**Lambda.** From a branch a leaf grows. From a trunk a branch grows. A single function could make each leaf, each branch—what makes one makes many. An earthworm crawls among the tree's roots.  
**A lambda expression** describes a pattern. From mathematics, the lambda calculus describes the world in patterns. In C# a lambda is a function that uses clear and short syntax.  
**An example.** Perhaps the most common place to use lambdas is with List. Here we use FindIndex, which receives a Predicate method. We specify this as a lambda expression.

**Left, right:**To the left, we have arguments. The "x" is just a name—we can use any valid name. The result is on the right.

**Often:**We pass lambda expressions as arguments, for sorting or for searching. We use them in queries.

**Based on:** .NET 4.6

**C# program that uses lambda, List**

using System;

using System.Collections.Generic;

class Program

{

static void Main()

{

List<int> elements = new List<int>() { 10, 20, 31, 40 };

// ... Find index of first odd element.

int oddIndex = elements.FindIndex(x => x % 2 != 0);

Console.WriteLine(oddIndex);

}

}

**Output**

2

**Lambda details**

x The argument name.

=> Separates argument list from result expression.

x % 2 !=0 Returns true if x is not even.

**Detailed examples.** We take a closer look at lambdas and anonymous functions. The => operator separates the parameters to a method from its statements in the method's body.

**Tip:**Lambda expressions use the token => in an expression context. In this context, the token is not a comparison operator.

**Goes to:**The => operator can be read as "goes to." It is always used when declaring a lambda expression.

**Invoke:**With Invoke, a method on Func and Action, we execute the methods in the lambdas.

**C# program that uses lambda expressions**

using System;

class Program

{

static void Main()

{

//

// Use implicitly-typed lambda expression.

// ... Assign it to a Func instance.

//

Func<int, int> func1 = x => x + 1;

//

// Use lambda expression with statement body.

//

Func<int, int> func2 = x => { return x + 1; };

//

// Use formal parameters with expression body.

//

Func<int, int> func3 = (int x) => x + 1;

//

// Use parameters with a statement body.

//

Func<int, int> func4 = (int x) => { return x + 1; };

//

// Use multiple parameters.

//

Func<int, int, int> func5 = (x, y) => x \* y;

//

// Use no parameters in a lambda expression.

//

Action func6 = () => Console.WriteLine();

//

// Use delegate method expression.

//

Func<int, int> func7 = delegate(int x) { return x + 1; };

//

// Use delegate expression with no parameter list.

//

Func<int> func8 = delegate { return 1 + 1; };

//

// Invoke each of the lambda expressions and delegates we created.

// ... The methods above are executed.

//

Console.WriteLine(func1.Invoke(1));

Console.WriteLine(func2.Invoke(1));

Console.WriteLine(func3.Invoke(1));

Console.WriteLine(func4.Invoke(1));

Console.WriteLine(func5.Invoke(2, 2));

func6.Invoke();

Console.WriteLine(func7.Invoke(1));

Console.WriteLine(func8.Invoke());

}

}

**Output**

2

2

2

2

4

2

2

**A syntax review.** Above we see many usages of lambda expressions. Sorry for the long example. The => operator separates arguments from methods. It does not compare numbers.

**Left side:**This is the parameters. It can be empty. Sometimes it can be implicit (derived from the right).

**Right side:**This is a statement list inside curly brackets with a return statement, or an expression.

**Func1 through func8.** Above, func1 through func8 denote anonymous function instances. The C# compiler often turns different syntax forms into the same code.  
**Func.** The key part of Func is that it returns a value. It can have zero, or many, arguments. But its invariant is a return value, indicated by the TResult parametric type.[**Func**](https://www.dotnetperls.com/func)

**Func examples**

Func<TResult> Has one result value, no parameter.

Func<T, TResult> Has one result value, one parameter.

Func<T1, T2, TResult> Has one result value, two parameters.

Func<T1, T2, T3, TResult> ....

