Apex Triggers -

1) Create an Apex Trigger -

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

for(Account a : Trigger.new){
If (a.Match_Billing_Address__c == true) {
a.ShippingPostalCode = a.BillingPostalCode;
}
}

}

```

2) Bulk Apex Triggers -

```
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
}
}
if(taskList.size()>0){
insert taskList;
}
}
```

Apex Triggers -

3)Get Started with Apex Unit Test -

```
@isTest
public class TestVerifyDate {

@isTest static void test1(){
Date d = VerifyDate.CheckDates(Date.parse('01/01/2020'),Date.parse('01/003/2020'));
System.assertEquals(Date.parse('01/03/2020'), d);
}

@isTest static void test2(){
Date d = VerifyDate.checkDates(Date.parse('01/01/2020'),Date.parse('03/03/2020'));
System.assertEquals(Date.parse('01/31/2020'), d);
}
}
```

4) Test Apex Triggers -

```
@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 Lasr Name

}

}
```

5) Create Test Data For Apex Test -

```
public class RandomContactFactory {

public 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;
}

}
```

2) Animal Locator Mock Test:

```
@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,
response.setStatusCode(200);
return response;
}


}
```

3) Animal Locator Test:

```
@isTest
private class AnimalLocatorTest {
@isTest static void AnimalLocatorMock1() {
Test.SetMock(HttpCallOutMock.class, new
AnimalLocatorMock());
string result = AnimalLocator.getAnimalNameById(3);
string expectedresult = 'cow';
System.assertEquals(result, expectedResult);
}

}
```

Apex Class Park Locator Test:
1) Apex Class Park Locator :

```
public class ParkLocator {
public static string[] country(String country) {
parkService.parksImplPort park = new parkService.parksImplPort();
return park.byCountry(country);
}
}
```

2) Apex Class For Park Locator Test :

```
@isTest
private class ParkLocatorTest {
@isTest static void testCallout() {
// This causes a fake response to be generated
Test.setMock(WebServiceMock.class, new ParkServiceMock());
// Call the method that invokes a callout
//Double x = 1.0;
//Double result = AwesomeCalculator.add(x,
y);

String country = 'Germany';
String[] result =
ParkLocator.Country(country);


// Verify that a fake result is returned
System.assertEquals(new
List<String>{'Hamburg Wadden Sea National Park'
,

'Hainich National Park'
,
'Bavarian Forest National

}
}
```

2) Apex Class For Park Locator MockTest :

```
@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) {
// start - specify the response you want to send
parkService.byCountryResponse response_x = new
parkService.byCountryResponse();
response_x.return_x = new List<String>{'Hamburg
'Bavarian Forest National Park'};

//calculatorServices.doAddResponse response_x = new
calculatorServices.doAddResponse();
//response_x.return_x = 3.0;
// end
response.put('response_x'

, response_x);

}
}
```

Apex Web Services :

1) Apex Class Account Manager :

```

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


@HttpGet
global static account getAccount() {

RestRequest request = RestContext.request;

String accountId =
request.requestURI.substring(request.requestURI.las
request.requestURI.lastIndexOf('/'));
List<Account> a = [select id, name, (select
id, name from contacts) from account where id =
:accountId];
List<contact> co = [select id, name from
contact where account.id = :accountId];
system.debug('** a[0]= '+ a[0]);
return a[0];

}

}
```

2) Apex Class Account Manager Test :

```
@istest
public class AccountManagerTest {
@istest static void testGetContactsByAccountId() {
Id recordId = createTestRecord();
// Set up a test request
RestRequest request = new RestRequest();
request.requestUri =
'https://yourInstance.salesforce.com/services/apexrest/Accounts/
request.httpMethod = 'GET';
RestContext.request = request;

Account thisAccount = AccountManager.getAccount();
System.assert(thisAccount!= null);
System.assertEquals('Test record', thisAccount.Name);
}

// Helper method
static Id createTestRecord() {

// Create test record
Account accountTest = new Account(
Name='Test record');
insert accountTest;
Contact contactTest = new Contact(
FirstName='John',
LastName='Doe',
AccountId=accountTest.Id
);
return accountTest.Id;
}
}
```

● Visual Force :

1) Display Image :

```
<apex:page showHeader="false">
<apex:image

url="https://developer.salesforce.com/files/salesforce-developer-
</apex:page>
```

2) Display User Info :

```
<apex:page >
{! $User.FirstName}
</apex:page>
```

3) Contact View :

```
<apex:page standardController="Contact">
<apex:pageBlockSection>
First Name : {! Contact.FirstName}
Last Name: {! Contact.LastName}
Owner Email : {! Contact.Owner.Email}
</apex:pageBlockSection>
</apex:page>
```

4) Opp View :

```
<apex:page standardController="Opportunity">
<apex:outputField value ="{! Opportunity.Name}"/>
<apex:outputField value ="{! Opportunity.Amount}"/>
<apex:outputField value ="{! Opportunity.CloseDate}"/>
<apex:outputField value ="{! Opportunity.Account.Name}"/>
```

```
</apex:page>
```

5) Create Contact :

