**Lecture 1. C# and Microsoft .Net Framework**

The history and foundations. The basic types and program structure. Visual Studio.

**.NET Framework**

|  |  |
| --- | --- |
| **.NET Framework** | |
| [Microsoft .NET Framework v4.5 logo.png](http://en.wikipedia.org/wiki/File:Microsoft_.NET_Framework_v4.5_logo.png) | |
| [The .NET Framework stack](http://en.wikipedia.org/wiki/File:DotNet.svg) | |
| [**Developer(s)**](http://en.wikipedia.org/wiki/Software_developer) | [Microsoft](http://en.wikipedia.org/wiki/Microsoft) |
| **Initial release** | 13 February 2002 |
| [**Stable release**](http://en.wikipedia.org/wiki/Software_release_life_cycle) | 4.5.2 (4.5.51209.34209) / 6 May 2014 |
| [**Operating system**](http://en.wikipedia.org/wiki/Operating_system) | [Windows 98](http://en.wikipedia.org/wiki/Windows_98) or later, [Windows NT 4.0](http://en.wikipedia.org/wiki/Windows_NT_4.0) or later |
| [**Type**](http://en.wikipedia.org/wiki/List_of_software_categories) | [Software framework](http://en.wikipedia.org/wiki/Software_framework) |
| [**License**](http://en.wikipedia.org/wiki/Software_license) | * .NET Core: [MIT License](http://en.wikipedia.org/wiki/MIT_License)([Open-source](http://en.wikipedia.org/wiki/Open-source))[[1]](http://en.wikipedia.org/w/index.php?title=.NET_Framework&printable=yes#cite_note-landwerth-dotnetcore-oss-1) * [FCL](http://en.wikipedia.org/wiki/Framework_Class_Library): [Ms-RSL](http://en.wikipedia.org/wiki/Shared_source#Microsoft_Reference_Source_License_.28Ms-RSL.29) ([Shared-source](http://en.wikipedia.org/wiki/Shared-source))[[2]](http://en.wikipedia.org/w/index.php?title=.NET_Framework&printable=yes#cite_note-sourcerelease-2) * The rest: [Proprietary](http://en.wikipedia.org/wiki/Proprietary_software)(Same license as Windows) |
| **Website** | [microsoft.com/net](http://microsoft.com/net) |

The Microsoft **.NET Framework** (pronounced *dot net*) is a [software framework](http://en.wikipedia.org/wiki/Software_framework) developed by [Microsoft](http://en.wikipedia.org/wiki/Microsoft) that runs primarily on [Microsoft Windows](http://en.wikipedia.org/wiki/Microsoft_Windows). It includes a large [class library](http://en.wikipedia.org/wiki/Class_library) known as[Framework Class Library](http://en.wikipedia.org/wiki/Framework_Class_Library) (FCL) and provides [language interoperability](http://en.wikipedia.org/wiki/Language_interoperability) (each language can use code written in other languages) across several [programming languages](http://en.wikipedia.org/wiki/Programming_language). Programs written for .NET Framework execute in a [software](http://en.wikipedia.org/wiki/Software) environment (as contrasted to [hardware](http://en.wikipedia.org/wiki/Computer_hardware) environment), known as [Common Language Runtime](http://en.wikipedia.org/wiki/Common_Language_Runtime) (CLR), an [application virtual machine](http://en.wikipedia.org/wiki/Process_virtual_machine) that provides services such as security, [memory management](http://en.wikipedia.org/wiki/Memory_management), and [exception handling](http://en.wikipedia.org/wiki/Exception_handling). FCL and CLR together constitute .NET Framework.

FCL provides [user interface](http://en.wikipedia.org/wiki/User_interface), [data access](http://en.wikipedia.org/wiki/Data_access), [database connectivity](http://en.wikipedia.org/wiki/Database_connection), [cryptography](http://en.wikipedia.org/wiki/Cryptography), [web application](http://en.wikipedia.org/wiki/Web_application) development, numeric [algorithms](http://en.wikipedia.org/wiki/Algorithm), and [network communications](http://en.wikipedia.org/wiki/Computer_networking). Programmers produce software by combining their own [source code](http://en.wikipedia.org/wiki/Source_code) with .NET Framework and other libraries. .NET Framework is intended to be used by most new applications created for Windows platform. Microsoft also produces an [integrated development environment](http://en.wikipedia.org/wiki/Integrated_development_environment) largely for .NET software called [Visual Studio](http://en.wikipedia.org/wiki/Microsoft_Visual_Studio).

**History**

Microsoft started development of .NET Framework in the late 1990s, originally under the name of Next Generation Windows Services (NGWS). By late 2000, the first beta versions of .NET 1.0 were released.

Version 3.0 of .NET Framework is included with [Windows Server 2008](http://en.wikipedia.org/wiki/Windows_Server_2008) and [Windows Vista](http://en.wikipedia.org/wiki/Windows_Vista). Version 3.5 is included with [Windows 7](http://en.wikipedia.org/wiki/Windows_7) and [Windows Server 2008 R2](http://en.wikipedia.org/wiki/Windows_Server_2008_R2), and can also be installed on[Windows XP](http://en.wikipedia.org/wiki/Windows_XP) and [Windows Server 2003](http://en.wikipedia.org/wiki/Windows_Server_2003).[[3]](http://en.wikipedia.org/w/index.php?title=.NET_Framework&printable=yes#cite_note-3) Version 4.5 is included with [Windows 8](http://en.wikipedia.org/wiki/Windows_8) and [Windows Server 2012](http://en.wikipedia.org/wiki/Windows_Server_2012). Windows XP was delivered without any versions of .NET Framework.[[4]](http://en.wikipedia.org/w/index.php?title=.NET_Framework&printable=yes#cite_note-4)

