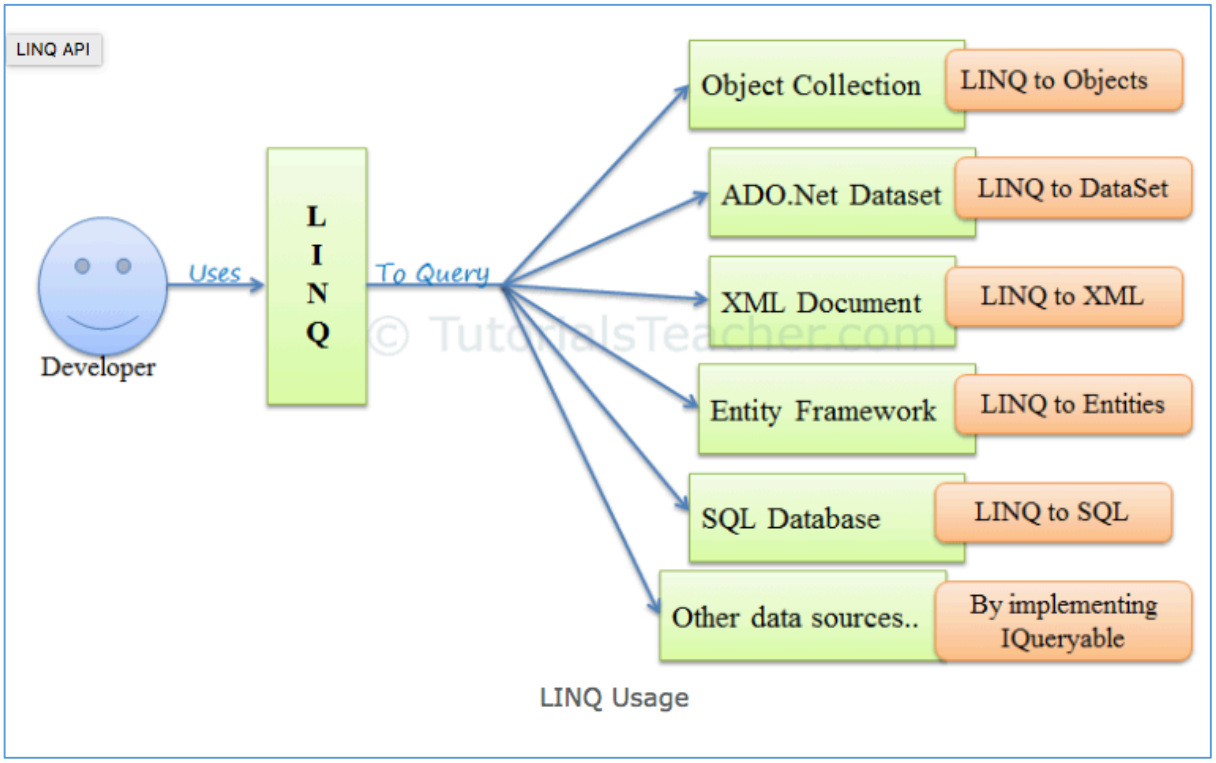
What Is LINQ?

Language Integrated Query

A query syntax used to retrieve data from different sources and formats.
Integrated into the programming language syntax for C# and VB.NET
Provides a single querying interface for different types of data sources.



LINQ Usage

LINQ queries return results as objects, so we can use the object oriented approach on the result set. This means we can use the same basic coding pattern for any data format for which a LINQ provider is available.

A Basic Example Using LINQ

```
using System;
using System.Linq;

namespace MyApp
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("My first LINQ query");

            // Using an array as a data source
            string[] names = { "Bob", "Fred", "Peter", "Archie" };

            // A simple LINQ query
            var myLinqQuery = from name in names where name.Contains('e') select name;

            // Query execution
            foreach (var name in myLinqQuery)
                Console.WriteLine(name + " ");
        }
    }
}
```

Next Steps:

- Simple LINQ example (in our .NET console application, or on https://try.dot.net
- Class discussion over what we recognise with this application that uses LINQ.

