# C# Succinctly

**By**

**Joe Mayo**

Foreword by Daniel Jebaraj

Copyright © 2015 by Syncfusion, Inc.

2501 Aerial Center Parkway

Suite 200

Morrisville, NC 27560

USA

All rights reserved.

# About the Author

Joe Mayo is an author, a consultant at Mayo Software, LLC, and an instructor who specializes in Microsoft .NET technology. Joe has written several books, including *C# Unleashed* (Sams) and *LINQ Programming* (McGraw-Hill), and coauthored *ASP.NET 2.0 MVP Hacks and Tips* (Wrox)*.* His articles have been published in CODE Magazine and the online publications Inform IT and C# Station.

Joe is a regular presenter on .NET topics and has received multiple Microsoft Visual C# MVP awards. His open source project, [LINQ to Twitter,](https://github.com/JoeMayo/LinqToTwitter) is hosted on GitHub, and you can read his blog at [Geeks with Blogs.](http://www.geekswithblogs.net/WinAZ) You can find Joe on Twitter as [@JoeMayo.](https://twitter.com/JoeMayo)

# Chapter 1 Introducing C# and .NET

Welcome to *C# Succinctly*. True to the *Succinctly* series concept, this book is very focused on a single topic: the C# programming language. I might briefly mention some technologies that you can write with C# or explain how a feature fits into those technologies, but the whole of this book is about helping you become familiar with C# syntax.

In this chapter, I’ll start with some introductory information and then jump straight into a simple C# program.

## What is .NET? s

.NET is a platform that includes languages, a runtime, and framework libraries, allowing developers to create many types of applications. C# is one of the .NET languages, which also includes Visual Basic, F#, C++, and more.

### Starting a New Program

You’ll need an editor or Integrated Development Environment (IDE) to write code. Microsoft offers Visual Studio (VS), which is available via Community Edition as a free download for training and individual purposes ([https://www.visualstudio.com/en-us/products/vs-2015-producteditions.aspx)](https://www.visualstudio.com/en-us/products/vs-2015-product-editions.aspx). There are other development tools, but you can also use any editor, including Notepad. Notepad++ is another editor that does syntax highlighting, but there are many more available. Essentially, you just need the ability to type a text document. Pick your editor or IDE of choice and it will work for all programs in this book.

To get started, we need a program to run. In VS, select **File** > **New** > **Project**, then select

**Installed** > **Templates** > **Visual C#** in the tree on the left, and finally select the **Console Application** project type. Name the solution **Chapter01**, name the project **Greetings**, set the location to your preference, and click **OK**. This will create a new solution for you. Delete the **Program.cs** file and add a **Greetings.cs** file. In any text editor, just create a file named

**Greetings.cs**. The following is a C# program that prints a greeting to the command line.

|  |
| --- |
| using System;    class Greetings  { static void Main()  {  Console.WriteLine("Greetings!");  }  } |

*Code Listing 1*

The **class** is a container for code, defining a type, named **Greetings**. A **class** has members and this example shows a method member named **Main**. A method is similar to functions and procedures in other programming languages. For desktop application types, like console or WPF, naming a method **Main** tells the C# compiler where the program begins executing. Both the **Greetings** class and **Main** method have curly braces, referred to as a block, indicating beginning and ending scope.

## Namespaces and Code Organization

There are various ways to organize code and the choice should be based on the standards of your team and the nature of the project you’re building. One of the common ways to organize code is with the C# namespace feature. Here’s a hierarchical description of where namespaces fit into the overall structure of a program:

Namespace

Type

Type Members

using static System.Math;

namespace Syncfusion

{

public class Calc

{

public static double Pythagorean(double a, double b)

{

double cSquared = Pow(a, 2) + Pow(b, 2);  
 return Sqrt(cSquared);

}

}

}

*Code Listing 2*

The **Calc** class is a member of the **Syncfusion** namespace. The **Pythagorean** method is a member of the **Calc** class. A method is a block of code with a name, parameters, and return value that you can call from other code. This follows the namespace, class, member organization.

