Get Started

A tour of the C# language

C# (pronounced "See Sharp") is a modern, object-oriented, and type-safe programming language. C# enables developers to build many types of secure and robust applications that run in .NET. C# has its roots in the C family of languages and will be immediately familiar to C, C++, Java, and JavaScript programmers. This tour provides an overview of the major components of the language in C# 8 and earlier. If you want to explore the language through interactive examples, try the introduction to C# tutorials.

C# is an object-oriented, component-oriented programming language. C# provides language constructs to directly support these concepts, making C# a natural language in which to create and use software components. Since its origin, C# has added features to support new workloads and emerging software design practices. At its core, C# is an object-oriented language. You define types and their behavior.

Several C# features help create robust and durable applications. Garbage collection automatically reclaims memory occupied by unreachable unused objects. Nullable types guard against variables that don't refer to allocated objects. Exception handling provides a structured and extensible approach to error detection and recovery. Lambda expressions support functional programming techniques. Language Integrated Query (LINQ) syntax creates a common pattern for working with data from any source. Language support for asynchronous operations provides syntax for building distributed systems. C# has a unified type system. All C# types, including primitive types such as `int` and `double`, inherit from a single root `object` type. All types share a set of common operations. Values of any type can be stored, transported, and operated upon in a consistent manner. Furthermore, C# supports both user-defined reference types and value types. C# allows dynamic allocation of objects and in-line storage of lightweight structures. C# supports generic methods and types, which provide increased type safety and performance. C# provides iterators, which enable implementers of collection classes to define custom behaviors for client code.

C# emphasizes versioning to ensure programs and libraries can evolve over time in a compatible manner. Aspects of C#'s design that were directly influenced by versioning considerations include the separate `virtual` and `override` modifiers, the rules for method overload resolution, and support for explicit interface member declarations.

.NET architecture

C# programs run on .NET, a virtual execution system called the common language runtime (CLR) and a set of class libraries. The CLR is the implementation by Microsoft of the common language infrastructure (CLI), an international standard. The CLI is the basis for creating execution and development environments in which languages and libraries work together seamlessly.

Source code written in C# is compiled into an intermediate language (IL) that conforms to the CLI specification. The IL code and resources, such as bitmaps and strings, are stored in an assembly, typically with an extension of .dll. An assembly contains a manifest that provides information about the assembly's types, version, and culture.

When the C# program is executed, the assembly is loaded into the CLR. The CLR performs Just-In-Time (JIT) compilation to convert the IL code to native machine instructions. The CLR provides other services related to automatic garbage collection, exception handling, and resource management. Code that's executed by the CLR is sometimes referred to as "managed code," in contrast to "unmanaged code," which is compiled into native machine language that targets a specific platform.

Language interoperability is a key feature of .NET. IL code produced by the C# compiler conforms to the Common Type Specification (CTS). IL code generated from C# can interact with code that was generated from the .NET versions of F#, Visual Basic, C++, or any of more than 20 other CTS-compliant languages. A single assembly may contain multiple modules written in different .NET languages, and the types can reference each other as if they were written in the same language.

In addition to the run time services, .NET also includes extensive libraries. These libraries support many different workloads. They're organized into namespaces that provide a wide variety of useful functionality for everything from file input and output to string manipulation to XML parsing, to web application frameworks to Windows Forms controls. The typical C# application uses the .NET class library extensively to handle common "plumbing" chores.

For more information about .NET, see Overview of .NET.

Tutorials

Hello world

In the Hello world tutorial, you'll create the most basic C# program. You'll explore the `string` type and how to work with text. You can also use the path on Microsoft Learn or Jupyter on Binder.

Numbers in C#

In the Numbers in C# tutorial, you'll learn how computers store numbers and how to perform calculations with different numeric types. You'll learn the basics of rounding, and how to perform mathematical calculations using C#. This tutorial is also available to run locally on your machine.

This tutorial assumes that you've finished the Hello world lesson.

Branches and loops

The Branches and loops tutorial teaches the basics of selecting different paths of code execution based on the values stored in variables. You'll learn the basics of control flow, which is the basis of how programs make decisions and choose different actions. This tutorial is also available to run locally on your machine.

This tutorial assumes that you've finished the Hello world and Numbers in C# lessons.

List collection

The List collection lesson gives you a tour of the List collection type that stores sequences of data. You'll learn how to add and remove items, search for items, and sort the lists. You'll explore different kinds of lists. This tutorial is also available to run locally on your machine.

