Collection : List, Set And Map

List:

Apex has a list collection type that holds an ordered collection of objects. You will use a list whenever you want to store a set of values that can be accessed with an index.

You can access an element of a list using a zero-based index, or by iterating over the list. Here's how to create a new list, and display its size:

```
List<Integer> myList = new
List<Integer>();
System.debug(myList.size());
```

Arrays in Apex are synonymous with lists. Apex provides an array-like syntax for accessing lists. Here is an alternative way to create exactly the same list:

```
Integer[] myList = new List<Integer>();
```

You can also define a list variable and initialize it at the same time as shown in the following example

```
List<String> myStrings = new List<String> {
'one', 'two' };
```

To add a new element to a list, use the add method.

```
myList.add(101);
```

Try It Out

```
List<Integer> myList = new List<Integer>();

myList.add(10);

// Retrieve the first element of the list
// using array notation
Integer i = myList[0];
// or using the get method
Integer j = myList.get(0);
System.debug('First element in the array using
myList[0] is ' + i);
```

Here is a portion of the output when you run this snippet in the Developer Console:

17:07:19:034	USER_DEBUG	[10]|DEBUG|First element in the array using myList[0] is 10
17:07:19:034	USER_DEBUG	[11]|DEBUG|First element in the array using myList.get(0) is 10

Sets

Besides Lists, Apex supports two other collection types: *Sets* and *Maps*.

A set is an unordered collection of objects that doesn't contain any duplicate values. Use a set when you don't need to keep track of the

order of the elements in the collection, and when the elements are unique and don't have to be sorted.

```
Set<String> s = new Set<String>{'a','b','c'};
// Because c is already a member, nothing will
happen.
s.add('c');
s.add('d');
if (s.contains('b')) {
    System.debug ('I contain b and have size '
+ s.size());
}
```

After running the example, you will see this line in the output:.

Maps:

Maps are collections of key-value pairs, where the keys are of primitive data types. Use a map when you want to store values that are to be referenced through a key. For example, using a map you can store a list of addresses that correspond to employee IDs.

```
Map<Integer, String> employeeAddresses = new
Map<Integer, String>();
employeeAddresses.put (1, '123 Sunny Drive, San
Francisco, CA');
employeeAddresses.put (2, '456 Dark Drive, San
Francisco, CA');
System.debug('Address for employeeID 2: ' +
employeeAddresses.get(2));
```

After running the example, you will see this line in the output:


```
Map<String, String> myStrings =
new Map<String,String>{'a'=>'apple','b'=>'bee'};

System.debug(myStrings.get('a'));
```