.NET Framework family also includes two versions for [mobile](http://en.wikipedia.org/wiki/Mobile_computing) or [embedded device](http://en.wikipedia.org/wiki/Embedded_device) use. A reduced version of the framework, [.NET Compact Framework](http://en.wikipedia.org/wiki/.NET_Compact_Framework), is available on [Windows CE](http://en.wikipedia.org/wiki/Windows_CE) platforms, including [Windows Mobile](http://en.wikipedia.org/wiki/Windows_Mobile) devices such as [smartphones](http://en.wikipedia.org/wiki/Smartphones). Additionally, [.NET Micro Framework](http://en.wikipedia.org/wiki/.NET_Micro_Framework) is targeted at severely resource-constrained devices.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Overview of .NET Framework release history**[[5]](http://en.wikipedia.org/w/index.php?title=.NET_Framework&printable=yes#cite_note-depend-5) | | | | | | |
| **Version number** | [**CLR**](http://en.wikipedia.org/wiki/Common_Language_Runtime) **version** | **Release date** | **Included in** | | | **Replaces** |
| **Development tool** | **Windows** | **Windows Server** |
| [1.0](http://en.wikipedia.org/wiki/.NET_Framework_version_history#.NET_Framework_1.0) | 1.0 | 2002-02-13 | [Visual Studio .NET](http://en.wikipedia.org/wiki/Visual_Studio_.NET)[[6]](http://en.wikipedia.org/w/index.php?title=.NET_Framework&printable=yes#cite_note-new-vs2003-6) | N/A | N/A | N/A |
| [1.1](http://en.wikipedia.org/wiki/.NET_Framework_version_history#.NET_Framework_1.1) | 1.1 | 2003-04-24 | [Visual Studio .NET 2003](http://en.wikipedia.org/wiki/Visual_Studio_.NET_2003)[[6]](http://en.wikipedia.org/w/index.php?title=.NET_Framework&printable=yes#cite_note-new-vs2003-6) | N/A | [2003](http://en.wikipedia.org/wiki/Windows_Server_2003) | 1.0[[7]](http://en.wikipedia.org/w/index.php?title=.NET_Framework&printable=yes#cite_note-7) |
| [2.0](http://en.wikipedia.org/wiki/.NET_Framework_version_history#.NET_Framework_2.0) | 2.0 | 2005-11-07 | [Visual Studio 2005](http://en.wikipedia.org/wiki/Visual_Studio_2005)[[8]](http://en.wikipedia.org/w/index.php?title=.NET_Framework&printable=yes#cite_note-new-vs2005-8) | N/A | [2003](http://en.wikipedia.org/wiki/Windows_Server_2003), [2003 R2](http://en.wikipedia.org/wiki/Windows_Server_2003_R2),[[9]](http://en.wikipedia.org/w/index.php?title=.NET_Framework&printable=yes#cite_note-9) [2008 SP2](http://en.wikipedia.org/wiki/Windows_Server_2008_SP2), [2008 R2 SP1](http://en.wikipedia.org/wiki/Windows_Server_2008_R2_SP1) | N/A |
| [3.0](http://en.wikipedia.org/wiki/.NET_Framework_version_history#.NET_Framework_3.0) | 2.0 | 2006-11-06 | [Expression Blend](http://en.wikipedia.org/wiki/Microsoft_Blend)[[10]](http://en.wikipedia.org/w/index.php?title=.NET_Framework&printable=yes#cite_note-new-blend-10)[[a]](http://en.wikipedia.org/w/index.php?title=.NET_Framework&printable=yes#endnote_a1none) | [Vista](http://en.wikipedia.org/wiki/Windows_Vista) | [2008 SP2](http://en.wikipedia.org/wiki/Windows_Server_2008_SP2), [2008 R2 SP1](http://en.wikipedia.org/wiki/Windows_Server_2008_R2_SP1) | 2.0[[5]](http://en.wikipedia.org/w/index.php?title=.NET_Framework&printable=yes#cite_note-depend-5) |
| [3.5](http://en.wikipedia.org/wiki/.NET_Framework_version_history#.NET_Framework_3.5) | 2.0 | 2007-11-19 | [Visual Studio 2008](http://en.wikipedia.org/wiki/Visual_Studio_2008)[[11]](http://en.wikipedia.org/w/index.php?title=.NET_Framework&printable=yes#cite_note-new-vs2008-11) | [7](http://en.wikipedia.org/wiki/Windows_7), [8](http://en.wikipedia.org/wiki/Windows_8),[[b]](http://en.wikipedia.org/w/index.php?title=.NET_Framework&printable=yes#endnote_b1none) [8.1](http://en.wikipedia.org/wiki/Windows_8.1)[[b]](http://en.wikipedia.org/w/index.php?title=.NET_Framework&printable=yes#endnote_b2none) | [2008 R2 SP1](http://en.wikipedia.org/wiki/Windows_Server_2008_R2_SP1) | 2.0, 3.0[[5]](http://en.wikipedia.org/w/index.php?title=.NET_Framework&printable=yes#cite_note-depend-5) |
| [4.0](http://en.wikipedia.org/wiki/.NET_Framework_version_history#.NET_Framework_4) | 4 | 2010-04-12 | [Visual Studio 2010](http://en.wikipedia.org/wiki/Visual_Studio_2010)[[12]](http://en.wikipedia.org/w/index.php?title=.NET_Framework&printable=yes#cite_note-new-vs2010-12) | N/A | N/A | N/A |
| [4.5](http://en.wikipedia.org/wiki/.NET_Framework_version_history#.NET_Framework_4.5) | 4 | 2012-08-15 | [Visual Studio 2012](http://en.wikipedia.org/wiki/Visual_Studio_2012)[[13]](http://en.wikipedia.org/w/index.php?title=.NET_Framework&printable=yes#cite_note-new-vs2012-13) | [8](http://en.wikipedia.org/wiki/Windows_8) | [2012](http://en.wikipedia.org/wiki/Windows_Server_2012) | 4.0[[5]](http://en.wikipedia.org/w/index.php?title=.NET_Framework&printable=yes#cite_note-depend-5) |
| [4.5.1](http://en.wikipedia.org/wiki/.NET_Framework_version_history#.NET_Framework_4.5.1) | 4 | 2013-10-17 | [Visual Studio 2013](http://en.wikipedia.org/wiki/Visual_Studio_2013)[[14]](http://en.wikipedia.org/w/index.php?title=.NET_Framework&printable=yes#cite_note-new-vs2013-14) | [8.1](http://en.wikipedia.org/wiki/Windows_8.1) | [2012 R2](http://en.wikipedia.org/wiki/Windows_Server_2012_R2) | 4.0, 4.5[[5]](http://en.wikipedia.org/w/index.php?title=.NET_Framework&printable=yes#cite_note-depend-5) |
| [4.5.2](http://en.wikipedia.org/wiki/.NET_Framework_version_history#.NET_Framework_4.5.2) | 4 | 2014-05-05 | N/A | N/A | N/A | 4.0, 4.5, 4.5.1[[5]](http://en.wikipedia.org/w/index.php?title=.NET_Framework&printable=yes#cite_note-depend-5) |