**Action.** This class indicates a function that receives no parameter and returns no value. It matches a void method with no arguments. This guy is a solitary character.[**Action**](https://www.dotnetperls.com/action)  
**Delegate.** The delegate keyword denotes an anonymous function. After this keyword, we use a formal parameter list. We can omit the list if there are no parameters.[**Delegate**](https://www.dotnetperls.com/delegate)  
**Anonymous functions.** This term includes both delegates and lambda syntaxes. An anonymous function has no name. Perhaps it is running from the law.[**Anonymous Functions**](https://www.dotnetperls.com/anonymous-function)

**Overloading:**Because it has no name, method overloading is not possible for anonymous functions.

[**Overload**](https://www.dotnetperls.com/overload)

**Note:**Many developers regard lambda expressions as a complete improvement over (and replacement for) the delegate syntax.

**Predicate.** Here we use this type with an int parameter. With a lambda expression, we specify that the function returns true if the argument is equal to 5.[**Predicate**](https://www.dotnetperls.com/predicate)

**Invoke:**In this program, the Invoke method is used to show that the Predicate works as expected.

**C# program that uses Predicate**

using System;

class Program

{

static void Main()

{

**Predicate**<int> predicate = value => value == 5;

Console.WriteLine(predicate.Invoke(4));

Console.WriteLine(predicate.Invoke(5));

}

}

**Output**

False

True

**Comparison.** This type is specifically used to compare objects. It is useful when calling the List.Sort or Array.Sort methods. It can be used with any object type.[**Comparison**](https://www.dotnetperls.com/comparison)

**Performance:**Using methods such as List.Sort or Array.Sort (with a Comparison) is often faster than using LINQ to sort on a property.

**Events.** Like any other method, events can be specified as lambda expressions. With events, many event handlers are called when a certain thing happens. This can simplify some programs.[**Event**](https://www.dotnetperls.com/event)  
**Performance.** I benchmarked a lambda against an anonymous method, one using the delegate keyword. I used the functions as arguments to the Count() extension.

**Result:**I found no differences. The lambda expression performed the same as the explicit Func instance.

**Thus:**Lambda expressions cause no excess performance hit beyond other delegate syntaxes.

**Locals used in benchmark: C#**

int[] array = { 1 };

Func<int, bool> f = delegate(int x)

{

return x == 1;

};

**Lambda expression tested: C#**

int c = array.Count(element => element == 1);

**Delegate tested: C#**

int c = array.Count(f);

**Expression-bodied methods.** A method can be specified with lambda expression syntax. We provide a method name, and the method is compiled like a lambda. A "return" statement is implicit.[**Return: Expression-Bodied**](https://www.dotnetperls.com/return)

**C# program that uses lambda syntax, method**

class Program

{

static int TreeBranches(int branches, int trunks) => (branches \* trunks);

static void Main()

{

// Call the expression-bodied method.

System.Console.WriteLine(TreeBranches(10, 2));

}

}

**Output**

20

**Expressive power.** Lambdas advance a language. We can achieve the same thing with regular, non-lambda methods. But they make a language easier to use, more "expressive."

Higher-order procedures can serve as powerful abstraction mechanisms, vastly increasing the expressive power of our language.

[**Structure and Interpretation of Computer Programs: MIT**](https://mitpress.mit.edu/sicp/full-text/book/book.html)  
**Specification.** The C# language specification describes anonymous function types. The annotated edition of The C# Programming Language (3rd Edition) covers all syntaxes available.

**Tip:**We can find more detail on this topic using the precise technical terminology on page 314 of this book.

**Boring:**This is pretty boring. Proceed at your own risk. Unless you are thinking about making a C# website, it may not be worth the effort.

**Some help.** Lambdas have unique syntactic rules. We had some help from the C# specification itself. We used lambdas with zero, one or many arguments, and with a return value.  
**Anonymous functions.** These have no names, but we learned lots of their details. With the delegate keyword, we also specify method objects.