

APEX IN SALESFORCE

What is Apex?

Apex is a strongly typed, object-oriented programming language that allows developers to execute flow and transaction control statements on Salesforce servers in conjunction with calls to the API. Using syntax that looks like Java and acts like database stored procedures, Apex enables developers to add business logic to most system events, including button clicks, related record updates, and Visualforce pages. Apex code can be initiated by Web service requests and from triggers on objects.

Available in: Salesforce Classic ([not available in all orgs](#)) and Lightning Experience

Available in: **Enterprise, Performance, Unlimited, Developer**, and **Database.com** Editions

Home **Mileage** Contacts

Mileage MF [Back to List](#)

[Customize Page](#) [Edit Layout](#) [Printable View](#) [Help for this Page](#)

[Open Activities Log](#) | [Activity History Log](#)

Mileage Detail [Edit](#) [Delete](#) [Close](#)

Mileage Name: MFR-0002
Date: 2/2/2010
Miles: 20
Contact: [Pat Stumiller](#)
Contact Address: 2 Place Jossieu
Paris, 75251 [Click to Map](#)
Created By: [Richard Castle](#), 2/2/2010 11:09 AM
Last Modified By: [Richard Castle](#), 2/2/2010 11:09 AM

[Edit](#) [Delete](#) [Close](#)

[Traffic](#) [More](#) **Map** [Satellite](#) [Terrain](#)

Code executes when Delete clicked

Code executes when custom link (or button) clicked

Code executes when custom Visualforce extension clicked

Open Activities [New Task](#) [New Event](#)

No records to display

Activity History [Log A Call](#) [Mail Merge](#) [Send An Email](#)

No records to display

[Open Activities Help](#)

[Activity History Help](#)

[Back To Top](#)

Always show me [more](#) records per related list

Understanding Apex Core Concepts

Apex code typically contains many things that you might be familiar with from other programming languages.

Programming elements in Apex



The section describes the basic functionality of Apex, as well as some of the core concepts.

Data Types Overview

Apex supports various data types, including a data type specific to Salesforce—the sObject data type.

Apex supports the following data types.

- A primitive, such as an Integer, Double, Long, Date, Datetime, String, ID, Boolean, among others.
- An sObject, either as a generic sObject or as a specific sObject, such as an Account, Contact, or MyCustomObject__c (you'll learn more about sObjects in a later unit.)
- A collection, including:
 - A list (or array) of primitives, sObjects, user defined objects, objects created from Apex classes, or collections
 - A set of primitives
 - A map from a primitive to a primitive, sObject, or collection
- A typed list of values, also known as an *enum*
- User-defined Apex classes
- System-supplied Apex classes

Using Statements

A *statement* is any coded instruction that performs an action.

In Apex, statements must end with a semicolon and can be one of the following types:

- Assignment, such as assigning a value to a variable
- Conditional (if-else)
- Loops:
 - Do-while
 - While
 - For
- Locking
- Data Manipulation Language (DML)
- Transaction Control
- Method Invoking
- Exception Handling

A *block* is a series of statements that are grouped together with curly braces and can be used in any place where a single statement would be allowed. For example:

Using Collections

Apex has the following types of collections:

- Lists (arrays)
- Maps
- Sets

A *list* is a collection of elements, such as Integers, Strings, objects, or other collections. Use a list when the sequence of elements is important. You can have duplicate elements in a list.

The first index position in a list is always 0.

To create a list:

- Use the `new` keyword
- Use the `List` keyword followed by the element type contained within `<>` characters.

```
1 List<Integer> My_List = new List<Integer>();
```

A *set* is a collection of unique, unordered elements. It can contain primitive data types, such as String, Integer, Date, and so on. It can also contain more complex data types, such as sObjects.

To create a set:

- Use the `new` keyword
- Use the `Set` keyword followed by the primitive data type contained within `<>` characters

Use the following syntax for creating a set:

```
1 Set<datatype> set_name  
2   [= new Set<datatype>();] |  
3   [= new Set<datatype>{value [, value2. . .] };] |  
4   ;
```

The following example creates a set of String. The values for the set are passed in using the curly braces `{}`.

```
1 Set<String> My_String = new Set<String>{'a', 'b', 'c'};
```


A *map* is a collection of key-value pairs. Keys can be any primitive data type. Values can include primitive data types, as well as objects and other collections. Use a map when finding something by key matters. You can have duplicate values in a map, but each key must be unique.