**Design**

**Interoperability**

Because computer systems commonly require interaction between newer and older applications, .NET Framework provides means to access functionality implemented in newer and older programs that execute outside .NET environment. Access to [COM](http://en.wikipedia.org/wiki/Component_Object_Model) components is provided in System.Runtime.InteropServices and System.EnterpriseServices namespaces of the framework; access to other functionality is achieved using the [P/Invoke](http://en.wikipedia.org/wiki/Platform_Invocation_Services) feature.

**Common Language Runtime engine**

[Common Language Runtime](http://en.wikipedia.org/wiki/Common_Language_Runtime) (CLR) serves as the execution engine of .NET Framework. All .NET programs execute under the supervision of CLR, guaranteeing certain properties and behaviors in the areas of memory management, security, and exception handling.

**Language independence**

.NET Framework introduces a [Common Type System](http://en.wikipedia.org/wiki/Common_Type_System), or CTS. CTS [specification](http://en.wikipedia.org/wiki/Specification) defines all possible [datatypes](http://en.wikipedia.org/wiki/Datatypes) and [programming](http://en.wikipedia.org/wiki/Programming_language) constructs supported by CLR and how they may or may not interact with each other conforming to [Common Language Infrastructure](http://en.wikipedia.org/wiki/Common_Language_Infrastructure) (CLI) specification. Because of this feature, .NET Framework supports the exchange of types and object instances between libraries and applications written using [any conforming .NET language](http://en.wikipedia.org/wiki/List_of_CLI_languages).

**Framework Class Library**

[Framework Class Library](http://en.wikipedia.org/wiki/Framework_Class_Library) (FCL) is a library of functionality available to all languages using .NET Framework. FCL provides [classes](http://en.wikipedia.org/wiki/Class_(computer_science)) that encapsulate a number of common functions, including [file](http://en.wikipedia.org/wiki/Computer_file) reading and writing, [graphic rendering](http://en.wikipedia.org/wiki/Rendering_(computer_graphics)),[database](http://en.wikipedia.org/wiki/Database) interaction, [XML](http://en.wikipedia.org/wiki/XML) document manipulation, and so on. It consists of classes, interfaces of reusable types that integrates CLR.

**Simplified deployment**

.NET Framework includes design features and tools which help manage the [installation](http://en.wikipedia.org/wiki/Installation_(computer_programs)) of computer software to ensure that it does not interfere with previously installed software, and that it conforms to security requirements.

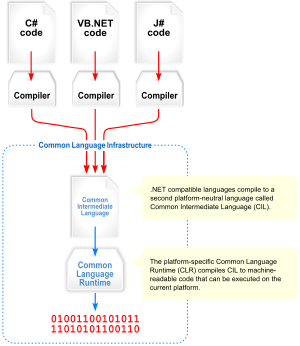
**Security**

The design addresses some of the vulnerabilities, such as [buffer overflows](http://en.wikipedia.org/wiki/Buffer_overflow), which have been exploited by malicious software. Additionally, .NET provides a common security model for all applications.

**Portability**

While Microsoft has never implemented the full framework on any system except Microsoft Windows, it has engineered the framework to be platform-agnostic,[[16]](http://en.wikipedia.org/w/index.php?title=.NET_Framework&printable=yes#cite_note-16) and cross-platform implementations are available for other operating systems (see [Silverlight](http://en.wikipedia.org/wiki/Microsoft_Silverlight) and [§ Alternative implementations](http://en.wikipedia.org/w/index.php?title=.NET_Framework&printable=yes#Alternative_implementations)). Microsoft submitted the specifications for CLI (which includes the core class libraries, CTS, and the [Common Intermediate Language](http://en.wikipedia.org/wiki/Common_Intermediate_Language)), [C#](http://en.wikipedia.org/wiki/C_Sharp_(programming_language)), and C++/CLI to both [ECMA](http://en.wikipedia.org/wiki/Ecma_International) and [ISO](http://en.wikipedia.org/wiki/International_Organization_for_Standardization), making them available as official standards. This makes it possible for third parties to create compatible implementations of the framework and its languages on other platforms.

**Architecture**

[](http://en.wikipedia.org/wiki/File:Overview_of_the_Common_Language_Infrastructure.svg)

Visual overview of the Common Language Infrastructure (CLI)

**Common Language Infrastructure (CLI)**

The purpose of CLI is to provide a language-neutral platform for application development and execution, including functions for [exception handling](http://en.wikipedia.org/wiki/Exception_handling), [garbage collection](http://en.wikipedia.org/wiki/Garbage_collection_(computer_science)), security, and interoperability. By implementing the core aspects of .NET Framework within the scope of CLI, this functionality will not be tied to a single language but will be available across the many languages supported by the framework. Microsoft's implementation of CLI is CLR.

