Contents

[**LINQ (Language- integrated Query)** 2](#_Toc163474100)

[Advantages of LINQ 3](#_Toc163474101)

[LINQ Query Syntax 3](#_Toc163474102)

[Query Syntax 3](#_Toc163474103)

[LINQ Query Syntax: 3](#_Toc163474104)

[LINQ Method Syntax 4](#_Toc163474105)

[Example : LINQ Method syntax in C# 5](#_Toc163474106)

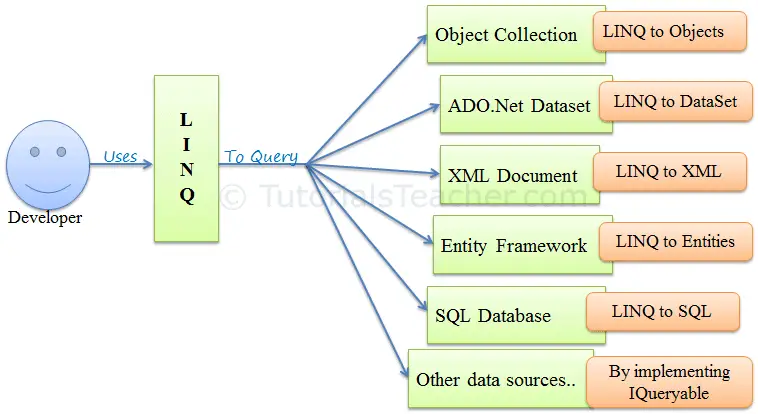
[Lambda Expression 5](#_Toc163474107)

[Standard Query Operators 6](#_Toc163474108)

[Standard Query Operators in Query Syntax 6](#_Toc163474109)

[Standard Query Operators in Method Syntax 6](#_Toc163474110)

# **LINQ (Language- integrated Query)**

Language-Integrated Query (LINQ) is a powerful set of technologies based on the integration of query capabilities directly into the C# language. LINQ Queries are the first-class language construct in C# .NET, just like classes, methods, events. The LINQ provides a consistent query experience to query objects (LINQ to Objects), relational databases (LINQ to SQL), and XML (LINQ to XML).

Example

int [] numbers = {4, 6, 8, 9, 10, 2, 1};

var n=from i in numbers where i<5 orderby i ascending select i;

foreach (int item in n)

{

Console.WriteLine(item);

}

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.

LINQ Query Syntax

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

1. Query Syntax or Query Expression Syntax
2. Method Syntax or Method Extension Syntax or Fluent

Query Syntax

Query syntax is similar to SQL (Structured Query Language) for the database. It is defined within the C# or VB code.

### LINQ Query Syntax:

**from** *<range variable>* **in** *<IEnumerable<T> or IQueryable<T> Collection>*

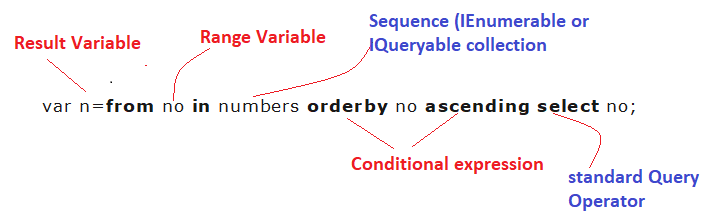
<Standard Query Operators> *<lambda expression>*

<select or groupBy operator> *<result formation>*

Example

int [] numbers = {4, 6, 8, 9, 10, 2, 1};

var n=**from** no **in** numbers **orderby** no **ascending select** no;



Query syntax starts with a ***From*** clause followed by a ***Range*** variable. The ***From*** clause is structured like "**From** rangeV*ariableName* **in** *IEnumerablecollection*". In English, this means, from each object in the collection. It is similar to a foreach loop: foreach(int no in numbers ).

After the From clause, you can use different Standard Query Operators to filter, group, join elements of the collection. **There are around 50 Standard Query Operators available in LINQ.** ([for more details click here](https://www.tutorialspoint.com/linq/linq_query_operators.htm))In the above figure, we have used "where" operator (aka clause) followed by a condition. This condition is generally expressed using lambda expression.

### LINQ Method Syntax

Method syntax (also known as fluent syntax) uses extension methods included in the Enumerable or Queryable static class, similar to how you would call the extension method of any class.

The compiler converts query syntax into method syntax at compile time.

Example:

// string collection

IList<string> cityList = new List<string>() {

"PUNE",

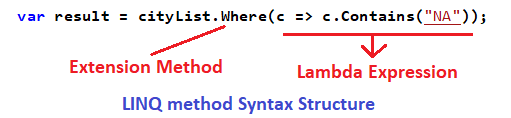
"MUMBAI",

"NASHIK",

"NAGPUR" };

// LINQ Method Syntax

var result = cityList.Where(c => c.Contains("NA"));



Example : LINQ Method syntax in C#

class student

{

public int roll { get; set; }

public string name { get; set; }

public string city{ get; set; }

}

student [] studs = {

new student () {roll=1, name="rajesh",city="nashik"},

new student () {roll=2,name="ashok",city="pune"},

new student () {roll=3,name="dinesh",city="nagar"},

new student () {roll=4,name="vinod",city="nashik"},

new student () {roll=5,name="darshan",city="solapur"},

};

// find students from nashik

var st1 = studs. Where(x => x.city == "nashik");

foreach (var item in st1)

{

Console.WriteLine(item.roll + " " + item.name + " " + item.city);

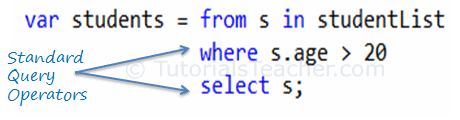
}

Lambda Expression

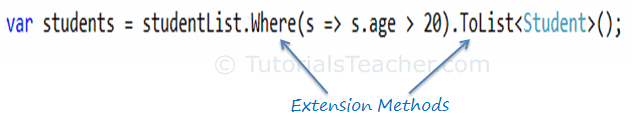
Standard Query Operators

Standard Query Operators in LINQ are actually extension methods for the IEnumerable<T> and IQueryable<T> types. They are defined in the 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.

Standard Query Operators in Query Syntax



Standard Query Operators in Method Syntax



| **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 |

### Where

The Where operator (Linq extension method) filters the collection based on a given criteria expression and returns a new collection. The criteria can be specified as lambda expression or Func delegate type.

Example

int [] numbers = {4, 6, 8, 9, 10, 2, 1};

var n=**from** no **in** numbers Where no>5 **select** no;

### Where extension method in Method Syntax

Unlike the query syntax, you need to pass whole lambda expression as a predicate function instead of just body expression in LINQ method syntax.

Example

int [] numbers = {4, 6, 8, 9, 10, 2, 1};

var n=numbers.Where(x=>x>5);