Why LINQ Was Invented

Before LINQ was invented, if we wanted to query an array of Student objects (e.g. to find all teenage students), we would need to use a for loop which is cumbersome and hard to read.

```
class Student
{
    public int StudentID { get; set; }
    public String StudentName { get; set; }
    public int Age { get; set; }
}

class Program
{
    static void Main(string[] args)
    {
        Student[] studentArray = {
            new Student() { StudentID = 1, StudentName = "John", Age = 18 },
            new Student() { StudentID = 2, StudentName = "Steve",  Age = 21 },
            new Student() { StudentID = 3, StudentName = "Bill",  Age = 25 },
            new Student() { StudentID = 4, StudentName = "Ram" , Age = 20 },
            new Student() { StudentID = 5, StudentName = "Ron" , Age = 31 },
            new Student() { StudentID = 6, StudentName = "Chris",  Age = 17 },
            new Student() { StudentID = 7, StudentName = "Rob",Age = 19  },
        };

        Student[] students = new Student[10];

        int i = 0;

        foreach (Student std in studentArray)
        {
            if (std.Age > 12 && std.Age < 20)
            {
                students[i] = std;
                i++;
            }
        }
    }
}
```

But if we had the students in a relational database table, it would be easy to do with SQL.
The C# team felt they needed to make this type of code more compact and readable.
So they introduced LINQ.

```
// LINQ Query Syntax to find out teenager students
var teenAgerStudent = from s in studentList
                      where s.Age > 12 && s.Age < 20
                      select s;
```

LINQ makes code more compact and readable, and applies to different data sources.
So even if we had these students in a database table instead of an array of Student objects, we can use the same query code.

Advantages Of LINQ

- **Familiar language:** Developers don't have to learn a new query language for each type of data source or data format.
- **Less coding:** It reduces the amount of code to be written as compared with a more traditional approach.
- **Readable code:** LINQ makes the code more readable so other developers can easily understand and maintain it.
- **Standardized way of querying multiple data sources:** The same LINQ syntax can be used to query multiple data sources.
- **Compile time safety of queries:** It provides type checking of objects at compile time.
- **IntelliSense Support:** LINQ provides IntelliSense for generic collections.
- **Shaping data:** You can retrieve data in different shapes.

Things to remember:

- Use **System.Linq** namespace to use LINQ.
- LINQ API includes two main static class Enumerable & Queryable.
- The static Enumerable class includes extension methods for classes that implements the IEnumerable<T> interface.
- IEnumerable<T> type of collections are in-memory collection like List, Dictionary, SortedList, Queue, HashSet, LinkedList.
- The static Queryable class includes extension methods for classes that implements the IQueryable<T> interface.
- Remote query provider implements e.g. Linq-to-SQL, LINQ-to-Amazon etc.

Next Steps:

- Understanding the Enumerable and Queryable classes (and interfaces).

LINQ Syntax

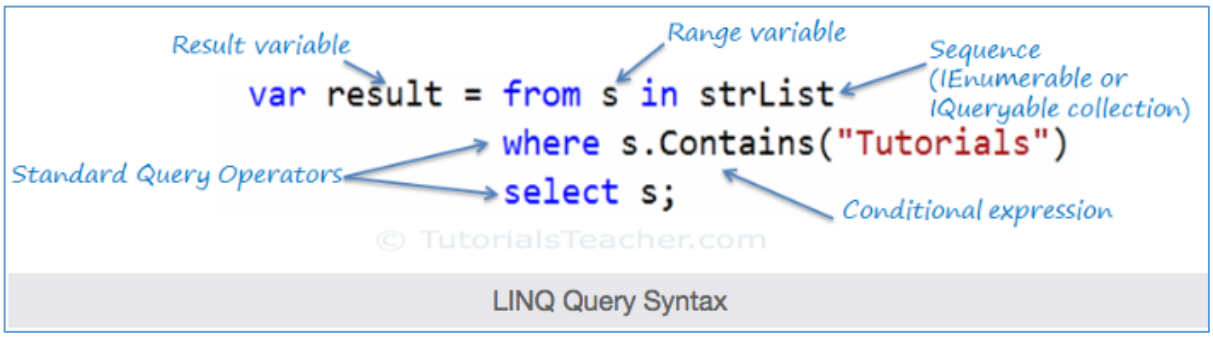
LINQ is a set of extension methods for classes that implement IEnumerable and IQueryable.

There are two ways to write a LINQ query to IEnumerable or IQueryable data sources.

1. Query syntax
2. Method syntax

LINQ Query Syntax

- Similar to SQL.
- Defined within C# or VB code.
- Starts with **from** and ends with **select** keywords.