[Common Intermediate Language](http://en.wikipedia.org/wiki/Common_Intermediate_Language) (CIL) code is housed in [CLI assemblies](http://en.wikipedia.org/wiki/Assembly_(CLI)). As mandated by the specification, assemblies are stored in [Portable Executable](http://en.wikipedia.org/wiki/Portable_Executable) (PE) format, common on Windows platform for all [DLL](http://en.wikipedia.org/wiki/Dynamic-link_library) and [EXE](http://en.wikipedia.org/wiki/EXE) files. The assembly consists of one or more files, one of which must contain the manifest, which has the [metadata](http://en.wikipedia.org/wiki/Metadata) for the assembly. The complete name of an assembly (not to be confused with the filename on disk) contains its simple text name, version number, culture, and [public key](http://en.wikipedia.org/wiki/Public_key) token. Assemblies are considered equivalent if they share the same complete name, excluding the revision of the version number. A private key can also be used by the creator of the assembly for strong naming. The public key token identifies which private key an assembly is signed with. Only the creator of the keypair (typically .NET developer signing the assembly) can sign assemblies that have the same strong name as a previous version assembly, since the creator is in possession of the private key. [Strong naming](http://en.wikipedia.org/wiki/Strong_name) is required to add assemblies to [Global Assembly Cache](http://en.wikipedia.org/wiki/Global_Assembly_Cache).

**Security**

.NET has its own security mechanism with two general features: [Code Access Security](http://en.wikipedia.org/wiki/Code_Access_Security) (CAS), and validation and verification. CAS is based on evidence that is associated with a specific assembly. Typically the evidence is the source of the assembly (whether it is installed on the local machine or has been downloaded from the intranet or Internet). CAS uses evidence to determine the permissions granted to the code. Other code can demand that calling code be granted a specified permission. The demand causes CLR to perform a call stack walk: every assembly of each method in the call stack is checked for the required permission; if any assembly is not granted the permission a security exception is thrown.

**Class library**

.NET Framework includes a set of [standard](http://en.wikipedia.org/wiki/Standard_library) class libraries. The class library is organized in a hierarchy of [namespaces](http://en.wikipedia.org/wiki/Namespaces). Most of the built-in APIs are part of either System.\* or Microsoft.\* namespaces. These class libraries implement a large number of common functions, such as file reading and writing, graphic rendering, database interaction, and XML document manipulation, among others. .NET class libraries are available to all [CLI compliant languages](http://en.wikipedia.org/wiki/List_of_CLI_languages). .NET Framework class library is divided into two parts: FCL and [Base Class Library](http://en.wikipedia.org/wiki/Base_Class_Library) (BCL).

FCL includes a small subset of the entire class library and is the core set of classes that serve as the basic [API](http://en.wikipedia.org/wiki/API) of CLR. Classes in mscorlib.dll and some classes in System.dll and System.core.dll are part of FCL. FCL classes are available in .NET Framework as well as its alternative implementations including [.NET Compact Framework](http://en.wikipedia.org/wiki/.NET_Compact_Framework), [Microsoft Silverlight](http://en.wikipedia.org/wiki/Microsoft_Silverlight) and [Mono](http://en.wikipedia.org/wiki/Mono_(software)).

BCL is a superset of FCL and refers to the entire class library that ships with .NET Framework. It includes an expanded set of libraries, including [Windows Forms](http://en.wikipedia.org/wiki/Windows_Forms), [ADO.NET](http://en.wikipedia.org/wiki/ADO.NET), [ASP.NET](http://en.wikipedia.org/wiki/ASP.NET), [Language Integrated Query](http://en.wikipedia.org/wiki/Language_Integrated_Query), [Windows Presentation Foundation](http://en.wikipedia.org/wiki/Windows_Presentation_Foundation), [Windows Communication Foundation](http://en.wikipedia.org/wiki/Windows_Communication_Foundation) among others. BCL is much larger in scope than standard libraries for languages like [C++](http://en.wikipedia.org/wiki/C%2B%2B), and comparable in scope to [standard libraries of Java](http://en.wikipedia.org/wiki/Java_Class_Library).

**Memory management**

CLR frees the developer from the burden of managing memory (allocating and freeing up when done); it handles memory management itself by detecting when memory can be safely freed. Instantiations of .NET types (objects) are allocated from the managed heap; a pool of memory managed by CLR. As long as there exists a reference to an object, which might be either a direct reference to an object or via a [graph](http://en.wikipedia.org/wiki/Graph_(data_structure)) of objects, the object is considered to be in use. When there is no reference to an object, and it cannot be reached or used, it becomes garbage, eligible for collection. .NET Framework includes a [garbage collector](http://en.wikipedia.org/wiki/Garbage_collection_(computer_science)) which runs periodically, on a separate [thread](http://en.wikipedia.org/wiki/Thread_(computing)) from the application's thread, that enumerates all the unusable objects and reclaims the memory allocated to them.

.NET [Garbage Collector](http://en.wikipedia.org/wiki/Garbage_collection_(computer_science)) (GC) is a non-deterministic, compacting, [mark-and-sweep](http://en.wikipedia.org/wiki/Tracing_garbage_collection#Na.C3.AFve_mark-and-sweep) garbage collector. GC runs only when a certain amount of memory has been used or there is enough pressure for memory on the system. Since it is not guaranteed when the conditions to reclaim memory are reached, GC runs are non-deterministic. Each .NET application has a set of roots, which are pointers to objects on the managed heap (*managed objects*). These include references to static objects and objects defined as local variables or method parameters currently in scope, as well as objects referred to by CPU registers. When GC runs, it pauses the application, and for each object referred to in the root, it [recursively](http://en.wikipedia.org/wiki/Recursion) enumerates all the objects reachable from the root objects and marks them as reachable. It uses CLI metadata and [reflection](http://en.wikipedia.org/wiki/Reflection_(computer_science)) to discover the objects encapsulated by an object, and then recursively walk them. It then enumerates all the objects on the heap (which were initially allocated contiguously) using reflection. All objects not marked as reachable are garbage. This is the *mark* phase. Since the memory held by garbage is not of any consequence, it is considered free space. However, this leaves chunks of free space between objects which were initially contiguous. The objects are then *compacted* together to make used memory contiguous again. Any reference to an object invalidated by moving the object is updated by GC to reflect the new location. The application is resumed after the garbage collection is over.

