**Lambda expressions**

A lambda expression is a block of code (an expression or a statement block) that is treated as an object. It can be passed as an argument to methods, and it can also be returned by method calls.

A lambda expression uses =>, the [lambda declaration operator](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/lambda-operator), to separate the lambda's parameter list from its executable code. To create a lambda expression, you specify input parameters (if any) on the left side of the lambda operator, and you put the expression or statement block on the other side. For example, the single-line lambda expression x => x \* x specifies a parameter that’s named x and returns the value of x squared. You can assign this expression to a delegate type, as the following example shows:

Func<int, int> square = x => x \* x;

Console.WriteLine(square(5));

// Output:

// 25

You also can assign a lambda expression to an expression tree type:

System.Linq.Expressions.Expression<Func<int, int>> e = x => x \* x;

Console.WriteLine(e);

// Output:

// x => (x \* x)

you can pass it directly as a method argument:

int[] numbers = { 2, 3, 4, 5 };

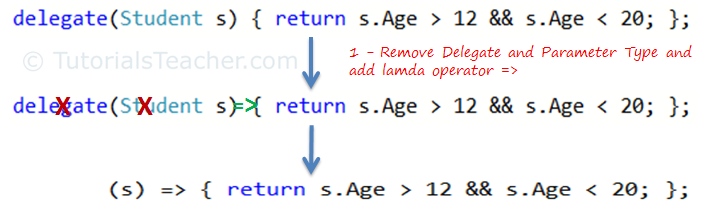
var squaredNumbers = numbers.Select(x => x \* x);

Console.WriteLine(string.Join(" ", squaredNumbers));

// Output:

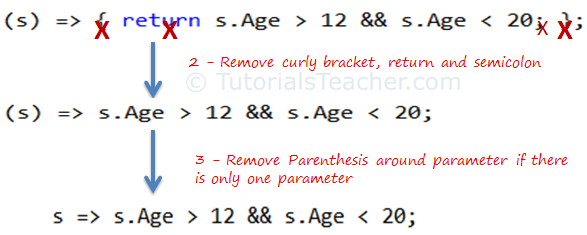
// 4 9 16 25

The Lambda expression evolves from anonymous method by first removing the delegate keyword and parameter type and adding a lambda operator =>.

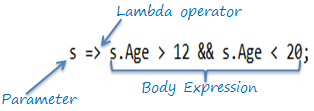
[](https://www.tutorialsteacher.com/Content/images/linq/lambda-expression-1.png)Lambda Expression from Anonymous Method

The above lambda expression is absolutely valid, but we don't need the curly braces, return and semicolon if we have only one statement that returns a value. So we can eliminate it.

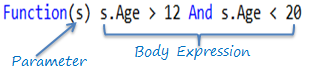
Also, we can remove parenthesis (), if we have only one parameter.

[](https://www.tutorialsteacher.com/Content/images/linq/lambda-expression-2.png)Lambda Expression from Anonymous Method

Thus, we got the lambda expression: s => s.Age > 12 && s.Age < 20 where **s** is a parameter, **=>**is the lambda operator and **s.Age > 12 && s.Age < 20** is the body expression:

[](https://www.tutorialsteacher.com/Content/images/linq/lambda-expression-structure.png)Lambda Expression Structure in C#

Same way we got lambda expression in VB.Net can be written as below:

[](https://www.tutorialsteacher.com/Content/images/linq/lambda-expression-vb.png)

public static void Main()

{

List<int> list = new List<int>() { 1, 2, 3, 4, 5, 6 };

List<int> evenNumbers = list.FindAll(x => (x % 2) == 0);

foreach (var num in evenNumbers)

{

Console.Write("{0} ", num);

}

Console.WriteLine();

Console.Read();

}

class Dog

{

public string Name { get; set; }

public int Age { get; set; }

}

class demo

{

static void Main()

{

List<Dog> dogs = new List<Dog>() {

new Dog { Name = "Rex", Age = 4 },

new Dog { Name = "Tom", Age = 0 },

new Dog { Name = "Tiger", Age = 3 }

};

var names = dogs.Select(x => x.Name);

foreach (var name in names)

{

Console.WriteLine(name);

}

Console.Read();

}

}