

# **Build Apex Coding Skills TRAIL**

<https://trailhead.salesforce.com/en/content/learn/trails/build-apex-coding-skills>

## **Apex Basics & Database**

[https://trailhead.salesforce.com/en/content/learn/modules/apex\\_database](https://trailhead.salesforce.com/en/content/learn/modules/apex_database)

# APEX BASICS

Apex is a programming language that uses Java-like syntax

Apex enables developers to add business logic to system events, such as button clicks, updates of related records, and Visualforce pages.

As a language, Apex is:

- Hosted—Apex is saved, compiled, and executed on the server—the Lightning Platform.

- Object oriented—Apex supports classes, interfaces, and inheritance.

- Multitenant aware—Because Apex runs in a multitenant platform, it guards closely against runaway code by enforcing limits, which prevent code from monopolizing shared resources.

- Integrated with the database—It is straightforward to access and manipulate records. Apex provides direct access to records and their fields, and provides statements and query languages to manipulate those records.

## Understanding the Data Types:

The Apex language is strongly typed, that is, every variable in Apex is declared with a specific data type.

All apex variables are initialized to null initially.

Apex variables are also Case-Insensitive.

Apex supports the following data types –

- Primitive (Integer, Double, Long, Date, Datetime, String, ID, or Boolean)
- Collections (Lists, Sets and Maps)
- sObject
- Enums
- Classes, Objects and Interfaces

```
System.debug('Hello World');
```

```
Integer barrelNumbers = 1000;
```

```
System.debug('Value of Barrel Numbers: '+ barrelNumbers);
```

```
barrelNumbers = barrelNumbers + 500;
```

```
System.debug('New value of Barrels: '+ barrelNumbers);
```

```
Integer extraBarrels = 50;
```

```
System.debug('Total barrels: ' + (barrelNumbers + extraBarrels));
```

```
Boolean shipmentDispatched;
```

```
shipmentDispatched = True;
```

```
System.debug('Is shipment dispatched? '+shipmentDispatched);
```

```
Date shipmentDate = Date.today();
```

```
System.debug('Shipment Date: ' + shipmentDate);
```

```
Date newShipmentDate = shipmentDate.addMonths(3);
```

```
System.debug('New Shipment Date: ' + newShipmentDate);
```

```
String companyName = 'Smart Logistics';
```

```
System.debug('Shipment Company Name: ' + companyName);
```

```
Boolean containsLog = companyName.contains('Logis');
```

```
System.debug('Does it contains Logis ' + containsLog);
```

// the following code will insert a new Acc record into database

```
/*
```

```
first
```

```
second
```

```
third
```

```
*/
```

```
Account a = new Account(Name = 'JP Log Acc', Phone = '9191919191');
```

```
System.debug('New Account Info: ' + a);
```

```
insert a;
```

```
Position__c pos = new Position__c(Name = 'Selenium Tester',
```

```
Department__c = 'IT');
```

```
insert pos;
```

```
System.debug('New Position: '+pos);
```

```
Contact c = new Contact(FirstName = 'John', LastName = 'Logistics');
```

```
insert c;
```

```
c.Department = 'Sales';
```

```
update c;
```

```
Contact c = new Contact(FirstName = Jake, LastName = 'Logistics');
```

```
Insert c;
```

```
if (c.Department == null)
```

```
{
```

```
    c.Department = 'Other';
```

```
    Insert c;
```

```
}
```

```
else
```

```
{
```

```
    System.debug('Department '+c.Department);
```

```
}
```

## Collections:

Collections is a type of variable that can store multiple numbers of records. For example, a List can store multiple numbers of Account object's records.

[https://developer.salesforce.com/docs/atlas.en-us.apexref.meta/apexref/apex\\_methods\\_system\\_list.htm](https://developer.salesforce.com/docs/atlas.en-us.apexref.meta/apexref/apex_methods_system_list.htm)

We have three types of collections: List, Set and Map

### List:

Below is the list which contains a list of primitive data types (string), that is a list of cities.

```
List<String> listOfStates = new List<String>{'NY','AR','LN'};
```

```
System.debug('List: '+ listOfStates);
```

Following are some most frequently used methods –

- size()
- add()
- get()
- clear()
- set()



```
listOfStates.add('MH');  
listOfStates.add('TN');
```

```
System.debug('States '+ listOfStates);
```

```
Integer numberOfStates = listOfStates.size();
```

```
System.debug('Number of States: ' + numberOfStates );
```

```
System.debug('State at 3rd position: ' + listOfStates.get(2) );
```

```
System.debug('Does list contains MH: ' + listOfStates.contains('M') );
```

```
listOfStates.set(1, 'AL');
```

```
System.debug('New States: ' + listOfStates);
```

```
System.debug('First State: ' + listOfStates[0]);
```

```
System.debug('2nd State: ' + listOfStates[1]);
```

```
System.debug('3rd State: ' + listOfStates[2]);
```

```
System.debug('4th State: ' + listOfStates[3]);
```

```
System.debug('5th State: ' + listOfStates[4]);
```

## Sets

A Set is a collection type which contains multiple numbers of unordered, unique records. A Set cannot have duplicate records.

