**1) Create ‘Account’ object record based on email sent to the Salesforce. Email will contain the account details. [Hint: Use Email Services]**

**Email sample-**

**Name: Acme Industries**

**Type: Prospect**

**Phone: (212) 555-5555**

**Employees: 680**

**etc…  
global class AccountCreate implements Messaging.InboundEmailHandler**

**{**

**global Messaging.InboundEmailResult handleInboundEmail(Messaging.InboundEmail email, Messaging.InboundEnvelope envelope)**

**{**

**Messaging.InboundEmailResult result = new Messaging.InboundEmailresult();**

**String Nam= email.subject;**

**// From below we can create account by email services**

**account acc = new account (name=Nam);**

**acc.Type='Prospect';**

**acc.phone= '(212) 555-5555';  
 acc .Employees= ‘680’;**

**insert acc;**

**result.success=true;**

**return result;**

**}**

**}  
  
2) Client wants only upto 10 records to be created on the Account object daily. Write code which will run every 15 minutes. It will check for Account object records created in last 24 hour period (12 am to 12 pm). If the count exceeds 10, delete the additional records. Counter will reset at 12 am to 0.  
  
global class AccountSchedulable implements schedulable**

**{**

**global void execute(SchedulableContext SC)**

**{**

**integer a = 0;**

**list<Account> listAcc = new list<Account>();**

**list<Account> lstAccDelete = new list<Account>();**

**for(Integer i=1;i<=10;i++)**

**{**

**Account Acc= new Account();**

**Acc.Name = 'TestNov ' + i;**

**listAcc.Add(Acc);**

**}**

**insert listAcc;**

**list<Account> extraAcc = [select Id,Name,CreatedDate From Account where CreatedDate = Today order By CreatedDate DESC];**

**For(Account Acc : extraAcc){**

**a++;**

**system.debug('How Many Records Inserted Today ' + a);**

**if(a>=11){**

**lstAccDelete.Add(Acc);**

**}**

**}**

**delete lstAccDelete;**

**list<Account> Acc = [select Id,Name,CreatedDate From Account where CreatedDate = Today order By CreatedDate DESC];**

**system.debug('How Many Records Deleted' + Acc);**

**}**

**}**

**/\*Code for scheduling the jobs first time**

**AccountSchedulable sch = new AccountSchedulable();**

**System.schedule('Extra Account Deletion: at 00 mins', '0 0 \* \* \* ?', sch);**

**System.schedule('Extra Account Deletion: at 15 mins', '0 15 \* \* \* ?', sch);**

**System.schedule('Extra Account Deletion: at 30 mins', '0 30 \* \* \* ?', sch);**

**System.schedule('Extra Account Deletion: at 45 mins', '0 45 \* \* \* ?', sch);**

**\*/  
3) Create custom object ‘Accounts Delete Log’ with fields: Deleted Account Name, Account Deleted Date, Account Deleted Time. When any record from Account object is deleted, its name & deletion date/ time should be saved in this ‘Accounts Delete Log’ object.  
  
public class AccountDeletionDetails {**

**Public void DeletionOfAccount()**

**{**

**List<Account> Actlist =[select id, name from account];**

**List<Accounts\_Delete\_Log\_\_c> delLogList = new List<Accounts\_Delete\_Log\_\_c>();**

**for(Account acc : Actlist)**

**{**

**Accounts\_Delete\_Log\_\_c delLog = new Accounts\_Delete\_Log\_\_c();**

**delLog.Name = acc.Name;**

**delLog.Account\_Deleted\_Date\_\_c = Date.today();**

**delLogList.add(delLog);**

**}**

**INSERT delLogList;**

**System.debug('List of inserted record====================>>>>>>>>>>>>' +delLogList);**

**}**

**}**  
4) Write code for following: After an Account record is created, create a Contact record linked to that Account record. [Restriction: Cannot use apex trigger for this]  
  
answer:=====🡺>>>  
//created class (AccountAndContactInsert) called by process builder ([Apexcallcalledfromprocessbuilder](javascript:void(0);))  
  
public class AccountAndContactInsert {

@InvocableMethod (label='Insert Record')

public static void InsertRecord(List<Id> accIds)

{

List<Contact> conList = new List<Contact>();

List<Account> accList = [Select Id, Name, Number\_of\_Contacts\_\_c from Account where Id =:accIds];

for(Account acc : accList)

{

if(acc.Number\_of\_Contacts\_\_c != null)

{

for(integer i=1;i<=acc.Number\_of\_Contacts\_\_c;i++)

{

Contact con = new Contact();

con.LastName = acc.Name+ 'NK' +i;

con.AccountId = acc.Id;

conList.add(con);

