**In this video we will discuss**  
1. What are Extension Methods  
2. How to implement extension methods   
  
   
  
**What are Extension Methods**  
**According to MSDN**, Extension methods enable you to "add" methods to existing types without creating a new derived type, recompiling, or otherwise modifying the original type.   
**Extension methods are a special kind of static method**, but they are called as if they were instance methods on the extended type.   
  
**For client code written in C# and Visual Basic**, there is no apparent difference between calling an extension method and the methods that are actually defined in a type.  
  
**Let us understand what this definition actually means.**  
**LINQ's standard query**operators (select, where etc ) are implemented in Enumerable class as extension methods on the IEnumerable<T> interface.  
  
**Now look at the following query**

List<int> Numbers = new List<int> { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

IEnumerable<int> EvenNumbers = Numbers.Where(n => n % 2 == 0);

In spite of **Where()**method not belonging to **List<T>**class, we are still able to use it as though it belong to **List<T>**class. This is possible because **Where()**method is implemented as extension method in **IEnumerable<T>**interface and **List<T>**implements **IEnumerable<T>**interface.  
  
**How to implement extension methods**  
We want to define a method in the string class (let's call it ChangeFirstLetterCase), which will change the case of the first letter of the string. For example, if the first letter of the string is lowercase the function should change it to uppercase and viceversa.  
  
We want to be able to call this function on the string object as shown below.

string result = strName.ChangeFirstLetterCase();

Defining **ChangeFirstLetterCase**() method directly in the **string**class is not possible as we don't own the string class. It belongs to .NET framework. Another alternative is to write a wrapper class as shown below.

public class StringHelper

{

    public static string ChangeFirstLetterCase(string inputString)

    {

        if (inputString.Length > 0)

        {

            char[] charArray = inputString.ToCharArray();

            charArray[0] = char.IsUpper(charArray[0]) ?

                char.ToLower(charArray[0]) : char.ToUpper(charArray[0]);

            return new string(charArray);

        }

        return inputString;

    }

}

Wrapper class works, but the problem is, we cannot call **ChangeFirstLetterCase**() method using the following syntax.  
string result = strName.ChangeFirstLetterCase();  
  
Instead we have to call it as shown below.  
string result = StringHelper.ChangeFirstLetterCase(strName);  
  
Convert **ChangeFirstLetterCase**() method to an extension method to be able to call it using the following syntax, as though it belongs to string class.  
string result = strName.ChangeFirstLetterCase();  
  
To **convert ChangeFirstLetterCase() method to an extension method**, make the following 2 changes  
**1.** Make StringHelper static class  
**2.** The type the method extends should be passed as a first parameter with this keyword preceeding it.  
  
With these 2 changes, we should be able to call this extension method in the same way we call an instance method. Notice that the extension method shows up in the intellisense as well, but with a different visual clue.  
string result = strName.ChangeFirstLetterCase();  
  
Please note that, we should still be able to call this extension method using wrapper class style syntax. In fact, behind the scene this is how the method actually gets called. Extension methods are just a syntactic sugar.  
string result = StringHelper.ChangeFirstLetterCase(strName);  
  
So, this means we should also be able to call LINQ extension methods (select, where etc), using wrapper class style syntax. Since all LINQ extension methods are defined in Enumerable class, the syntax will be as shown below.

List<int> Numbers = new List<int> { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

IEnumerable<int> EvenNumbers = Enumerable.Where(Numbers, n => n % 2 == 0);