[https://developer.salesforce.com/docs/atlas.en-us.apexref.meta/apexref/apex\\_methods\\_system\\_set.htm](https://developer.salesforce.com/docs/atlas.en-us.apexref.meta/apexref/apex_methods_system_set.htm)

Most commonly used methods:

`add()`

`contains()`

`remove()`

`size()`

Example

We will be defining the set of products which company is selling

```
Set<String> products = new Set<String>{'laptop', 'desktop', 'keyboard'};
```

```
System.debug('Set of products: '+products);
```

```
products.add('desktop');
```

```
System.debug('New Set of products: '+products);
```

```
products.remove('keyboard');
```

```
System.debug('Set of products: '+products);
```

# Maps

A Map is a list of key-value pairs, with each pair having a unique key in the map.

Both key and value can be of any data type.

[https://developer.salesforce.com/docs/atlas.en-us.apexref.meta/apexref/apex\\_methods\\_system\\_map.htm#apex\\_System\\_Map\\_methods](https://developer.salesforce.com/docs/atlas.en-us.apexref.meta/apexref/apex_methods_system_map.htm#apex_System_Map_methods)

Example: A Map of Product Code to Product Names :-

```
Map<String, String> pCodeToPName = new Map<String, String>{  
    'one'=>'TV', 'two'=>'Tablet', 'three'=>'Phone'  
};
```

```
System.debug('Product Map: '+pCodeToPName);
```

Most common Map methods:

```
get(key)  
put(key, value)  
remove(key)  
keySet()  
values()  
size()
```

```
String pName = pCodeToPName.get('two');
```

```
System.debug('Product Name for key two: '+pName);
```

```
Set<String> setOfProductKeys = pCodeToPName.keySet();
```

```
System.debug('Set of Keys: '+setOfProductKeys);
```

```
List<String> listOfProducts = pCodeToPName.values();
```

```
System.debug('List of Products: '+listOfProducts);
```

---

```
Account a = new Account(Name = 'Acc1',Phone = '9191919191');  
insert a;  
System.debug('Account details: ' + a);
```

```
List<Account> acclist = new List<Account>();
```

```
Account a1 = new Account( Name = 'Acc2', Phone = '9191919192' );  
Account a2 = new Account( Name = 'Acc3', Phone = '9191919193' );  
Account a3 = new Account( Name = 'Acc4', Phone = '9191919194' );
```

```
acclist.add(a1);  
acclist.add(a2);  
acclist.add(a3);
```

```
insert acclist;
```

```
System.debug('List of Accounts: ' + acclist);
```

Integer i;

```
for(i = 1; i <= 5; i++)  
{  
    System.debug('Value of i is: ' + i);  
}
```

System.debug('Above For loop has ended with value of I: '+i);

System.debug('-----');

```
for(Integer j = 1; j <= 10; j++)  
{  
    if( Math.Mod( j, 2) == 0)  
        System.debug(j + ' is Even');  
    else  
        System.debug(j + ' is Odd');  
}
```

System.debug('Above For loop has ended with value of J: ' + j);

```
List<Account> testAccounts = new List<Account>();
```

```
for(Integer i = 1; i <= 10; i++)  
{  
    Account a = new Account(Name = 'Account ' + i);  
    testAccounts.add(a);  
}
```

```
insert testAccounts;
```

```
Integer j = 1;
```

```
for(Account a: testAccounts)  
{  
    a.Description = 'New Description ' + j;  
    System.debug( j + ' : ' + ' Account ID: ' + a.ID);  
    j++;  
}
```

```
update testAccounts;
```

# SOQL

Standard Examples: [link](#)

```
List<Contact> listOfContacts = [ SELECT FirstName, LastName FROM  
Contact ];
```

```
system.debug( listOfContacts );
```

```
listOfContacts = [ SELECT FirstName, LastName FROM Contact WHERE  
FirstName = 'Stella' ];
```

```
System.debug( listOfContacts );
```

```
String targetDepartment = 'Finance';
```

```
Contact[] listOfContacts = [SELECT FirstName, LastName  
FROM Contact WHERE Department = :targetDepartment];
```

```
System.debug( listOfContacts );
```

```
listOfContacts = [ SELECT Name, Email FROM Contact  
WHERE LastName IN ('James', 'Barr', 'Nedaerk', 'Forbes') ];
```

```
System.debug( listOfContacts );
```



```
listOfContacts = [ SELECT Name, Email FROM Contact  
ORDER BY Name ASC LIMIT 5 ];
```

```
System.debug( listOfContacts );
```

```
// Child to Parent relationship query:
```

```
listOfContacts = [ SELECT Name, Account.Name FROM Contact ];
```

```
System.debug( listOfContacts );
```

```
// Parent to Child relationship query:
```

```
listOfAccounts = [ SELECT Name, (SELECT Name FROM Contacts)  
FROM Account ];
```

```
System.debug( listOfAccounts );
```

```
listOfAccounts =
```

```
[ SELECT Name, Description (SELECT Name FROM Contacts)  
FROM Account  
WHERE Id IN (SELECT AccountId FROM Contact WHERE LastName =  
'Forbes') ];
```

```
System.debug( listOfAccounts );
```

```

for(Account a : listOfAccounts )
{
    a.Description = 'New Description '+j;
    System.debug( j + ' : ' +' Account ID: ' + a.ID);
    j++;
}

