user should be configurable without code.

● Each user will have the same maximum.

● Users should see the following error message if they break their max: “Too many Events created this month for <<Name>> (<<User ID>>): <<Maximum>>”

Answer:

Here I have created Trigger Handler and Trigger.

Trigger Handler:

```
public class eventTriggerHandler {

    public static void limitNoOfEvents(List<Event>eventList)

    {

   Integer monthNumber = Date.today().Month();
   Integer yearNumber  = Date.today().Year();
   Integer maxEvents  = 100;

       List<Event>eventListForThisMonth= [Select Id, CreatedById, CreatedDate from Event where
CALENDAR_YEAR(CreatedDate)=:yearNumber
                                    and CALENDAR_MONTH(CreatedDate)=:monthNumber and CreatedById
=:UserInfo.getUserId()];

       If(eventListForThisMonth.Size()>maxEvents)
          eventList[0].addError('Too many Events created this month for user' +Event.CreatedById );

    }

}
```

Trigger:

```
trigger eventTrigger on Event (before insert) {
  if(trigger.isInsert && trigger.isBefore )
  eventTriggerHandler.limitNoOfEvents(trigger.new);
}
```

Coding Test #5: Apex + Trigger

We have 2 objects

- Bonus - this has 2 fields Start & End Date

- Bonus Payouts - Date & Lookup to Bonus

For every entry in Bonus: - We need to create entry in Bonus Payout object for every month between start and end date

So for Example - Start Date is Jan 15 & End Date is Jun10

- There should be a record for Jan,Feb,March,April,May,June .

Start Date is Jan15 & End Date is Jun1

-There should be a record for Jan,Feb,March,April,May

Set up daily batch process that takes Bonus records to create Bonus Payouts Don't forget your test class! Test code coverage test the edge case

Batch Class:

```
global class bonusBatchClass implements Database.Batchable<SObject>{

   list<Bonus__c>bonusList = new list<Bonus__c>();

   global Database.QueryLocator start(Database.BatchableContext BC)
   {
      //String query= 'SELECT Id, Name, Start_Date__c, End_Date__c FROM Bonus__c WHERE Start_Date__c=TODAY LIMIT 1000';

      return Database.getQueryLocator([SELECT Id, Name, Start_Date__c, End_Date__c FROM Bonus__c WHERE Start_Date__c=TODAY]) ;
```

```
    }

    global void execute(Database.BatchableContext BC, list<Bonus__c>bonusList)
    {

        //To create Bonus Payout records
       list<Bonus_Payouts__c>bonusPayoutList = new list<Bonus_Payouts__c>();
       for(Bonus__c b :bonusList)
       {
          Bonus_Payouts__c bPayouts = new  Bonus_Payouts__c();
          bPayouts.Name    = b.Name + '';
          bPayouts.Date__c = Date.today();
          bPayouts.Bonus__c= b.Id;
          bonusPayoutList.add(bPayouts);
       }
      insert bonusPayoutList;

    }
    global void finish(Database.BatchableContext BC)
    {

    }

}
```

Schedule class:

```
global class schedular_bonus implements Schedulable  {
```

```
    global void execute(SchedulableContext sc)

    {

        bonusBatchClass bBatch = new bonusBatchClass();
        database.executeBatch(bBatch);
    }

}
```

(Please refer daily schedule in scheduled jobs)