GC used by .NET Framework is also [*generational*](http://en.wikipedia.org/wiki/Garbage_collection_(computer_science)#Generational_GC_.28ephemeral_GC.29). Objects are assigned a *generation*; newly created objects belong to *Generation 0*. The objects that survive a garbage collection are tagged as *Generation 1*, and the Generation 1 objects that survive another collection are *Generation 2* objects. .NET Framework uses up to Generation 2 objects. Higher generation objects are garbage collected less frequently than lower generation objects. This helps increase the efficiency of garbage collection, as older objects tend to have a longer lifetime than newer objects. Thus, by eliminating older (and thus more likely to survive a collection) objects from the scope of a collection run, fewer objects need to be checked and compacted.

**Standardization and licensing**

In August 2000, [Microsoft](http://en.wikipedia.org/wiki/Microsoft), [Hewlett-Packard](http://en.wikipedia.org/wiki/Hewlett-Packard), and [Intel](http://en.wikipedia.org/wiki/Intel) worked to standardize CLI and [C#](http://en.wikipedia.org/wiki/C_Sharp_(programming_language)). By December 2001, both were ratified [ECMA](http://en.wikipedia.org/wiki/Ecma_International) standards. [ISO](http://en.wikipedia.org/wiki/International_Organisation_for_Standardisation) followed in April 2003. The current version of ISO standards are ISO/IEC 23271:2012 and ISO/IEC 23270:2006.

While Microsoft and their partners hold patents for CLI and C#, ECMA and ISO require that all patents essential to implementation be made available under "[reasonable and non-discriminatory terms](http://en.wikipedia.org/wiki/Reasonable_and_Non_Discriminatory_Licensing)". In addition to meeting these terms, the companies have agreed to make the patents available royalty-free. However, this did not apply for the part of .NET Framework not covered by ECMA/ISO standards, which included [Windows Forms](http://en.wikipedia.org/wiki/Windows_Forms), [ADO.NET](http://en.wikipedia.org/wiki/ADO.NET), and [ASP.NET](http://en.wikipedia.org/wiki/ASP.NET). Patents that Microsoft holds in these areas may have deterred non-Microsoft implementations of the full framework.

On 3 October 2007, Microsoft announced that much of the [source code](http://en.wikipedia.org/wiki/Source_code) for .NET Framework [Framework Class Library](http://en.wikipedia.org/wiki/Framework_Class_Library) (including [ASP.NET](http://en.wikipedia.org/wiki/ASP.NET), [ADO.NET](http://en.wikipedia.org/wiki/ADO.NET), and [Windows Presentation Foundation](http://en.wikipedia.org/wiki/Windows_Presentation_Foundation)) was to have been made available with the final release of [Visual Studio 2008](http://en.wikipedia.org/wiki/Visual_Studio_2008) towards the end of 2007 under the [shared source](http://en.wikipedia.org/wiki/Shared_source) [Microsoft Reference License](http://en.wikipedia.org/wiki/Microsoft_Reference_License). The source code for other libraries including [Windows Communication Foundation](http://en.wikipedia.org/wiki/Windows_Communication_Foundation) (WCF), [Windows Workflow Foundation](http://en.wikipedia.org/wiki/Windows_Workflow_Foundation) (WF), and [Language Integrated Query](http://en.wikipedia.org/wiki/Language_Integrated_Query) (LINQ) were to be added in future releases. Being released under the shared-source Microsoft Reference License means this source code was made available for debugging purpose only, primarily to support integrated debugging of FCL in [Visual Studio](http://en.wikipedia.org/wiki/Visual_Studio).

On 12 November 2014, Microsoft announced it would [open source](http://en.wikipedia.org/wiki/Open-source_software) .NET Core. The effort will include cross-platform support for .NET, the source release of Microsoft's CoreCLR implementation, source for the "entire [...] library stack" for .NET Core, and the adoption of a conventional ("bazaar"-like) [open source development model](http://en.wikipedia.org/wiki/Open-source_software#Development_model) under the stewardship of the [.NET Foundation](http://en.wikipedia.org/w/index.php?title=.NET_Foundation&action=edit&redlink=1). [Miguel de Icaza](http://en.wikipedia.org/wiki/Miguel_de_Icaza) describes .NET Core as a "redesigned version of .NET that is based on the simplified version of the class libraries", and Microsoft's Immo Landwerth explained that .NET Core would be "the foundation of all future .NET platforms".

At the time of the announcement, the initial release of the .NET Core project had been seeded with a subset of the libraries' source code and coincided with the relicensing of Microsoft's existing .NET reference source away from the restrictions of the Microsoft Reference License. Both projects are made available under the [MIT License](http://en.wikipedia.org/wiki/MIT_License). Landwerth explained that one of the disadvantages of the previously selected shared source license was that it "made Rotor [the Microsoft reference implementation] a non-starter" as a community-developed open source project because it did not meet the criteria of an [OSI](http://en.wikipedia.org/wiki/Open_Source_Initiative)-approved license.

Microsoft also produced an update to its patent grants, which further extends the scope beyond its previous pledges. Whereas before projects like [Mono](http://en.wikipedia.org/wiki/Mono_(software)) existed in a legal [grey area](http://en.wikipedia.org/wiki/Grey_area) because Microsoft's earlier grants applied only to the technology in "covered specifications", including strictly the 4th editions each of ECMA-334 and ECMA-335, the new patent promise places no ceiling on the specification version and even extends to any .NET runtime technologies documented on MSDN that haven't been formally specified by the ECMA group, if a project chooses to implement them. This permits Mono and other projects to maintain feature parity with modern .NET features that have been introduced since the 4th edition was published without being at risk of patent litigation over the implementation of those features. The new grant does maintain the restriction that any implementation must maintain minimum compliance with the mandatory parts of the CLI specification.