```

```

for(Position__c pos: [SELECT Name FROM Position__c] )

{
    System.debug('Positions: '+pos.Name );
}

```

```

AggregateResult[] groupedResults
= [SELECT COUNT(Id) c, Department FROM Contact Group By
Department];

```

```

for(AggregateResult result: groupedResults) {

    System.debug('Count: ' + result.get('c') +' Department ' +
result.get('Department'));

}

```

Most efficient form of Query:

```

for(List<Position__c> positions: [SELECT Name FROM Position__c] )

{
    System.debug('Number of positions: '+positions.size() );
}

```

}

## SOSL

[https://trailhead.salesforce.com/en/content/learn/modules/apex\\_database/apex\\_database\\_sosl](https://trailhead.salesforce.com/en/content/learn/modules/apex_database/apex_database_sosl)

Salesforce Object Search Language (SOSL) is a Salesforce search language that is used to perform text searches in records.

Use SOSL to search fields across multiple standard and custom object records in Salesforce.

Implement custom search functionality: [link](#)

```
List<List<SObject>> searchList =
```

```
[ FIND 'Edge' IN ALL FIELDS  
  RETURNING Account(Name), Contact (FirstName , LastName) ] ;
```

SOQL and SOSL are two separate languages with different syntax. Each language has a distinct use case:

- Use SOQL to retrieve records for a single object.
- Use SOSL to search fields across multiple objects. SOSL queries can search most text fields on an object.

More examples:

[https://developer.salesforce.com/docs/atlas.en-us.soql\\_sosl.meta/soql\\_sosl/sforce\\_api\\_calls\\_sosl\\_examples.htm#sforce\\_api\\_calls\\_sosl\\_examples](https://developer.salesforce.com/docs/atlas.en-us.soql_sosl.meta/soql_sosl/sforce_api_calls_sosl_examples.htm#sforce_api_calls_sosl_examples)



## APEX TRIGGERS

```
trigger newTrigger on Account (before insert, before update) {  
  
    System.debug('Im in a trigger for Account');  
  
}
```

Run the following in Anonymous window (Ctrl + e):

```
Account a = new Account(Name = 'ABC Logistics2');  
  
insert a;
```

```
trigger newTrigger1 on Account (before insert) {  
  
    for(Account a: Trigger.New)  
    {  
        a.Description = 'New Description';  
  
    }  
  
}
```

```
Account a = new Account(Name = 'ABC Logistics2');
```

```
insert a;
```

---

trigger newTriggerCon1 on Contact (before insert, after insert, before update)

```
{  
    if(trigger.isInsert)  
    {  
        if(trigger.isBefore)  
        {  
            System.debug('This is the before event');  
        }  
  
        if(trigger.isAfter)  
        {  
            System.debug('This is the after event');  
        }  
    }  
  
    if(trigger.isUpdate)  
    {  
        System.debug('This is an update event');  
    }  
}
```

---

```
trigger newTriggerCon2 on Contact (before update, after update) {
```

```
    for(Contact c: Trigger.Old)
    {
        System.debug('Before: '+c.Department);
    }
```

```
    for(Contact c: Trigger.New)
    {
        System.debug('After: '+c.Department);
    }
```

```
}
```

---



```
trigger AddRelatedRecord on Account(after insert, after update) {
```

```
    List<Opportunity> oppList = new List<Opportunity>();
```

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

```
    Integer i = 0;
```

```
    for(Account a : Trigger.New) {
```

```
        Contact c = new Contact(firstname = 'John'+i, lastname = 'Doe'+i, AccountId = acc.id);
```

```
        conList.add(c);
```

```
        Date closeD = Date.newInstance(2020, 4, 25);
```

```
        Opportunity opp = new Opportunity(Name = '50 Laptops'+a.Name, CloseDate =  
closeD, stagename = 'Prospecting', AccountId = acc.id);
```

```
        oppList.add(opp);
```

```
        i++;
```

```
    }
```

```
    if (oppList.size() > 0) {
```

```
        insert oppList;
```

```
    }
```

```
    if (conList.size() > 0) {
```

```
        insert conList;
```

```
    }
```

```
}
```