**System** is a namespace in the FCL and **Math** is a class in the **System** namespace. The **using static** clause allows the code to use static members of the **Math** class without full qualification. Instead of writing **Math.Pow(a, 2)**, which squares the value of **a**, you can use the shorthand syntax in the **Pythagorean** method. The **Pythagorean** method uses **Math.Sqrt**, which provides square root, similarly. The following sample shows how you can use this code.

|  |
| --- |
| using Syncfusion; using System;    using Crypto = System.Security.Cryptography;    namespace NamespaceDemo  { class Program  {  static void Main()  {  double hypotenuse = Calc.Pythagorean(2, 3); Console.WriteLine("Hypotenuse: " + hypotenuse);  Crypto.AesManaged aes = new Crypto.AesManaged();  Console.ReadKey();  }  }  } |

*Code Listing 3*

The **Main** method calls the **Pythagorean** method of the **Calc** class, passing arguments **2** and **3** and receiving a result in **hypotenuse**. Since **Calc** is in the **Syncfusion** namespace, the code adds a **using** clause for **Syncfusion** to the top of the file. Had the code not included that **using**

clause, **Main** would have been required to use the fully qualified name, **Syncfusion.Calc.Pythagorean**.

### Running the Program

The rest of this chapter returns to the previous Greetings program in this chapter.

In VS, click the green **Start** arrow on the toolbar and it will build and run the program. The program runs and stops so quickly that you won’t see the command-line output, so you can press **Ctrl + F5** to make the command line stay open. This book uses Visual Studio 2015, but Syncfusion has published [*Visual Studio 2013 Succinctly*,](https://www.syncfusion.com/resources/techportal/ebooks/visualstudio2013) which explains many features that are still valid in Visual Studio 2015. In the meantime, I’m going to show you how to use the C# compiler directly—the benefit being that you see what the IDE is doing for you.

Minimally, you need the .NET Framework installed on your machine, which is free for commercial as well as non-commercial use. If you installed VS, you already have the .NET Framework. Otherwise, download it from [http://www.microsoft.com/en-](http://www.microsoft.com/en-us/download/details.aspx?id=30653)

[us/download/details.aspx?id=30653](http://www.microsoft.com/en-us/download/details.aspx?id=30653) and install it. This link is for .NET Framework 4.5, but any future version should work fine.

## Summary

This chapter included a couple broader takeaways regarding how C# fits into the .NET Framework ecosystem and how to create a C# program. Remember that C# is a programming language, but it builds programs that use the FCL to run applications managed by the CLR. What this gives you is the ability to compile programs into assemblies that can be deployed and run on any machine that supports the CLR. The program entry point is the **Main** method. You can use any editor or an IDE like Visual Studio to write your code. To run a program, press **F5** in VS or compile with **csc.exe** on the command line. To deploy, copy the program to a machine with the CLR installed. In the next chapter, you’ll learn more about how to code logic in C# using expressions and statements.

# Chapter 2 Coding Expressions and Statements

For efficiency, many of the examples in the rest of the book are snippets, but you can still add these statements inside of a **Main** method to compile and get a better feel for C# syntax. There will be plenty of complete programs too.

## Looping Statements

C# supports several loops, including **for**, **foreach**, **while**, and **do**. The code listings that follow perform similar logic.

double[] temperatures = { 72.3, 73.8, 75.1, 74.9 };   
for (int i = 0; i < temperatures.Length; i++)   
{

Console.WriteLine(i);

}

*Code Listing 30*

The **for** loop initializes **i** to **0**, makes sure **i** is less than the number of items in the **temperature** array, executes the **Console.WriteLine**, and then increments **i**. It continues executing until the condition (**i < temperatures.Length**) is false, and then moves on to the next statement in the program.

|  |
| --- |
| foreach (int temperature in temperatures)  {  Console.WriteLine(temperature);  } |

*Code Listing 31*

The **foreach** loop used in Code Listing 31 is simpler and will execute for each value in the **temperatures** array.

Next is an example of a **while** loop.

int tempCount = 0;

while (tempCount < temperatures.Length)

{

Console.WriteLine(tempCount); tempCount++;

}

*Code Listing 32*

The **while** loop evaluates the condition and executes if it’s true. Notice that I initialized **tempCount** to **0** and increment **tempCount** inside of the loop on each iteration.

Finally, the following example shows how to write a **do**-**while** loop.

|  |
| --- |
| int tempCount2 = 0; do  {  Console.WriteLine(tempCount2++);  }  while (tempCount2 <= temperatures.Length); |

*Code Listing 33*

A **do**-**while** loop is good for when you want to execute logic at least one time. This example increments **tempCount2** as a parameter to **Console.WriteLine**. Remember, the postfix operator changes the variable after evaluation.

## Summary

C# has a full set of operators and types that allow you to write a wide range of expressions and statements. With branching statements and loops, you can write logic of your choosing. All of the code in this chapter has been in the **Main** method, but clearly that’s inadequate and you’ll quickly grow out of that. The next chapter explores some new C# features to help organize code with methods and properties.