Microsoft's press release highlights that the cross-platform commitment would now allow for a fully open source server-side .NET stack. However, Microsoft does not plan to release the source for [Windows Presentation Foundation](http://en.wikipedia.org/wiki/Windows_Presentation_Foundation) or [Windows Forms](http://en.wikipedia.org/wiki/Windows_Forms).

**Alternative implementations**

.NET Framework is the predominant implementation of .NET technologies. Other implementations for parts of the framework exist. Although the runtime engine is described by an ECMA/ISO specification, other implementations of it may be encumbered by [patent issues](http://en.wikipedia.org/wiki/Software_patent); ISO standards may include the disclaimer, "Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights." It is more difficult to develop alternatives to FCL, which is not described by an open standard and may be subject to copyright restrictions. Additionally, parts of FCL have Windows-specific functionality and behavior, so implementation on non-Windows platforms can be problematic.

Some alternative implementations of parts of the framework are listed here.

* [.NET Micro Framework](http://en.wikipedia.org/wiki/.NET_Micro_Framework) is a .NET platform for extremely resource-constrained devices. It includes a small version of CLR and supports development in [C#](http://en.wikipedia.org/wiki/C_Sharp_(programming_language)) (though some developers were able to use [VB.NET](http://en.wikipedia.org/wiki/VB.NET), albeit with an amount of hacking, and with limited functionalities) and debugging (in an emulator or on hardware), both using [Microsoft Visual Studio](http://en.wikipedia.org/wiki/Microsoft_Visual_Studio). It also features a subset of .NET Framework Class Library (about 70 classes with about 420 methods), a [GUI](http://en.wikipedia.org/wiki/GUI)framework loosely based on [Windows Presentation Foundation](http://en.wikipedia.org/wiki/Windows_Presentation_Foundation), and additional libraries specific to embedded applications.
* [Mono](http://en.wikipedia.org/wiki/Mono_(software)) is an implementation of CLI and FCL, and provides additional functionality. It is [dual-licensed](http://en.wikipedia.org/wiki/Dual-license) under [free software](http://en.wikipedia.org/wiki/Free_software) and [proprietary software](http://en.wikipedia.org/wiki/Proprietary_software) licenses. It includes support for ASP.NET, ADO.NET, and [Windows Forms](http://en.wikipedia.org/wiki/Windows_Forms) libraries for a wide range of architectures and operating systems. It also includes C# and VB.NET compilers.
* [Portable.NET](http://en.wikipedia.org/wiki/Portable.NET) (part of [DotGNU](http://en.wikipedia.org/wiki/DotGNU)) provides an implementation of CLI, portions of FCL, and a C# compiler. It supports a variety of CPUs and operating systems.
* Microsoft [Shared Source Common Language Infrastructure](http://en.wikipedia.org/wiki/Shared_Source_Common_Language_Infrastructure) is a [non-free](http://en.wikipedia.org/wiki/Shared_source#Microsoft_Shared_Source_Common_Language_Infrastructure) implementation of CLR. However, the last version only runs on Microsoft Windows XP SP2, and was not updated since 2006, therefore it does not contain all features of version 2.0 of .NET Framework.
* CrossNet is an implementation of CLI and portions of FCL. It is free software using the open source [MIT License](http://en.wikipedia.org/wiki/MIT_License).

**Performance**

The garbage collector, which is integrated into the environment, can introduce unanticipated delays of execution over which the developer has little direct control. "In large applications, the number of objects that the garbage collector needs to deal with can become very large, which means it can take a very long time to visit and rearrange all of them."

.NET Framework provides support for calling [Streaming SIMD Extensions](http://en.wikipedia.org/wiki/Streaming_SIMD_Extensions) (SSE) via [managed code](http://en.wikipedia.org/wiki/Managed_code) from April 2014 in Visual Studio 2013 Update 2. However, [Mono](http://en.wikipedia.org/wiki/Mono_(software)) has provided support for [SIMD](http://en.wikipedia.org/wiki/SIMD) Extensions as of version 2.2 within the Mono.Simd namespace; before. Mono's lead developer [Miguel de Icaza](http://en.wikipedia.org/wiki/Miguel_de_Icaza) has expressed hope that this SIMD support will be adopted by CLR's ECMA standard. Streaming SIMD Extensions have been available in [x86](http://en.wikipedia.org/wiki/X86) CPUs since the introduction of the [Pentium III](http://en.wikipedia.org/wiki/Pentium_III). Some other architectures such as [ARM](http://en.wikipedia.org/wiki/ARM_architecture) and [MIPS](http://en.wikipedia.org/wiki/MIPS_architecture) also have SIMD extensions. In case the CPU lacks support for those extensions, the instructions are simulated in software.

**Security**

[Unobfuscated](http://en.wikipedia.org/wiki/Obfuscated_code) [managed](http://en.wikipedia.org/wiki/Managed_code) [CIL](http://en.wikipedia.org/wiki/Common_Intermediate_Language) bytecode can often be easier to [reverse-engineer](http://en.wikipedia.org/wiki/Reverse-engineering#Reverse_engineering_of_software) than native code. NET [decompiler](http://en.wikipedia.org/wiki/Decompiler) programs enable developers with no reverse-engineering skills to view the source code behind unobfuscated .NET assemblies ([DLL](http://en.wikipedia.org/wiki/Dynamic-link_library)/[EXE](http://en.wikipedia.org/wiki/EXE)). In contrast, applications built with [Visual C++](http://en.wikipedia.org/wiki/Visual_C%2B%2B) are much harder to reverse-engineer and source code is almost never produced successfully, mainly due to compiler optimizations and lack of [reflection](http://en.wikipedia.org/wiki/Reflection_(computer_programming)). One concern is over possible loss of [trade secrets](http://en.wikipedia.org/wiki/Trade_secret) and the bypassing of license control mechanisms. To mitigate this, Microsoft has included [Dotfuscator](http://en.wikipedia.org/wiki/Dotfuscator) Community Edition with [Visual Studio .NET](http://en.wikipedia.org/wiki/Visual_Studio_.NET) since 2002. Third-party obfuscation tools are also available from vendors such as [vmware](http://en.wikipedia.org/wiki/Vmware), [V.i. Labs](http://en.wikipedia.org/wiki/V.i._Labs), [Xenocode](http://en.wikipedia.org/wiki/Xenocode), [Red Gate Software](http://en.wikipedia.org/wiki/Red_Gate_Software). Method-level encryption tools for .NET code are available from vendors such as [SafeNet](http://en.wikipedia.org/wiki/SafeNet)

