Sum with Selector

This example sums lengths of all strings in the list.

```
var stringList = new List<string> { "88888888", "22", "666666", "333" };

// these two lines do the same
int lengthSum = stringList.Select(x => x.Length).Sum();  // lengthSum: 19
int lengthSum = stringList.Sum(x => x.Length);           // lengthSum: 19
```

Sum with Query Syntax

LINQ **query expression** to get sum of numeric values in the collection.

```
var list = new List<int> { 8, 2, 6, 3 };
int sum = (from x in list select x).Sum();  // sum: 19
```

LINQ **query expression** to get sum of numbers which **match specified predicate**.

```
var list = new List<int> { 8, 2, 6, 3 };
int sum = (from x in list where x > 4 select x).Sum();  // sum: 14
```

LINQ **query expression** to get sum of string lengths using **selector**.

```
var list = new List<string> { "88888888", "22", "666666", "333" };
int lengthSum = (from x in list select x.Length).Sum();  // lengthSum: 19
```

Sum with Group By

This example shows how to calculate sum for each group. Lets have players. Each player belongs to a team and have a score. Team total score is sum of score of all players in the team.

```
var players = new List<Player> {
    new Player { Name = "Alex", Team = "A", Score = 10 },
    new Player { Name = "Anna", Team = "A", Score = 20 },
    new Player { Name = "Luke", Team = "L", Score = 60 },
    new Player { Name = "Lucy", Team = "L", Score = 40 },
};

var teamTotalScores =
    from player in players
    group player by player.Team into playerGroup
    select new
    {
        Team = playerGroup.Key,
        TotalScore = playerGroup.Sum(x => x.Score),
    };

// teamTotalScores is collection of anonymous objects:
// { Team = "A", TotalScore = 30 }
// { Team = "L", TotalScore = 100 }
```

Max with Selector

This example gets length of the longest string.

```
var stringList = new List<string> { "1", "88888888", "333", "22" };

// these two lines do the same
int maxLength = stringList.Select(x => x.Length).Max();  // maxLength: 8
int maxLength = stringList.Max(x => x.Length);           // maxLength: 8
```

Max for Types Implementing IComparable

LINQ method **Max** can be used also on **collection of custom type** (not numeric type like int or decimal). The custom type have to implement **IComparable** interface.

Lets have custom type **Money** which implements **IComparable**.

```
public struct Money : IComparable<Money>
{
    public Money(decimal value) : this() { Value = value; }
    public decimal Value { get; private set; }
    public int CompareTo(Money other) { return Value.CompareTo(other.Value); }
}
```

And this is usage of **Max** method **on the Money collection**.

```
var amounts = new List<Money> { new Money(30), new Money(10) };
Money maxAmount = amounts.Max();  // maxAmount: Money with value 30
```

Max with Query Syntax

LINQ **query expression** to get maximal item in the collection.

```
var list = new List<int> { 1, 8, 3, 2 };
int maxNumber = (from x in list select x).Max();  // maxNumber: 8
```

LINQ **query expression** to get maximal item which **matches specified predicate**.

```
var list = new List<int> { 1, 8, 3, 2 };
int maxNumber = (from x in list where x < 5 select x).Max();  // maxNumber: 3
```

LINQ **query expression** to get maximal string length using **selector**.

```
var list = new List<string> { "1", "88888888", "333", "22" };
int maxLength = (from x in list select x.Length).Max();  // maxLength: 8
```

Max with Group By

This example shows how to get maximal value per group. Lets have players. Each player belongs to a team and have a score. Team best score is maximal score of players in a team.

```
var players = new List<Player> {
    new Player { Name = "Alex", Team = "A", Score = 10 },
    new Player { Name = "Anna", Team = "A", Score = 20 },
    new Player { Name = "Luke", Team = "L", Score = 60 },
    new Player { Name = "Lucy", Team = "L", Score = 40 },
};

var teamBestScores =
    from player in players
    group player by player.Team into playerGroup
    select new
    {
        Team = playerGroup.Key,
        BestScore = playerGroup.Max(x => x.Score),
    };

// teamBestScores is collection of anonymous objects:
// { Team = "A", BestScore = 20 }
// { Team = "L", BestScore = 60 }
```

Min with Selector

This example gets length of the longest string.

```
var stringList = new List<string> { "88888888", "22", "666666", "333" };

// these two lines do the same
int minLength = stringList.Select(x => x.Length).Min();  // minLength: 2
int minLength = stringList.Min(x => x.Length);           // minLength: 2
```

Min for Types Implementing IComparable

LINQ method **Min** can be used also on **collection of custom type** (not numeric type like int or decimal). The custom type have to implement **IComparable** interface.

