LINQ FOR BEGINNERS

.NET @ .NET WEB ACADEMY

What is LINQ?

- LINQ means Language Integrated Query.
- It helps find, sort, and filter data.
- No long loops - just short, simple commands.
- Works with lists, databases, and more!

Lambda Expressions

- A shortcut to write tiny functions.
- Helps LINQ know what to look for.
- Example: `g => g.Title` means ‘Get the title!’
- `g` is just a temporary name for each item.

LINQ Example

Original List

LINQ Method

```
var adventureGames = games.Where(g => g.Genre == "Adventure");
```

Lambda Expression

The Dataset: Video Games

```
internal class Game
{
    14 references
    public required string Title { get; set; }
    11 references
    public string? Genre { get; set; }
    8 references
    public int ReleaseYear { get; set; }
    11 references
    public double Rating { get; set; }
    10 references
    public int Price { get; set; }
}
```

In all examples, we're using this *List* of *Game* objects.

```
var games = new List<Game>
{
    new Game { Title = "The Legend of Zelda", Genre = "Adventure", ReleaseYear = 1986, Rating = 9.5, Price = 60 },
    new Game { Title = "Super Mario Bros.", Genre = "Platformer", ReleaseYear = 1985, Rating = 9.2, Price = 50 },
    new Game { Title = "Elden Ring", Genre = "Adventure", ReleaseYear = 2022, Rating = 9.8, Price = 50 },
    new Game { Title = "Stardew Valley", Genre = "Simulation", ReleaseYear = 2016, Rating = 9.0, Price = 15 },
    new Game { Title = "Tetris", Genre = "Puzzle", ReleaseYear = 1984, Rating = 8.9, Price = 10 }
};
```

Retrieving All Games

```
foreach (var game in games)
    Console.WriteLine(game.Title);
```

This is the traditional way to display all game titles. Use a *foreach* loop to iterate through a collection and display information.

Retrieving All Games

```
var allGames = games.Select(g => g.Title);
foreach (var title in allGames)
    Console.WriteLine(title);
```

Rewrite the loop with the *Select* function to make the code more concise and expressive. LINQ allows querying data directly using a fluent syntax.

Filtering Games by Genre

```
var rpgGames = games.Where(g => g.Genre == "RPG");
foreach (var game in rpgGames)
    Console.WriteLine(game.Title);
```

Use the *Where* method to filter games based on a specific condition, such as matching a genre. This method returns a subset of elements that satisfy the condition.

Checking Conditions

```
var modernGamesExist = games.Any(g => g.ReleaseYear > 2000);
Console.WriteLine($"Are there modern games? {modernGamesExist}");
```

The *Any* method checks if any element in the collection matches the condition, returning a boolean value. This is useful for quick validations.

Sorting Games

```
var sortedByYear = games.OrderBy(g => g.ReleaseYear);
foreach (var game in sortedByYear)
    Console.WriteLine($"{game.Title} ({game.ReleaseYear})");
```

Sort games by a property like release year using *OrderBy*. Sorting helps organize data, making it easier to analyze.

Sorting Games

```
var topRatedGames = games.OrderByDescending(g => g.Rating);
foreach (var game in topRatedGames)
    Console.WriteLine($"{game.Title} - Rating: {game.Rating}");
```

To reverse the order, use *OrderByDescending*, such as sorting games by their rating from highest to lowest.

Aggregating Data

```
var averagePrice = games.Average(g => g.Price);
Console.WriteLine($"Average Game Price: ${averagePrice}");
```

LINQ aggregation methods like *Average* and *Max* perform calculations over a collection, such as finding the average price or the highest-rated game.

Aggregating Data

```
var highestRating = games.Max(g => g.Rating);
var bestGame = games.First(g => g.Rating == highestRating);
Console.WriteLine(
    $"Highest Rated Game: {bestGame.Title} ({bestGame.Rating})");
```

Find the game with the highest rating by combining *Max* and *First*.

Grouping by Genre

```
var gamesByGenre = games.GroupBy(g => g.Genre);
foreach (var group in gamesByGenre)
{
    Console.WriteLine($"Genre: {group.Key}");
    foreach (var game in group)
        Console.WriteLine($"  {game.Title}");
}
```

Organize games by genre using *GroupBy*. This method creates a collection of groups, each containing elements that share a key.

Chaining Queries

```
var budgetAdventureGames = games
    .Where(g => g.Genre == "Adventure" && g.Price <= 60)
    .OrderBy(g => g.Rating)
    .Select(g => $"{g.Title} - ${g.Price}");
foreach (var game in budgetAdventureGames)
    Console.WriteLine(game);
```

Combine multiple LINQ methods to create powerful and concise queries. For example, filter, sort, and project data into a desired format in one go.

Pagination (Skip and Take)

```
var paginatedGames = games.Skip(2).Take(2);
foreach (var game in paginatedGames)
    Console.WriteLine(game.Title);
```

Use *Skip* and *Take* to implement pagination. This is especially useful for displaying large datasets in smaller chunks.

Query vs. Method Syntax

```
var adventureGames = games.Where(g => g.Genre == "Adventure");
```

```
var adventureGamesQuery = from g in games
                          where g.Genre == "Adventure"
                          select g;
foreach (var game in adventureGamesQuery)
    Console.WriteLine(game.Title);
```

LINQ queries can be written in two styles: method syntax (fluent API) and query syntax (SQL-like). Both achieve the same result but offer different readability and familiarity.

Practical Scenarios

Find the cheapest game

```
var cheapestGame = games.OrderBy(g => g.Price).First();
Console.WriteLine(
    $"Cheapest Game: {cheapestGame.Title} - ${cheapestGame.Price}");
```

Use *OrderBy* and *First* to quickly locate the game with the lowest price.

Practical Scenarios

List all genres (distinct)

```
var genres = games.Select(g => g.Genre).Distinct();
foreach (var genre in genres)
    Console.WriteLine(genre);
```

Extract unique genres from the collection using *Select* and *Distinct*.

Get the Source Code!

You can download the full source code for this LINQ project and follow along at your own pace.

📥 **Download Now:** https://dotnetwebacademy.com/linq

Start practicing and take your LINQ skills to the next level!

.NET @ .NET WEB ACADEMY

Keep Learning and Coding!

Great job finishing this LINQ guide! Now you can use LINQ to write better and faster code.

If you liked this handbook, share it with others who might find it helpful.

For more tips and tutorials, check out my *YouTube channel* or visit the *.NET Web Academy*.

Happy coding! 😊

.NET @ .NET WEB ACADEMY