**C# Language**

The language basics. Data types. Ref and Out specificators. Operations.Namespaces.

**What is C#...**

First, say C# like this: “See Sharp,” just like the music note. When it comes to computer programming language, C# is the best place for a newbie to start. Why? Because it’s simple to learn, well supported and maintained, and extremely versatile.

**...and WHY should I take this C# course?**

You’re a newcomer to coding? Your job is to LEARN THE CODE and that’s what this online C# course is all about. You’ll learn C#’s programming language, its syntax, with clear instruction and detailed videos. And you’ll write code. Lots of code.

Learning the code is Job 1. That’s what this course will teach you. Later comes problem solving, and learning how to create and organize large-scale solutions that are reusable, testable, and of high quality. But all that’s later.

**Want to program? You MUST learn to write code**

The BEST way to become a highly proficient C# programmer is to write code — LOTS of code. And you’ll be writing, writing, and writing in this highly focused course. You’ll get limber with nearly 100 warm up exercises you’ll use to practice and increase your fluency in C# language fundamentals.

You’ll receive and own over 48 lectures, eight quizzes, and more than six and a half hours of focused content — complete instruction on how to understand the solid foundation of C# Programming language.

Get ready to dive right into the code because you’ll actually create real, live applications that run in the console. You’ll get in there and declare and manipulate variables. You’ll write “Flow of Control” statements we call “Loops and Conditions.”

On your path to becoming an expert programmer, will learn how to create methods for code reuse and readability. Cool! And you’ll keep it all nice an neat by learning to organize your code into related classes.

**You’re about to learn the secrets of today’s most powerful and popular programming language**

**Here’s why learning C# is the KEY BUILDING BLOCK of your programming foundation:**

 Java is a programming language many adore but C# is even better. It offers those new to programming many, many advantages. It’s growing in popularity with programmers by leaps and bounds. That’s a trend we’ve seen for many years — C# is becoming more and more popular every year.

 Learning C# programming language makes it easier for you to understand other C-based programming languages. (Just like learning to speak Spanish helps you understand Italian.) And there are many, many very valuable C languages in the C family including C, Java, C++, Objective –C, and more.

 And how about job opportunities? In Cleveland, for example, 98% of C# programmers enjoy gainful employment in their field. C# programmers are in high demand! C# runs on multiple platforms. Windows? Linux? iPhones? Androids? You betcha.

 C# was created and is maintained by Microsoft so you know it gets plenty of support. This software giant is continually adding new and more potent C# features including new versions of C#’s Visual Studio. And the Visual Studio toolset is widely regarded as the #1, A+, five-star, gold-plated, best development environment on Planet Earth.

 C# runs on the .NET framework (say, “dot net”). It’s what pros call a “consistent object-oriented programming environment,” that’s rock-steady for creating web and desktop applications. In fact, no matter what kind of application you’ll build, you’ll find C# will make it steady and stable across all computer platforms.

 Apps? How many and what kind will you develop? No holds barred because C# handles apps for desktop, Windows Store, web apps, mobile apps, iPhone and Android apps, web and data services, and these are just for starters.

 C# makes it easy to deploy and version your code saving you countless sleepless nights. It has a built-in safety net – a “sandbox” – to help you avoid the horrors of the dreaded, dead-on-arrival, pull-the-plug “blue screen of death” when your program crashes the operating system beyond all hope.

**C#**is a [multi-paradigm programming language](https://en.wikipedia.org/wiki/Multi-paradigm_programming_language) encompassing [strong typing](https://en.wikipedia.org/wiki/Strong_typing), [imperative](https://en.wikipedia.org/wiki/Imperative_programming), [declarative](https://en.wikipedia.org/wiki/Declarative_programming), [functional](https://en.wikipedia.org/wiki/Functional_programming), [generic](https://en.wikipedia.org/wiki/Generic_programming), [object-oriented](https://en.wikipedia.org/wiki/Object-oriented_programming) ([class](https://en.wikipedia.org/wiki/Class_(computer_science))-based), and[component-oriented](https://en.wikipedia.org/wiki/Component-based_software_engineering) programming disciplines. It was developed by [Microsoft](https://en.wikipedia.org/wiki/Microsoft) within its [.NET](https://en.wikipedia.org/wiki/.NET_Framework) initiative and later approved as a standard by [Ecma](https://en.wikipedia.org/wiki/Ecma_International) (ECMA-334) and [ISO](https://en.wikipedia.org/wiki/International_Organization_for_Standardization) (ISO/IEC 23270:2006). C# is one of the programming languages designed for the [Common Language Infrastructure](https://en.wikipedia.org/wiki/Common_Language_Infrastructure).

C# is intended to be a simple, modern, general-purpose, object-oriented programming language. Its development team is led by [Anders Hejlsberg](https://en.wikipedia.org/wiki/Anders_Hejlsberg). The most recent version is C# 5.0, which was released on August 15, 2012.