Lets have custom type **Money** which implements **IComparable**.

```
public struct Money : IComparable<Money>
{
    public Money(decimal value) : this() { Value = value; }
    public decimal Value { get; private set; }
    public int CompareTo(Money other) { return Value.CompareTo(other.Value); }
}
```

And this is usage of **Min** method **on the Money collection**.

```
var amounts = new List<Money> { new Money(30), new Money(10) };
Money minAmount = amounts.Min();  // minAmount: Money with value 10
```

Min with Query Syntax

LINQ **query expression** to get minimal item in the collection.

```
var list = new List<int> { 8, 2, 6, 3 };
int minNumber = (from x in list select x).Min();  // minNumber: 2
```

LINQ **query expression** to get minimal item which **matches specified predicate**.

```
var list = new List<int> { 8, 2, 6, 3 };
int minNumber = (from x in list where x > 4 select x).Min();  // minNumber: 6
```

LINQ **query expression** to get minimal string length using **selector**.

```
var list = new List<string> { "88888888", "22", "666666", "333" };
int minLength = (from x in list select x.Length).Min();  // minLength: 2
```

Min with Group By

This example shows how to get minimal value per group. Lets have players. Each player belongs to a team and have a score. Team best score is minimal score of players in a team.

```
var players = new List<Player> {
    new Player { Name = "Alex", Team = "A", Score = 10 },
    new Player { Name = "Anna", Team = "A", Score = 20 },
    new Player { Name = "Luke", Team = "L", Score = 60 },
    new Player { Name = "Lucy", Team = "L", Score = 40 },
};

var teamBestScores =
    from player in players
    group player by player.Team into playerGroup
    select new
    {
        Team = playerGroup.Key,
        BestScore = playerGroup.Min(x => x.Score),
    };

// teamBestScores is collection of anonymous objects:
// { Team = "A", BestScore = 10 }
// { Team = "L", BestScore = 40 }
```

Count Examples

Counts **number of items** in the **IEnumerable** collection.

```
IEnumerable<string> items = new List<string> { "A", "B", "C" };
int count = items.Count();  // count: 3
```

Returns **0** if there is **no item** in the collection.

```
IEnumerable<string> items = new List<string> { };
int count = items.Count();  // count: 0
```

Count with Predicate

Counts **number of items** which **match specified predicate**.

```
IEnumerable<int> items = new List<int> { 8, 3, 2 };

// these two lines do the same
int count = items.Where(x => x < 5).Count();  // count: 2
int count = items.Count(x => x < 5);          // count: 2
```

Count with Query Syntax

LINQ **query expression** to count number of **all items** in the collection.

```
IEnumerable<int> items = new List<int> { 8, 3, 2 };
int count = (from x in items select x).Count();  // count: 3
```

LINQ **query expression** to count number of items which **match specified predicate**.

```
IEnumerable<int> items = new List<int> { 8, 3, 2 };
int count = (from x in items where x < 5 select x).Count();  // count: 2
```

Count with Group By

This example shows how to count number of items per group. Lets count players in each team.

```
var players = new List<Player> {
    new Player { Name = "Alex", Team = "A" },
    new Player { Name = "Anna", Team = "A" },
    new Player { Name = "Luke", Team = "L" },
    new Player { Name = "Lucy", Team = "L" },
    new Player { Name = "Mike", Team = "M" },
};

var playersPerTeam =
    from player in players
    group player by player.Team into playerGroup
    select new
    {
        Team = playerGroup.Key,
        Count = playerGroup.Count(),
    };

// playersPerTeam is collection of anonymous objects:
// { Team = "A", Count = 2 }
// { Team = "L", Count = 2 }
// { Team = "M", Count = 1 }
```

Average with Selector

You can calculate average value on **collection of any type** (IEnumerable<T>), but you have to **tranform the type T into any numeric type** using selector. It can be done by **Select** method or using **selector** as a parameter of the **Average** method.

This example counts the average length of the strings.

```
var stringList = new List<string> { "1", "88888888", "333", "22" };

// these two lines do the same
double result = stringList.Select(x => x.Length).Average();  // result: 3.5
double result = stringList.Average(x => x.Length);           // result: 3.5
```

Average with Query Syntax

LINQ **query expression** to calculate average value of **all items** in the collection.

```
var list = new List<int> { 1, 8, 3, 2 };
double result = (from x in list select x).Average();  // result: 3.5
```

LINQ **query expression** to calculate average value of items which **match specified predicate**.

```
var list = new List<int> { 1, 8, 3, 2 };
double result = (from x in list where x < 5 select x).Average();  // result: 2.0
```

LINQ **query expression** to calculate average string length using **selector**.

```
var list = new List<string> { "1", "88888888", "333", "22" };
double result = (from x in list select x.Length).Average();  // result: 3.5
```

Average with Group By

This example shows how to calculate average value for each group. Lets have players. Each player belongs to a team and have a score. The result is average score per player in a team.

```
var players = new List<Player> {
    new Player { Name = "Alex", Team = "A", Score = 10 },
    new Player { Name = "Anna", Team = "A", Score = 20 },
    new Player { Name = "Luke", Team = "L", Score = 60 },
    new Player { Name = "Lucy", Team = "L", Score = 40 },
};

var teamAverageScores =
    from player in players
    group player by player.Team into playerGroup
    select new
    {
        Team = playerGroup.Key,
        AverageScore = playerGroup.Average(x => x.Score),
    };

// teamAverageScores is collection of anonymous objects:
// { Team = "A", AverageScore = 15.0 }
// { Team = "L", AverageScore = 50.0 }
```