To create a map:

- Use the `new` keyword
- Use the `Map` keyword followed by a key-value pair, delimited by a comma and enclosed in `<>` characters.

Use the following syntax for creating a map:

```
1 Map<key_datatype, value_datatype> map_name
2     [=new Map<key_datatype, value_datatype>();] |
3     [=new Map<key_datatype, value_datatype>
4     {key1_value => value1_value
5     [, key2_value => value2_value. . .}];] |
6     ;
```

The following example creates a map that has a data type of `Integer` for the key and `String` for the value. In this example, the values for the map are being passed in between the curly braces `{}` as the map is being created.

```
1 Map<Integer, String> My_Map = new Map<Integer, String>{1 => 'a', 2 => 'b', 3 => 'c'};
```

Using Loops

While the `if` statement enables your application to do things based on a condition, loops tell your application to do the same thing again and again based on a condition. Apex supports the following types of loops:

- Do-while
- While
- For

A *Do-while* loop checks the condition after the code has executed.

A *While* loop checks the condition at the start, before the code executes.

A *For* loop enables you to more finely control the condition used with the loop. In addition, Apex supports traditional For loops where you set the conditions, as well as For loops that use lists and SOQL queries as part of the condition.

When Should I Use Apex?

The Salesforce prebuilt applications provide powerful CRM functionality. In addition, Salesforce provides the ability to customize the prebuilt applications to fit your organization. However, your organization may have complex business processes that are unsupported by the existing functionality. In this case, Lightning Platform provides various ways for advanced administrators and developers to build custom functionality.

Apex

Use Apex if you want to:

- Create Web services.
- Create email services.
- Perform complex validation over multiple objects.
- Create complex business processes that are not supported by workflow.
- Create custom transactional logic (logic that occurs over the entire transaction, not just with a single record or object).
- Attach custom logic to another operation, such as saving a record, so that it occurs whenever the operation is executed, regardless of whether it originates in the user interface, a Visualforce page, or from SOAP API.

APEX BEST PRACTICES

1. Bulkify your Code
2. Avoid SOQL Queries or DML statements inside FOR Loops
3. Bulkify your Helper Methods
4. Using Collections, Streamlining Queries, and Efficient For Loops
5. Streamlining Multiple Triggers on the Same Object
6. Querying Large Data Sets
7. Use of the Limits Apex Methods to Avoid Hitting Governor Limits
8. Use `@future` Appropriately
9. Writing Test Methods to Verify Large Datasets
10. Avoid Hardcoding IDs

Built-In Exceptions and Common Methods

Apex provides a number of built-in exception types that the runtime engine throws if errors are encountered during execution. You've seen the `DmlException` in the previous example. Here is a sample of some other built-in exceptions. For a complete list of built-in exception types, see [Exception Class and Built-In Exceptions](#).

DmlException

Any problem with a DML statement, such as an `insert` statement missing a required field on a record.

This example makes use of `DmlException`. The `insert` DML statement in this example causes a `DmlException` because it's inserting a merchandise item without setting any of its required fields. This exception is caught in the `catch` block and the exception message is written to the debug log using the `System.debug` statement.

```
1 try {
2     Merchandise__c m = new Merchandise__c();
3     insert m;
4 } catch(DmlException e) {
5     System.debug('The following exception has occurred: ' + e.getMessage());
6 }
```


ListException

Any problem with a list, such as attempting to access an index that is out of bounds.

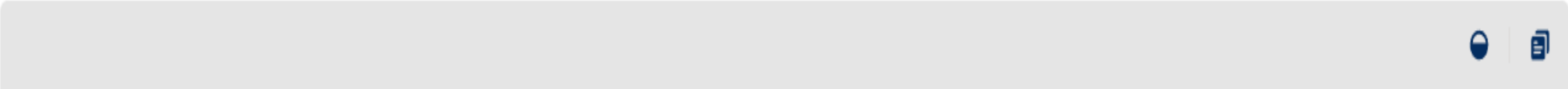
This example creates a list and adds one element to it. Then, an attempt is made to access two elements, one at index 0, which exists, and one at index 1, which causes a ListException to be thrown because no element exists at this index. This exception is caught in the catch block. The `System.debug` statement in the catch block writes the following to the debug log: The following exception has occurred: List index out of bounds: 1.

```
1  try {
2      List<Integer> li = new List<Integer>();
3      li.add(15);
4      // This list contains only one element,
5      // but we're attempting to access the second element
6      // from this zero-based list.
7      Integer i1 = li[0];
8      Integer i2 = li[1]; // Causes a ListException
9  } catch(ListException le) {
10     System.debug('The following exception has occurred: ' + le.getMessage());
11 }
```

NullPointerException

Any problem with dereferencing a `null` variable.

