Language Integrated Query (LINQ)

System.LINQ namespace

LINQ is a built in technology that provides a consistent query syntax across different data sources you use in code. Traditionally the developer uses strings containing syntax (such as SQL) to express their queries. With LINQ, you are using a standard query syntax in C# with IntelliSense and type checking to help guide the development.

Use LINQ to retrieve, filter, order, group, and sort data from any source. Data Sources include Databases, XML, Datasets, Entities, Collections, Arrays, and Objects that implement IEnumerable.

Features of LINQ:

- Use **LINQ** to transform data from the source to a different output. For example, results from a database stored in a collection.
- **LINQ** can be expressed in a declarative query syntax or by calling extension generic methods. Compare the where syntax with the Where<> extension method in this example:

	int[] numbers = {23, 45, 1223, 78, 1234}; //data source
Syntax:	**IEnumerable<int>** result = from number in **numbers** where number = 100 select number;
Method:	**IEnumerable<int>** result = **numbers**.Where(number => number = 100);

- Variables in the query are usually strongly typed (**IEnumerable<int>**) however you can use **var** (means variant or any type) to infer the results at compile time:

```
var result = from number in numbers where number = 100 select number;
```

- You can combine query syntax with method syntax to obtain a result. Some operations such as Count<> and Max<> are only available using methods because there is no query syntax equivalent.
- Query syntax is often more readable so use wherever possible and method syntax wherever necessary. However, there is no performance difference.
- A query executes when you process the data not when the expression is declared. For example, in a for each statement, you loop thru the result:

```
foreach(int n in result) //this is when the query executes
 { … }
```

Key Query Syntax Keywords:

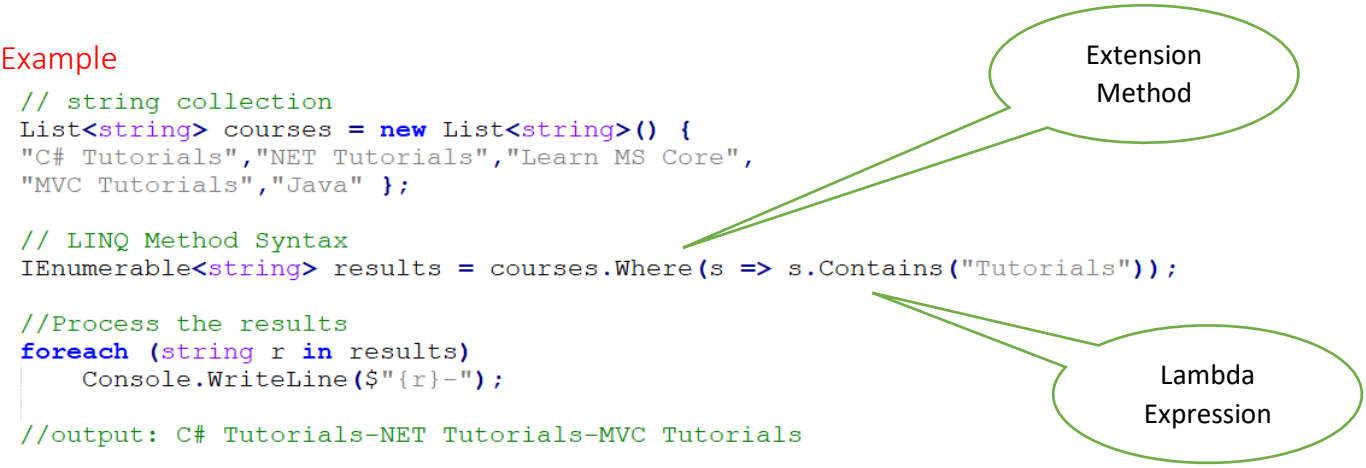
from	Specifies a data source and a range variable (similar to an iteration variable).
where	Filters source elements based on one or more Boolean expressions separated by logical AND and OR operators (&& or ||).
select	Specifies the type and shape that the elements in the returned sequence will have when the query is executed.
group	Groups query results according to a specified key value.
into	Provides an identifier that can serve as a reference to the results of a join, group or select clause.
orderby	Sorts query results in ascending or descending order based on the default comparer for the element type.
join	Joins two data sources based on an equality comparison between two specified matching criteria.
let	Introduces a range variable to store sub-expression results in a query expression.
in on equals	Contextual keywords in a join clause.
by	Contextual keyword in a group clause.
ascending	Contextual keyword in an orderby clause.
descending	Contextual keyword in an orderby clause.

The standard query syntax operators include **from, where, select, orderby, groupby, join, max, and average**. There are extension methods equivalent to the query operators along with many others such as **SelectMany, Skip, SkipWhile, Reverse, Sort**, etc!

Use IntelliSense (dot syntax) to discover the many extension methods with data sources such as arrays and collections. Different providers (namespaces) implement their own query operators and additional methods. For example, the System.Data.SQLClient namespace provides LINQ syntax and extension methods for use with Sequel Server Databases.

Lambda Expressions are a powerful and flexible tool in LINQ development. Although not required for basic LINQ queries some complex queries will require Lambda expressions. The Lambda expression can include method calls and complex logic.

Example




```
// string collection
List<string> courses = new List<string>() {
"C# Tutorials","NET Tutorials","Learn MS Core",
"MVC Tutorials","Java" };

// LINQ Method Syntax
IEnumerable<string> results = courses.Where(s => s.Contains("Tutorials"));

//Process the results
foreach (string r in results)
    Console.WriteLine($"{r}-");

//output: C# Tutorials-NET Tutorials-MVC Tutorials
```

Explanation: The letter s used in the Lambda expression above is a valid identifier used as a placeholder for the data queried. The compiler knows from the <string> template that s is a string. The => means 'goes to'. On the right is the string method Contains used to retrieve all the strings in the list that contain "Tutorials".

Example

```
//custom Student Type
class Student
{
    public int ID; public string Name; public int Age;
}
//collection of Student objects
List<Student> students = new List<Student>() {
    new Student() { ID = 1, Name = "Peter", Age = 19 },
    new Student() { ID = 1, Name = "Melissa", Age = 18 }
};

//select student object where age is less than 19 years
var results = students.Where(p => p.Age < 19).Select(p => p).ToList();

foreach (Student s in results)
    Console.WriteLine(s.Name);
//output: Melissa
```

Explanation: The p identifier is a Student object. The Select method is selecting the entire object in the results. The data source is the List<Student> and the result is a List<Student>. If you wanted names only, you write Select(p => p.Name). The var results becomes a List<string>.

Microsoft Documentation: Basic LINQ Query Operations in C#:

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/basic-linq-query-operations