}

}

}

INSERT conList;

}

}   
  
  
  
5) After a Contact record is created, send an email to that person, welcoming them to Salesforce Platform. [Restriction: Cannot use email templates/ alerts for this]  
answer:=====🡺>>> class = >>>>>CreateLeadAssignment  
  
global class CreateLeadAssignment implements Messaging.InboundEmailHandler

{

global Messaging.InboundEmailResult handleInboundEmail(Messaging.inboundEmail email, Messaging.InboundEnvelope env)

{

Messaging.InboundEmailResult result = new Messaging.InboundEmailResult();

String myPlainText= email.plainTextBody;

Lead[] leads = [SELECT Id, Name, Email FROM Lead WHERE Email = :email.fromAddress];

Lead newLead = new Lead(LastName = 'shetty', Company = 'vyom labs', Email = email.fromAddress );

insert newLead;

result.success = true;

return result;

}

}

}

6)Create custom object ‘Complaints’ with appropriate fields. It will have ‘Response Date’ field (Date/time). It will also have ‘Priority’ field (Picklist).

Write a code which will auto-update Response Date field based on Priority using following mapping:

Priority Response Date

Critical Creation Date + 6 Business Hours

High Creation Date + 2 Business Days

Medium Creation Date + 5 Business Days

Low Creation Date + 10 Business Days  
  
  
answer:=====🡺>>> CLASS = >PriorityResponse  
public class PriorityResponse {

public void priorityrr()

{

List<Complaint\_\_c> PR = new List<Complaint\_\_c>();

List<Complaint\_\_c> Prio = [select id, name, Priority\_\_c, Response\_Date\_\_c, CreatedDate from Complaint\_\_c ];

for(Complaint\_\_c Acc:Prio){

if(Acc.Priority\_\_c == 'Critical'){

Acc.Response\_Date\_\_c = Acc.CreatedDate.AddHours(6);

}

else if(Acc.Priority\_\_c == 'High'){

Acc.Response\_Date\_\_c = Acc.CreatedDate.AddDays(2);

}

else if(Acc.Priority\_\_c == 'Medium'){

Acc.Response\_Date\_\_c = Acc.CreatedDate.AddDays(5);

}

else if(Acc.Priority\_\_c == 'Low'){

Acc.Response\_Date\_\_c = Acc.CreatedDate.AddDays(10);

}

datetime myDate = datetime.newInstance(Acc.Response\_Date\_\_c.year(), Acc.Response\_Date\_\_c.month(), Acc.Response\_Date\_\_c.day());

String day = myDate.format('EEEE');

system.debug(day);

if(day == 'Sunday' || day == 'Saturday'){

Acc.Priority\_\_c.AddError('Error');

}

PR.add(Acc);

}

update PR;

}

}  
  
7) Write Test Classes with 80%+ code coverage for the solution code of above questions.  
Answer==========🡺>>>>>>>  
Test class for assignment no – 6  
Prioritytest  
@Istest

public class Prioritytest {

@Istest

public static void Prioritytestresponse()

{

Complaint\_\_c CR = new Complaint\_\_c( Name= 'testpune10', Priority\_\_c = 'Critical');

insert CR;

List<Id> ComplaintIds = new List<Id>();

ComplaintIds.add(CR.Id);

Test.startTest();

PriorityResponse.priorityrr(ComplaintIds);

Test.stopTest();

}

//=======================================================================>>>>>>>>>>>>>>>>>>>>>>

@Istest

public static void Prioritytestresponse1()

{

Complaint\_\_c CR = new Complaint\_\_c( Name= 'testpune10', Priority\_\_c = 'High');

insert CR;

List<Id> ComplaintIds = new List<Id>();

ComplaintIds.add(CR.Id);

Test.startTest();

PriorityResponse.priorityrr(ComplaintIds);

Test.stopTest();

}

//=======================================================================>>>>>>>>>>>>>>>>>>>>>>

@Istest

public static void Prioritytestresponse2()

{

Complaint\_\_c CR = new Complaint\_\_c( Name= 'testpune10', Priority\_\_c = 'Medium');

insert CR;

List<Id> ComplaintIds = new List<Id>();

ComplaintIds.add(CR.Id);

Test.startTest();

PriorityResponse.priorityrr(ComplaintIds);

Test.stopTest();

}

//=======================================================================>>>>>>>>>>>>>>>>>>>>>>

@Istest

public static void Prioritytestresponse3()

{

Complaint\_\_c CR = new Complaint\_\_c( Name= 'testpune10', Priority\_\_c = 'Low');

insert CR;