[Taken from https://www.tutorialsteacher.com/linq/linq-query-syntax]

```
// string collection
IList<string> stringList = new List<string>() {
    "C# Tutorials",
    "VB.NET Tutorials",
    "Learn C++",
    "MVC Tutorials" ,
    "Java"
};

// LINQ Query Syntax
var result = from s in stringList
             where s.Contains("Tutorials")
             select s;
```

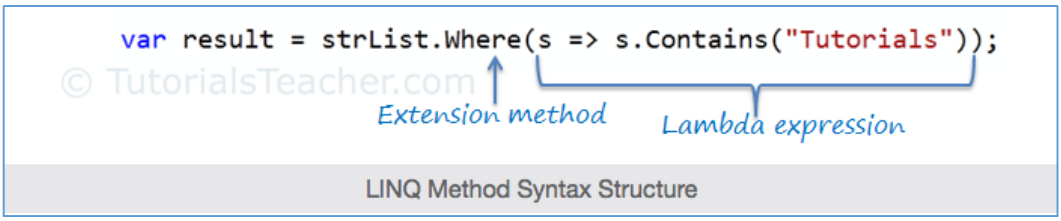
Start with **from** each object in the collection (similar to foreach loop)

Then we can use different Standard Query Operators to filter, group, join elements (are approx. 50 of these).

The **where** operator followed by a condition, usually expressed using lambda expressions.

Ends with a **select** or **group** clause (to shape the data. You can select the whole object or only some properties of it.

LINQ Method Syntax:



LINQ Method Syntax Structure

- Known as fluent syntax.
- Uses extension methods included in **Enumerable** or **Queryable** static class.

```
// string collection
IList<string> stringList = new List<string>() {
    "C# Tutorials",
    "VB.NET Tutorials",
    "Learn C++",
    "MVC Tutorials" ,
    "Java"
};

// LINQ Query Syntax
var result = stringList.Where(s => s.Contains("Tutorials"));
```

Comparison Of Query And Method Syntax:

```
// Student collection
IList<Student> studentList = new List<Student>>() {
        new Student() { StudentID = 1, StudentName = "John", Age = 13} ,
        new Student() { StudentID = 2, StudentName = "Moin",  Age = 21 } ,
        new Student() { StudentID = 3, StudentName = "Bill",  Age = 18 } ,
        new Student() { StudentID = 4, StudentName = "Ram" , Age = 20} ,
        new Student() { StudentID = 5, StudentName = "Ron" , Age = 15 }
    };

// LINQ Query Syntax to find out teenager students
var teenAgerStudent = from s in studentList
                      where s.Age > 12 && s.Age < 20
                      select s;
```

```
// Student collection
IList<Student> studentList = new List<Student>() {
        new Student() { StudentID = 1, StudentName = "John", Age = 13} ,
        new Student() { StudentID = 2, StudentName = "Moin",  Age = 21 } ,
        new Student() { StudentID = 3, StudentName = "Bill",  Age = 18 } ,
        new Student() { StudentID = 4, StudentName = "Ram" , Age = 20} ,
        new Student() { StudentID = 5, StudentName = "Ron" , Age = 15 }
    };

// LINQ Method Syntax to find out teenager students
var teenAgerStudents = studentList.Where(s => s.Age > 12 && s.Age < 20)
                                  .ToList<Student>();
```

LINQ Standard Query Operators

- Extension methods for the IEnumerable<T> and IQueryable<T> types.
- Defined in System.Linq.Enumerable and System.Linq.Queryable classes.
- There are over 50 standard query operators available in LINQ that provide different functionalities like filtering, sorting, grouping, aggregation, concatenation, etc.

Classification	Standard Query Operators
Filtering	Where, OfType
Sorting	OrderBy, OrderByDescending, ThenBy, ThenByDescending, Reverse
Grouping	GroupBy, ToLookup
Join	GroupJoin, Join
Projection	Select, SelectMany
Aggregation	Aggregate, Average, Count, LongCount, Max, Min, Sum
Quantifiers	All, Any, Contains
Elements	ElementAt, ElementAtOrDefault, First, FirstOrDefault, Last, LastOrDefault, Single, SingleOrDefault
Set	Distinct, Except, Intersect, Union
Partitioning	Skip, SkipWhile, Take, TakeWhile
Concatenation	Concat
Equality	SequenceEqual
Generation	DefaultEmpty, Empty, Range, Repeat
Conversion	AsEnumerable, AsQueryable, Cast, ToArray, ToDictionary, ToList

[Taken from https://www.tutorialsteacher.com/linq/linq-standard-query-operators]

More Information:

- https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/introduction-to-linq-queries
- https://www.dotnettricks.com/learn/linq
- https://linqsamples.com/tutorials/linq-for-beginners