This example creates a String variable named `s` but we don't initialize it to a value, hence, it is null. Calling the `contains` method on our null variable causes a `NullPointerException`. The exception is caught in our catch block and this is what is written to the debug log: The following exception has occurred: Attempt to de-reference a null object.



```
1 try {
2     String s;
3     Boolean b = s.contains('abc'); // Causes a NullPointerException
4 } catch(NullPointerException npe) {
5     System.debug('The following exception has occurred: ' + npe.getMessage());
6 }
```

QueryException

Any problem with SOQL queries, such as assigning a query that returns no records or more than one record to a singleton sObject variable. The second SOQL query in this example causes a QueryException. The example assigns a Merchandise object to what is returned from the query. Note the use of `LIMIT 1` in the query. This ensures that at most one object is returned from the database so we can assign it to a single object and not a list. However, in this case, we don't have a Merchandise named XYZ, so nothing is returned, and the attempt to assign the return value to a single object results in a QueryException. The exception is caught in our catch block and this is what you'll see in the debug log: The following exception has occurred: List has no rows for assignment to SObject.

```
1  try {
2      // This statement doesn't cause an exception, even though
3      // we don't have a merchandise with name='XYZ'.
4      // The list will just be empty.
5      List<Merchandise__c> lm = [SELECT Name FROM Merchandise__c WHERE Name = 'XYZ'];
6      // lm.size() is 0
7      System.debug(lm.size());
8
9      // However, this statement causes a QueryException because
10     // we're assigning the return value to a Merchandise__c object
11     // but no Merchandise is returned.
12     Merchandise__c m = [SELECT Name FROM Merchandise__c WHERE Name = 'XYZ' LIMIT 1];
13 } catch(QueryException qe) {
14     System.debug('The following exception has occurred: ' + qe.getMessage());
15 }
```

SObjectException

Any problem with sObject records, such as attempting to change a field in an `update` statement that can only be changed during `insert`. This example results in an `SObjectException` in the try block, which is caught in the catch block. The example queries an invoice statement and selects only its Name field. It then attempts to get the `Description__c` field on the queried sObject, which isn't available because it isn't in the list of fields queried in the `SELECT` statement. This results in an `SObjectException`. This exception is caught in our catch block and this is what you'll see in the debug log: The following exception has occurred: SObject row was retrieved via SOQL without querying the requested field: Invoice_Statement__c.Description__c.

```
1 try {
2     Invoice_Statement__c inv = new Invoice_Statement__c(
3         Description__c='New Invoice');
4     insert inv;
5
6     // Query the invoice we just inserted
7     Invoice_Statement__c v = [SELECT Name FROM Invoice_Statement__c WHERE Id = :inv.Id];
8     // Causes an SObjectException because we didn't retrieve
9     // the Description__c field.
10    String s = v.Description__c;
11 } catch(SObjectException se) {
12     System.debug('The following exception has occurred: ' + se.getMessage());
13 }
```

STANDARD CLASSES

UserInfo

System

String

Integer

Test

All about UserInfo Class in Apex

UserInfo class contains methods to get the LoggedIn User or Context User information.

All methods in UserInfo class are static method, so you can access the methods using the syntax – `ClassName.methodName()` like `UserInfo.methodName()`

getUserId() Method – return the current user Id.

```
Id currentUserId = UserInfo.getUserId();
```

getProfileId() Method – returns the current user profile Id.

```
Id userProfileId = UserInfo.getProfileId();
```

getUiTheme() method – returns the preferred theme of the current user. Use `getUiThemeDisplayed()` to determine the theme actually displayed to the current user.

getUiThemeDisplayed() method – returns the theme displayed by the current user. Theme here means whether user is using Salesforce Classic or Lightning theme, using this you can identify current theme and do some business requirement.

```
String currentTheme = UserInfo.getUiThemeDisplayed();
```

getFirstName() – return the current user First Name of String type.

getLanguage() – returns the current user language of String type.

getLastName() – returns the current user Last Name of String type.

getUserName() – returns the current user login name of String type.

getUserRoleId() – returns the current user's role Id of String Type.

getUserType() – returns the current user's type of return type String.