List<Id> ComplaintIds = new List<Id>();

ComplaintIds.add(CR.Id);

Test.startTest();

PriorityResponse.priorityrr(ComplaintIds);

Test.stopTest();

}

}  
8) Fetch daily weather data from https://openweathermap.org/api for Pune and email it daily at 10 am to your personal email. [Use REST API].  
global class WeatherSchedule1 implements schedulable {

global void execute(SchedulableContext SC)

{

WeatherReport();

}

@future (callout=true)

global static void WeatherReport() {

string requestEndPoint = 'https://api.openweathermap.org/data/2.5/weather?q=pune&appid=4ad168fdd70a96954b81ce20322fdb07';

Http http = new Http();

HttpRequest request = new HttpRequest();

request.setEndPoint(requestEndPoint);

request.setMethod('GET');

HttpResponse response = http.send(request);

if(response.getStatus()=='OK' && response.getStatusCode()==200){

Map<String, Object> results =(Map<String, Object>)JSON.deserializeUntyped(response.getBody());

String report = response.getBody();

system.debug('tested ' +report);

//return report;

Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();

String[] toAddress = new String[] {'Nagnath.kalburge@vyomlabs.com'};

mail.setToAddresses(toAddress);

mail.setReplyTo('Nagnathkalburge15@gmail.com');

mail.setSubject('Dail Weather Report Recived');

mail.setPlainTextBody('Todays Weather Report' +report);

Messaging.sendEmail(new Messaging.SingleEmailMessage[] { mail });

}

}

}

/\*WeatherSchedule1 acc = new WeatherSchedule1();

System.schedule('Weather Report Generated', '0 0 10 \* \* ?', acc);  
\*/

**9) Make a Callout from Trigger. (Consider any open API e.g. currency conversion postal code).**

**============================🡺>>>>>>>>>>>>>>>>>>>>  
public class Apex\_callout {**

**@future(callout=true)**

**Public static void callcount(string zipcode,id ids)**

**{**

**Http http=new Http();**

**HttpRequest req=new HttpRequest();**

**req.setEndpoint('https://api.postalpincode.in/pincode/'+zipcode);//Alandi%20Devachi**

**req.setMethod('GET');**

**HttpResponse res=http.send(req);**

**system.debug('--->>'+res);**

**if(res.getStatus()=='OK' && res.getStatusCode()==200)**

**{**

**system.debug('success===> '+ res.getBody());**

**list <Account> accd=[select id,Description from Account where id=:ids];**

**for (account a : accd )**

**{**

**a.Description= res.getBody();**

**system.debug('success===> '+a.Description);**

**update accd;**

**}**

**}**

**}**

**}  
trigger Pincode\_Address on Account (after insert,after update) {**

**if(!System.isFuture() && !System.isBatch()){**

**for(Account acc:trigger.new){**

**Apex\_callout.callcount(acc.BillingPostalCode,acc.Id);**

**}**

**}**

**}  
============================🡺>>>>>>>>>>>>>>>>>>>>**

10) create a simple REST API in apex with HTTP Request to delete, retrieve and update a record  
@RestResource(urlMapping='/Account/\*')

global with sharing class MyRestResource {

@HttpDelete

global static void doDelete() {

RestRequest req = RestContext.request;

RestResponse res = RestContext.response;

String accountId = req.requestURI.substring(req.requestURI.lastIndexOf('/')+1);

Account account = [SELECT Id FROM Account WHERE Id = :accountId];

delete account;

}

// /services/apexrest/Account/doDelete/0015g00000V02UVAAZ

@HttpGet

global static List<Account> doGet() {

RestRequest req = RestContext.request;

RestResponse res = RestContext.response;

List<Account> result = [SELECT Id, Name, Phone FROM Account];

return result;

}

// /services/apexrest/Account/doGet

@HttpPost

global static String doPost(String name,

String phone, String website) {

Account account = new Account();

account.Name = name;

account.phone = phone;

account.website = website;

insert account;

return account.Id;

}

// /services/apexrest/Account/doPost

/\*{

"name": "Super hero squad",

"phone": "8888490916",

"website": "www.2016.com"}

\*/

@HttpPut

global static ID DoPut(String subject, String status,

String origin, String priority, String id) {

Case thisCase = new Case(

Id=id,

Subject=subject,

Status=status,

Origin=origin,

Priority=priority);

// Match case by Id, if present.

