# About Extension Methods

**[Extension methods](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/extension-methods" \t "https://exercism.org/tracks/csharp/concepts/_blank)** allow adding methods to existing types without creating a new derived type, recompiling, or otherwise modifying the original type.

Extension methods are static methods, but they're called as if they were instance methods on the extended type. It achieves this by using this before the type, indicating the instance we put the . on is passed as the first parameter. For client code, there's no apparent difference between calling an extension method and the methods defined in a type.

namespace MyExtensions

{

public static int WordCount(this string str)

{

return str.Split().Length;

}

}

"Hello World".WordCount();// => 2

Extension methods are brought into scope at the namespace level. This means that if you are in different namespace to the one the extension method is defined in, it's namespace must be in a using directive first; For example, if we wanted to use the above example in our code, we would first need a using MyExtensions directive. If you are in the same namespace as the one the extension method is defined in, you can use the extension methods without a using directive.

A well-known example of extension methods are the **[LINQ](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/" \t "https://exercism.org/tracks/csharp/concepts/_blank)** standard query operators that add query functionality to the existing IEnumerable types. To bring these into scope we need a using System.Linq; directives.