This tutorial assumes that you've finished the lessons listed above.

101 Linq Samples

This sample requires the dotnet-try global tool. Once you install the tool, and clone the try-samples repo, you can learn Language Integrated Query (LINQ) through a set of 101 samples you can run interactively. You can explore different ways to query, explore, and transform data sequences.

Fundamental

Main() and command-line arguments

The `Main` method is the entry point of a C# application. (Libraries and services do not require a `Main` method as an entry point.) When the application is started, the `Main` method is the first method that is invoked.

There can only be one entry point in a C# program. If you have more than one class that has a `Main` method, you must compile your program with the StartupObject compiler option to specify which `Main` method to use as the entry point. For more information, see StartupObject (C# Compiler Options).

Overview

- The `Main` method is the entry point of an executable program; it is where the program control starts and ends.
- `Main` is declared inside a class or struct. `Main` must be `static` and it need not be `public`. (In the earlier example, it receives the default access of `private`.) The enclosing class or struct is not required to be static.
- `Main` can either have a `void`, `int`, or, starting with C# 7.1, `Task`, or `Task<int>` return type.
- If and only if `Main` returns a `Task` or `Task<int>`, the declaration of `Main` may include the `async` modifier. This specifically excludes an `async void Main` method.
- The `Main` method can be declared with or without a `string[]` parameter that contains command-line arguments. When using Visual Studio to create Windows applications, you can add the parameter manually or else use the GetCommandLineArgs() method to obtain the command-line arguments. Parameters are read as zero-indexed command-line arguments. Unlike C and C++, the name of the program is not treated as the first command-line argument in the `args` array, but it is the first element of the GetCommandLineArgs() method.

What's New In C#

What's new in C# 10.0

C# 10.0 adds the following features and enhancements to the C# language (as of .NET 6 Preview 7):

- `global using` directives
- File-scoped namespace declaration
- Extended property patterns
- Allow `const` interpolated strings
- Record types can seal `ToString()`
- Allow both assignment and declaration in the same deconstruction
- Allow `AsyncMethodBuilder` attribute on methods

Some of the features you can try are available only when you set your language version to "preview". These features may have more refinements in future previews before .NET 6.0 is released.

C# 10.0 is supported on .NET 6. For more information, see C# language versioning.

You can download the latest .NET 6.0 SDK from the .NET downloads page. You can also download Visual Studio 2022 preview, which includes the .NET 6.0 preview SDK.

Global using directives

You can add the `global` modifier to any using directive to instruct the compiler that the directive applies to all source files in the compilation. This is typically all source files in a project.

File-scoped namespace declaration

You can use a new form of the `namespace` declaration to declare that all subsequent declarations are members of the declared namespace:

C#Copy

```
namespace MyNamespace;
```

This new syntax saves both horizontal and vertical space for the most common `namespace` declarations.

Extended property patterns

Beginning with C# 10.0, you can reference nested properties or fields within a property pattern. For example, a pattern of the form

C#Copy

```
{ Prop1.Prop2: pattern }
```

is valid in C# 10.0 and later and equivalent to

C#Copy

```
{ Prop1: { Prop2: pattern } }
```

that is valid in C# 8.0 and later.

For more information, see the Extended property patterns feature proposal note. For more information about a property pattern, see the Property pattern section of the Patterns article.

Constant interpolated strings

In C# 10.0, `const` strings may be initialized using string interpolation if all the placeholders are themselves constant strings. String interpolation can create more readable constant strings as you build constant strings used in your application. The placeholder expressions can't be numeric constants because those constants are converted to strings at runtime. The current culture may affect their string representation. Learn more in the language reference on `const` expressions.

Note

When using .NET 6.0 preview 5, this feature requires setting the `<LangVersion>` element in your csproj file to `preview`.

Record types can seal ToString

In C# 10.0, you can add the `sealed` modifier when you override `ToString` in a record type. Sealing the `ToString` method prevents the compiler from synthesizing a `ToString` method for any derived record types. This ensures all derived record types use the `ToString` method defined in a common base record type. You can learn more about this feature in the article on records.

Note

When using .NET 6.0 preview 5, this feature requires setting the `<LangVersion>` element in your csproj file to `preview`.