// Otherwise, create new case.

upsert thisCase;

// Return the case ID.

return thisCase.Id;

}

// /services/apexrest/Account/doPut/

/\*

{

"id": "5005g00000FwspEAAR",

"status" : "Working",

"subject" : "Bigfoot Sighting!",

"priority" : "Medium"

}

\*/

@HttpPatch

global static ID doPatch() {

RestRequest req = RestContext.request;

RestResponse res = RestContext.response;

String accountId = req.requestURI.substring(req.requestURI.lastIndexOf('/')+1);

Account account = [SELECT Id FROM Account WHERE Id = :accountId];

Map<String, Object> params = (Map<String, Object>)JSON.deserializeUntyped(req.requestbody.tostring());

// Iterate through each parameter field and value

for(String fieldName : params.keySet()) {

// Set the field and value on the Case sObject

account.put(fieldName, params.get(fieldName));

}

update account;

return account.Id;

}

// /services/apexrest/Account/doPatch/0015g00000V01ntAAB

/\*

{

"name": "nagnathlk",

"phone": "8888490916",

"website": "www.2016.com"}

\*/

}  
  
11) Create a TotalAmount field on account object, and need to sum of the opportunity amount for all the opportunity which is closed won for that account, use batch class.  
answer:=====🡺>>> class = >>>>> AdvanceApexAssignment11  
global class AdvanceApexAssignment11 implements

Database.Batchable<sObject>,Database.Stateful{

global decimal i =0;

//Start method

global Database.QueryLocator start(Database.BatchableContext BC)

{

String query='select id,name,total\_amount\_\_c,(select id, accountId, name from Opportunities) from Account';

return Database.getQueryLocator(query);

}

//Excute method

global void execute(Database.BatchableContext BC, List<Account>scope)

{

system.debug('account list ============>>>>' +scope);

SET<id> opset = new set<id>();

//list<account> newlist = new list<account>();

AggregateResult[] groupedResults = [SELECT AccountId, SUM(Amount) sum

FROM Opportunity WHERE AccountId IN : scope AND Stagename = 'Closed Won' GROUP BY AccountId];

system.debug('opportunity vale============>>>>' +groupedResults);

for (AggregateResult ar : groupedResults) {

i=(decimal)ar.get('sum');

system.debug('value of i ' +i);

}

for (Opportunity op : [select id, accountid, name from opportunity])

{

opset.add(op.AccountId);

}

account act = [select id, name,total\_amount\_\_c from account where id=:opset limit 1];

act.total\_amount\_\_c = i;

system.debug('value of amount' + act.total\_amount\_\_c );

// newlist.add(ac);

system.debug('value of newlist' +act);

update act;

}

global void finish(Database.BatchableContext BC)

{

// Finish logic

system.debug('after finish'+BC.getJobId());

}

**//12) create a custom email to lead functionally to generate a lead.  
  
global class CreateLeadAssignment implements Messaging.InboundEmailHandler**

**{**

**global Messaging.InboundEmailResult handleInboundEmail(Messaging.inboundEmail email, Messaging.InboundEnvelope env)**

**{**

**Messaging.InboundEmailResult result = new Messaging.InboundEmailResult();**

**String myPlainText= email.plainTextBody;**

**Lead[] leads = [SELECT Id, Name, Email FROM Lead WHERE Email = :email.fromAddress];**

**Lead newLead = new Lead(LastName = 'shetty', Company = 'vyom labs', Email = email.fromAddress );**

**insert newLead;**

**result.success = true;**

**return result;**

**}**

**}  
  
// 13) create a custom wrapper class and using that insert a data in account and contact object  
public List<WrapObjecs> wrapLst = new List<WrapObjecs>();**

**public void InsertData()**

**{**

**Account a = new Account(Name ='Wrapper3');**

**Contact c = new Contact(lastName = 'wrapper3');**

**WrapObjecs wrap = new WrapObjecs(a, c);**

**wrap.insertData();**

**}**

**public class WrapObjecs**

**{**

**public Account acc = new Account();**

**public Contact con = new Contact();**

**public WrapObjecs(Account acc1, COntact con1)**

**{**

**acc = acc1;**

**con = con1;**

**}**

**public void insertData()**

**{**

**insert acc;**

**insert con;**

**System.debug('Records inserted');**

**}**

**}**

**}**15) make the invocable class which runs from the process builder to delete a related opportunity when account status is updated to the inactive status."  
answer:=====🡺>>>  
created class (DeleteAccountRelatedOpportunity) called by process builder (DeleteAccountRelatedOpportunity)  
public class DeleteAccountRelatedOpportunity {

@InvocableMethod (label='Delete Oppotunity Record')

public static void DeleteOppotunityRecord(List<Id> accIds)

{

List<Opportunity> cont = new List<Opportunity>([select id, AccountId from Opportunity where AccountId in : accIds]);

delete cont;

}

}