```
<apex:page standardController="Contact">
<apex:form>
<apex:pageBlockSection>
<apex:inputField value ="{! Contact.FirstName}"/>
<apex:inputField value ="{! Contact.LastName}"/>
<apex:inputField value ="{! Contact.Email}"/>
</apex:pageBlockSection>
<apex:commandButton action="{! save}" value
="Save"/>
</apex:form>
</apex:page>
```

6) Account List :

```

<apex:page standardController="Account" recordSetVar="accounts">
<apex:form>
<apex:pageBlock>
<apex:repeat value="{!Accounts}" id="acccount_list"
rendered="true" var="a">

<li>
<apex:outputLink value="/{!a.id}"/>
<apex:outputText value="{!a.name}"/>

</li>


</apex:repeat>


</apex:pageBlock>
```

```


</apex:form>
</apex:page>
```

7) Show Image :

```
<apex:page >
<apex:image url="{! URLFOR($Resource.vfimagetest,
</apex:page>
```

8) New case List Controller Apex Class :

```
public class NewCaseListController {
public List<Case> getNewCases(){
List<Case> filterList = [Select ID, CaseNumber from Case
where status ='New'];
return filterList;
}
}
```

9) New Case List Visual Force Page :

```
<apex:page controller="NewCaseListController">
<apex:repeat var="case" value="{!newCases}">
<apex:outputLink value="/{!case.ID}">
<apex:outputText
value="{!case.CaseNumber}"></apex:outputText>
</apex:outputLink>
</apex:repeat>
</apex:page>
```

10) Contact Form :

```
<apex:page >
Hello
</apex:page>
```

11) Contact Form :

```
<apex:page standardController="Contact">

<head>
<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-
<title>Quick Start: Visualforce</title>

<!-- Import the Design System style sheet -->
<apex:slds />

</head>
<body>

<apex:form>
<apex:pageBlock title="New Contact">
<!--Buttons –>
<apex:pageBlockButtons>
<apex:commandButton action="{!save}" value="Save"/>
</apex:pageBlockButtons>
<!--Input form -->
<apex:pageBlockSection columns="1">
<apex:inputField value="{!Contact.Firstname}"/>
<apex:inputField value="{!Contact.Lastname}"/>
<apex:inputField value="{!Contact.Email}"/>
</apex:pageBlockSection>
</apex:pageBlock>
</apex:form>
```

```
</body>

</apex:page>
```

Asynchronous Methods :

1) Account Processor Apex Class :

```
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;
}
}
```

2)Account Processor Apex Class Test :

```
@isTest
public class AccountProcessorTest {

```

```
public static testmethod void testAccountProcessor(){

account a = new Account();
a.Name = 'Test Account';
insert a;

Contact con = new Contact();
con.FirstName = 'Binary';
con.LastName = 'Programming';
con.AccountId = a.Id;

insert con;

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);
}

}
```

Use Batch Apex :

1) LeadProcessor Apex Class :

```
global class LeadProcessor implements Database.Batchable<sObject> {
global Integer count = 0;

global Database.QueryLocator start(Database.BatchableContext bc){
return Database.getQueryLocator('SELECT ID,
}

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);
}



}
```

2) LeadProcessor Apex Class Test :

```
@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();
}

}
```

Control Processes With Queueable Apex :

1) AddPrimaryContact Apex Class :

```
public class AddPrimaryContact implements Queueable{

private Contact con;
private 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;
}
}
}
```

2) AddPrimaryContact Apex Class Test :

```
@isTest
public class AddPrimaryContactTest {

static testmethod void testQueueable(){
List<Account> testAccounts = new List<Account>();
for(Integer i=0;i<50;i++){
testAccounts.add(new Account(Name='Account
}
for(Integer j=0;j<50;j++){
testAccounts.add(new Account(Name='Account
}
insert testAccounts;

Contact testContact = new Contact(FirstName ='John', LastName ='Doe');
insert testContact;

AddPrimaryContact addit = new
addPrimaryContact(testContact, 'CA');

Test.startTest();
system.enqueueJob(addit);
Test.stopTest();

System.assertEquals(50,[Select count() from contact where accountId in (Select Id from Account where BillingState='CA')]);
}

}
```

Schedule Jobs Using The Apex Scheduler :

1) DailyLeadProcessor Apex Class :

```
global class DailyLeadProcessor implements Schedulable {
global void execute(SchedulableContext ctx) {
List<Lead> lList = [Select Id, LeadSource from Lead where LeadSource = null];

if(!lList.isEmpty()) {
for(Lead l: lList) {
l.LeadSource = 'Dreamforce';
}
update lList;
}
}

}
```

2) DailyLeadProcessor Apex Class Test :

```
@isTest
public class DailyLeadProcessorTest {
```

```
//Seconds Minutes Hours Day_of_month Month Day_of_week
optional_year
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 lead = new Lead(LastName = 'Test ' + i,
LeadSource = ", Company = 'Test Company ' + i, Status = 'Open -
leads.add(lead);
}

insert leads;

Test.startTest();
// Schedule the test job
String jobId = System.schedule('Update LeadSource to

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

}
```