Assignment and declaration in same deconstruction

This change removes a restriction from earlier versions of C#. Previously, a deconstruction could assign all values to existing variables, or initialize newly declared variables:

C#Copy

```
// Initialization:
(int x, int y) = point;

// assignment:
int x1 = 0;
int y1 = 0;
(x1, y1) = point;
```

C# 10.0 removes this restriction:

C#Copy

```
int x = 0;
(x, int y) = point;
```

Note

When using .NET 6.0 preview 5, this feature requires setting the `<LangVersion>` element in your csproj file to `preview`.

Allow AsyncMethodBuilder attribute on methods

In C# 10.0 and later, you can specify a different async method builder for a single method, in addition to specifying the method builder type for all methods that return a given task-like type. This enables advanced performance tuning scenarios where a given method may benefit from a custom builder.

To learn more, see the section on `AsyncMethodBuilder` in the article on attributes read by the compiler.

Rest Client

Section 1.10.32 of "de Finibus Bonorum et Malorum", written by Cicero in 45 BC

"Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt. Neque porro quisquam est, qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit, sed quia non numquam eius modi tempora incidunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim ad minima veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur? Quis autem vel eum iure reprehenderit qui in ea voluptate velit esse quam nihil molestiae consequatur, vel illum qui dolorem eum fugiat quo voluptas nulla pariatur?"

1914 translation by H. Rackham

"But I must explain to you how all this mistaken idea of denouncing pleasure and praising pain was born and I will give you a complete account of the system, and expound the actual teachings of the great explorer of the truth, the master-builder of human happiness. No one rejects, dislikes, or avoids pleasure itself, because it is pleasure, but because those who do not know how to pursue pleasure rationally encounter consequences that are extremely painful. Nor again is there anyone who loves or pursues or desires to obtain pain of itself, because it is pain, but because occasionally circumstances occur in which toil and pain can procure him some great pleasure. To take a trivial example, which of us ever undertakes laborious physical exercise, except to obtain some advantage from it? But who has any right to find fault with a man who chooses to enjoy a pleasure that has no annoying consequences, or one who avoids a pain that produces no resultant pleasure?"

Section 1.10.33 of "de Finibus Bonorum et Malorum", written by Cicero in 45 BC

"At vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis praesentium voluptatum deleniti atque corrupti quos dolores et quas molestias excepturi sint occaecati cupiditate non provident, similique sunt in culpa qui officia deserunt mollitia animi, id est laborum et dolorum fuga. Et harum quidem rerum facilis est et expedita distinctio. Nam libero tempore, cum soluta nobis est eligendi optio cumque nihil impedit quo minus id quod maxime placeat facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et aut officiis debitis aut rerum necessitatibus saepe eveniet ut et voluptates repudiandae sint et molestiae non recusandae. Itaque earum rerum hic tenetur a sapiente delectus, ut aut reiciendis voluptatibus maiores alias consequatur aut perferendis doloribus asperiores repellat."

1914 translation by H. Rackham

"On the other hand, we denounce with righteous indignation and dislike men who are so beguiled and demoralized by the charms of pleasure of the moment, so blinded by desire, that they cannot foresee the pain and trouble that are bound to ensue; and equal blame belongs to those who fail in their duty through weakness of will, which is the same as saying through shrinking from toil and pain. These cases are perfectly simple and easy to distinguish. In a free hour, when our power of choice is untrammelled and when nothing prevents our being able to do what we like best, every pleasure is to be welcomed and every pain avoided. But in certain circumstances and owing to the claims of duty or the obligations of business it will frequently occur that pleasures have to be repudiated and annoyances accepted. The wise man therefore always holds in these matters to this principle of selection: he rejects pleasures to secure other greater pleasures, or else he endures pains to avoid worse pains."

C# 7.0

What's new in C# 7.0 through C# 7.3

C# 7.0 through C# 7.3 brought a number of features and incremental improvements to your development experience with C#. This article provides an overview of the new language features and compiler options. The descriptions describe the behavior for C# 7.3, which is the most recent version supported for .NET Framework-based applications.

The language version selection configuration element was added with C# 7.1, which enables you to specify the compiler language version in your project file.

C# 7.0-7.3 adds these features and themes to the C# language:

Tuples and discards

> You can create lightweight, unnamed types that contain multiple public fields. Compilers and IDE tools understand the semantics of these types.
>
> Discards are temporary, write-only variables used in assignments when you don't care about the value assigned. They're most useful when