### ****Complete Table of LINQ Methods in C#****

Here's a comprehensive list of LINQ methods that can be used with common collections like List, Array, Dictionary, and other IEnumerable<T> or IQueryable<T> collections in C#. Each entry includes the method, its description, syntax, and a simple example.

| **Method** | **Description** | **Syntax** | **Example** |
| --- | --- | --- | --- |
| **Where** | Filters elements based on a condition. | **collection.Where(predicate)** | **var evenNumbers = numbers.Where(x => x % 2 == 0);** |
| **Select** | Projects each element into a new form. | **collection.Select(selector)** | **var squares = numbers.Select(x => x \* x);** |
| **SelectMany** | Projects each element into a collection and flattens the resulting collections into one collection. | **collection.SelectMany(selector)** | **var characters = words.SelectMany(w => w.ToCharArray());** |
| **OrderBy** | Sorts elements in ascending order based on a key. | **collection.OrderBy(keySelector)** | **var sortedNames = names.OrderBy(n => n.Length);** |
| **OrderByDescending** | Sorts elements in descending order based on a key. | **collection.OrderByDescending(keySelector)** | **var sortedNames = names.OrderByDescending(n => n.Length);** |
| **ThenBy** | Performs a secondary sorting in ascending order. | **collection.OrderBy(keySelector).ThenBy(keySelector)** | **var sortedNames = names.OrderBy(n => n.Length).ThenBy(n => n);** |
| **ThenByDescending** | Performs a secondary sorting in descending order. | **collection.OrderBy(keySelector).ThenByDescending(keySelector)** | **var sortedNames = names.OrderBy(n => n.Length).ThenByDescending(n => n);** |
| **GroupBy** | Groups elements by a specified key. | **collection.GroupBy(keySelector)** | **var groups = numbers.GroupBy(x => x % 2);** |
| **Distinct** | Removes duplicate elements from a collection. | **collection.Distinct()** | **var uniqueNumbers = numbers.Distinct();** |
| **Union** | Returns the union of two collections, removing duplicates. | **collection1.Union(collection2)** | **var unionNumbers = list1.Union(list2);** |
| **Intersect** | Returns the intersection of two collections (common elements). | **collection1.Intersect(collection2)** | **var commonNumbers = list1.Intersect(list2);** |
| **Except** | Returns the difference between two collections (elements in the first collection not in the second). | **collection1.Except(collection2)** | **var difference = list1.Except(list2);** |
| **Take** | Takes the first n elements from a collection. | **collection.Take(n)** | **var firstThree = numbers.Take(3);** |
| **TakeWhile** | Takes elements from a collection while a condition is true. | **collection.TakeWhile(predicate)** | **var smallNumbers = numbers.TakeWhile(x => x < 5);** |
| **Skip** | Skips the first n elements in a collection. | **collection.Skip(n)** | **var afterFirstThree = numbers.Skip(3);** |
| **SkipWhile** | Skips elements while a condition is true, then returns the rest. | **collection.SkipWhile(predicate)** | **var afterSmallNumbers = numbers.SkipWhile(x => x < 5);** |
| **Join** | Joins two collections based on matching keys. | **collection1.Join(collection2, key1, key2, resultSelector)** | **var joined = customers.Join(orders, c => c.Id, o => o.CustomerId, (c, o) => new { c.Name, o.OrderId });** |
| **GroupJoin** | Groups and joins two collections based on a key. | **collection1.GroupJoin(collection2, key1, key2, resultSelector)** | **var groupJoin = customers.GroupJoin(orders, c => c.Id, o => o.CustomerId, (c, orders) => new { c.Name, orders });** |
| **Aggregate** | Applies an accumulator function over a sequence. | **collection.Aggregate(seed, func)** | **var sum = numbers.Aggregate(0, (total, next) => total + next);** |
| **All** | Checks if all elements in a collection satisfy a condition. | **collection.All(predicate)** | **bool allEven = numbers.All(x => x % 2 == 0);** |
| **Any** | Checks if any element in a collection satisfies a condition. | **collection.Any(predicate)** | **bool hasEven = numbers.Any(x => x % 2 == 0);** |
| **Contains** | Checks if a collection contains a specific element. | **collection.Contains(element)** | **bool hasThree = numbers.Contains(3);** |
| **Count** | Counts the number of elements in a collection. | **collection.Count()** | **int count = numbers.Count();** |
| **Sum** | Computes the sum of numeric elements in a collection. | **collection.Sum()** | **int total = numbers.Sum();** |
| **Average** | Computes the average of numeric elements in a collection. | **collection.Average()** | **double average = numbers.Average();** |
| **Max** | Returns the maximum value in a collection. | **collection.Max()** | **int max = numbers.Max();** |
| **Min** | Returns the minimum value in a collection. | **collection.Min()** | **int min = numbers.Min();** |
| **First** | Returns the first element of a collection. | **collection.First()** | **int first = numbers.First();** |
| **FirstOrDefault** | Returns the first element, or a default value if no elements exist. | **collection.FirstOrDefault()** | **int first = numbers.FirstOrDefault();** |
| **Last** | Returns the last element of a collection. | **collection.Last()** | **int last = numbers.Last();** |
| **LastOrDefault** | Returns the last element, or a default value if no elements exist. | **collection.LastOrDefault()** | **int last = numbers.LastOrDefault();** |
| **Single** | Returns the only element in a collection (throws an exception if more than one element exists). | **collection.Single()** | **int single = numbers.Single(x => x == 5);** |
| **SingleOrDefault** | Returns the only element, or a default value if no elements exist (throws if more than one exists). | **collection.SingleOrDefault()** | **int single = numbers.SingleOrDefault(x => x == 5);** |
| **ToList** | Converts a collection to a List<T>. | **collection.ToList()** | **List<int> list = numbers.ToList();** |
| **ToArray** | Converts a collection to an array. | **collection.ToArray()** | **int[] array = numbers.ToArray();** |
| **ToDictionary** | Converts a collection to a Dictionary<TKey, TValue>. | **collection.ToDictionary(keySelector, valueSelector)** | **var dict = students.ToDictionary(s => s.Id, s => s.Name);** |