System Class

Contains methods for system operations, such as writing debug messages and scheduling jobs.

Namespace

System

System Methods

The following are methods for `System`. All methods are static.

- **`abortJob(jobId)`**
Stops the specified job. The stopped job is still visible in the job queue in the Salesforce user interface.
- **`assert(condition, msg)`**
Asserts that the specified condition is true. If it is not, a fatal error is returned that causes code execution to halt.
- **`assertEquals(expected, actual, msg)`**
Asserts that the first two arguments are the same. If they are not, a fatal error is returned that causes code execution to halt.
- **`assertNotEquals(expected, actual, msg)`**
Asserts that the first two arguments are different. If they are the same, a fatal error is returned that causes code execution to halt.
- **`currentPageReference()`**
Returns a reference to the current page. This is used with Visualforce pages.
- **`currentTimeMillis()`**
Returns the current time in milliseconds, which is expressed as the difference between the current time and midnight, January 1, 1970 UTC.
- **`debug(msg)`**
Writes the specified message, in string format, to the execution debug log. The `DEBUG` log level is used.
- **`debug(logLevel, msg)`**
Writes the specified message, in string format, to the execution debug log with the specified log level.

What is Apex String Class Salesforce?

String Class Salesforce is basically a class variable that consists of various Apex String Methods and these particular String methods Salesforce allows users to perform multiple operations in different strings. So when these String Methods combine together they make a string class in Salesforce.

List of Apex String Methods Salesforce-

There are different types of Apex String Methods in Apex String Class that allow the users to deal with various pre-defined methods. These pre-defined methods are apex string methods Salesforce.

Some of the popular Apex String Methods are listed below-

- **contains** – The contains method will return the value True if the given string is present in the substring.

Example –

```
String myProduct1 = 'Salesforce';  
String myProduct2 = 'Intllipaas Salesforce Course';  
Boolean result = myProduct2.contains(myProduct1);  
System.debug('Output will be true as it contains the String and  
Output is:'+result);
```


- **charAt(index)** – The charAt method allows the return of the value of the character specified in the particular index. Example-

```
String str = 'Hello Intellipaat.';  
System.assertEquals(937, str.charAt(0));
```

- **equals** – The equals method allows returning true value if the given string and the string passed in the method have the same binary sequence of characters.

Note– The values must not be NULL, also, it is case sensitive.

```
String string1 = 'Intellipaat';  
String string2 = 'Intellipaat';  
Boolean result = string2.equals(string1);  
System.debug('Value of Result will be true as they are same and Result is: '+result);
```

- **equalsIgnoreCase** – This method will return true if stringtoCompare has the same sequence of characters as the given string.

Note– this method is not case-sensitive.

```
String string1 = 'intellipaat';  
String string2 = 'INTELLIPAAT';  
Boolean result =  
myString1.equalsIgnoreCase(string2);  
System.assertEquals(result, true);
```

- **remove** – This method deletes the string specified by `stringToRemove` from the specified string. This is useful when you want to delete certain characters from a string and you don't know the exact index of the character to be deleted.

Note- This method is case-sensitive. If the same string appears but the case is different, this method will not work as a string method in Salesforce

```
String s1 = 'Salesforce and force.com';  
String s2 =  
    s1.remove('force');  
System.assertEquals(  
    'Sales and .com', s2);
```

- **indexOf** – Returns the index of the first occurrence of the specified substring. If the substring does not occur, this method returns -1. Example-

```
String myString1 = 'abcde';  
String myString2 = 'cd';  
Integer result = myString1.indexOf(mystring2);  
System.assertEquals(2, result);
```


Integer Class

Contains methods for the Integer primitive data type.

Namespace

System

Usage

For more information on integers, see [Integer Data Type](#).

Integer Methods

The following are methods for `Integer`.

- **`format()`**
Returns the integer as a string using the locale of the context user.
- **`valueOf(stringToInteger)`**
Returns an Integer that contains the value of the specified String. As in Java, the String is interpreted as representing a signed decimal integer.
- **`valueOf(fieldValue)`**
Converts the specified object to an Integer. Use this method to convert a history tracking field value or an object that represents an Integer value.

Test Class

Contains methods related to Apex tests.

Namespace

System

Test Methods

The following are methods for `Test`. All methods are static.

isRunningTest()

startTest()

stopTest()

Trailhead Links

APEX	Apex Developer Guide	https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_dev_guide.htm
		https://trailhead.salesforce.com/projects/quickstart-apex