**SequenceEqual() method is used to determine whether two sequences are equal.**This method returns true if the sequences are equal otherwise false.    
  
   
  
**For 2 sequences to be equal**  
**1.** Both the sequences should have same number of elements and  
**2.** Same values should be present in the same order in both the sequences  
  
**Example 1 :**SequenceEqual() returns true.  
  
string[] countries1 = { "USA", "India", "UK" };

string[] countries2 = { "USA", "India", "UK" };

var result = countries1.SequenceEqual(countries2);

Console.WriteLine("Are Equal = " + result);

**Example 2 :** In this case, **SequenceEqual()**returns false, as the default comparison is case sensitive.   
  
string[] countries1 = { "USA", "INDIA", "UK" };

string[] countries2 = { "usa", "india", "uk" };

var result = countries1.SequenceEqual(countries2);

Console.WriteLine("Are Equal = " + result);

**Example 3:**If we want the comparison to be **case-insensitive**, then use the other overloaded version of SequenceEqual() method to which we can pass an alternate comparer.  
  
string[] countries1 = { "USA", "INDIA", "UK" };

string[] countries2 = { "usa", "india", "uk" };

var result = countries1.SequenceEqual(countries2,StringComparer.OrdinalIgnoreCase);

Console.WriteLine("Are Equal = " + result);

**Example 4 :** SequenceEqual() returns false. This is because, although both the sequences contain same data, the data is not present in the same order.  
  
string[] countries1 = { "USA", "INDIA", "UK" };

string[] countries2 = { "UK", "INDIA", "USA" };

var result = countries1.SequenceEqual(countries2);

Console.WriteLine("Are Equal = " + result);

**Example 5 :** To fix the problem in Example 4, use **OrderBy()**to sort data in the source sequences.  
  
string[] countries1 = { "USA", "INDIA", "UK" };

string[] countries2 = { "UK", "INDIA", "USA" };

var result = countries1.OrderBy(c => c).SequenceEqual(countries2.OrderBy(c => c));

Console.WriteLine("Are Equal = " + result);

**Example 6 :** When comparing complex types, the default comparer will only check if the object references are equal. So, in this case SequenceEqual() returns false.  
  
List<Employee> list1 = new List<Employee>()

{

    new Employee { ID = 101, Name = "Mike"},

    new Employee { ID = 102, Name = "Susy"},

};

List<Employee> list2 = new List<Employee>()

{

    new Employee { ID = 101, Name = "Mike"},

    new Employee { ID = 102, Name = "Susy"},

};

var result = list1.SequenceEqual(list2);

Console.WriteLine("Are Equal = " + result);

**To solve the problem in Example 6, there are 3 ways**  
**1.** Use the other overloaded version of SequenceEqual() method to which we can pass a custom class that implements IEqualityComparer  
**2.** Override Equals() and GetHashCode() methods in Employee class  
**3.** Project the properties into a new anonymous type, which overrides Equals() and GetHashCode() methods  
  
We discussed implementing these 3 options for Distinct() method in [Part 26](http://csharp-video-tutorials.blogspot.com/2014/08/part-26-set-operators-in-linq.html) of [LINQ Tutorial](https://www.youtube.com/playlist?list=PL6n9fhu94yhWi8K02Eqxp3Xyh_OmQ0Rp6). In the same way these options can be implemented for SequenceEqual() method. 