**Design goals**

The ECMA standard lists these design goals for C#

* The C# language is intended to be a simple, modern, general-purpose, [object-oriented programming](https://en.wikipedia.org/wiki/Object-oriented_programming) language.
* The language, and implementations thereof, should provide support for software engineering principles such as [strong type](https://en.wikipedia.org/wiki/Strong_type) checking, array [bounds checking](https://en.wikipedia.org/wiki/Bounds_checking), detection of attempts to use [uninitialized variables](https://en.wikipedia.org/wiki/Uninitialized_variable), and automatic [garbage collection](https://en.wikipedia.org/wiki/Garbage_collection_(computer_science)). Software robustness, durability, and programmer productivity are important.
* The language is intended for use in developing [software components](https://en.wikipedia.org/wiki/Software_components) suitable for deployment in distributed environments.
* Source code portability is very important, as is programmer portability, especially for those programmers already familiar with [C](https://en.wikipedia.org/wiki/C_(programming_language)) and [C++](https://en.wikipedia.org/wiki/C%2B%2B_(programming_language)).
* Support for [internationalization](https://en.wikipedia.org/wiki/Internationalization_and_localization) is very important.
* C# is intended to be suitable for writing applications for both hosted and [embedded systems](https://en.wikipedia.org/wiki/Embedded_system), ranging from the very large that use sophisticated [operating systems](https://en.wikipedia.org/wiki/Operating_system), down to the very small having dedicated functions.
* Although C# applications are intended to be economical with regard to memory and [processing power](https://en.wikipedia.org/wiki/Processing_power) requirements, the language was not intended to compete directly on performance and size with C or assembly language.

The name "C sharp" was inspired by musical notation where a [sharp](https://en.wikipedia.org/wiki/Sharp_(music)) indicates that the written note should be made a [semitone](https://en.wikipedia.org/wiki/Semitone) higher in [pitch](https://en.wikipedia.org/wiki/Pitch_(music)). This is similar to the language name of [C++](https://en.wikipedia.org/wiki/C%2B%2B), where "++" indicates that a variable should be incremented by 1. The sharp symbol also resembles a [ligature](https://en.wikipedia.org/wiki/Typographic_ligature) of four "+" symbols (in a two-by-two grid), further implying that the language is an increment of C++.

Due to technical limitations of display (standard fonts, browsers, etc.) and the fact that the sharp symbol is not present on the standard keyboard, the [number sign](https://en.wikipedia.org/wiki/Number_sign) was chosen to represent the sharp symbol in the written name of the programming language. This convention is reflected in the ECMA-334 C# Language Specification. However, when it is practical to do so (for example, in advertising or in box art), Microsoft uses the intended musical symbol.

The "sharp" suffix has been used by a number of other .NET languages that are variants of existing languages, including [J#](https://en.wikipedia.org/wiki/J_Sharp) (a .NET language also designed by Microsoft that is derived from Java 1.1), [A#](https://en.wikipedia.org/wiki/A_Sharp_(.NET)) (from [Ada](https://en.wikipedia.org/wiki/Ada_(programming_language))), and the [functional programming](https://en.wikipedia.org/wiki/Functional_programming) language [F#](https://en.wikipedia.org/wiki/F_Sharp_(programming_language)). The original implementation of [Eiffel for .NET](https://en.wikipedia.org/wiki/EiffelStudio) was called Eiffel#, a name retired since the full [Eiffel](https://en.wikipedia.org/wiki/Eiffel_(programming_language)) language is now supported. The suffix has also been used for [libraries](https://en.wikipedia.org/wiki/Library_(computing)), such as [Gtk#](https://en.wikipedia.org/wiki/Gtk_Sharp) (a .NET[wrapper](https://en.wikipedia.org/wiki/Wrapper_pattern) for [GTK+](https://en.wikipedia.org/wiki/GTK%2B) and other [GNOME](https://en.wikipedia.org/wiki/GNOME) libraries) and [Cocoa#](https://en.wikipedia.org/wiki/Cocoa_Sharp) (a wrapper for [Cocoa](https://en.wikipedia.org/wiki/Cocoa_(API))).

**History**

During the development of the .NET Framework, the [class libraries](https://en.wikipedia.org/wiki/Base_Class_Library) were originally written using a [managed code](https://en.wikipedia.org/wiki/Managed_code) compiler system called *Simple Managed C* (SMC). In January 1999, [Anders Hejlsberg](https://en.wikipedia.org/wiki/Anders_Hejlsberg) formed a team to build a new language at the time called Cool, which stood for "C-like Object Oriented Language". Microsoft had considered keeping the name "Cool" as the final name of the language, but chose not to do so for trademark reasons. By the time the .NET project was publicly announced at the July 2000 [Professional Developers Conference](https://en.wikipedia.org/wiki/Professional_Developers_Conference), the language had been renamed C#, and the class libraries and [ASP.NET](https://en.wikipedia.org/wiki/ASP.NET) runtime had been ported to C#.

C#'s principal designer and lead architect at Microsoft is [Anders Hejlsberg](https://en.wikipedia.org/wiki/Anders_Hejlsberg), who was previously involved with the design of [Turbo Pascal](https://en.wikipedia.org/wiki/Turbo_Pascal), [Embarcadero Delphi](https://en.wikipedia.org/wiki/Embarcadero_Delphi) (formerly CodeGear Delphi, Inprise Delphi and Borland Delphi), and [Visual J++](https://en.wikipedia.org/wiki/Visual_J%2B%2B). In interviews and technical papers he has stated that flaws in most major programming languages (e.g. [C++](https://en.wikipedia.org/wiki/C%2B%2B), [Java](https://en.wikipedia.org/wiki/Java_(programming_language)), [Delphi](https://en.wikipedia.org/wiki/Embarcadero_Delphi), and [Smalltalk](https://en.wikipedia.org/wiki/Smalltalk)) drove the fundamentals of the [Common Language Runtime](https://en.wikipedia.org/wiki/Common_Language_Runtime) (CLR), which, in turn, drove the design of the C# language itself.

[James Gosling](https://en.wikipedia.org/wiki/James_Gosling), who created the [Java](https://en.wikipedia.org/wiki/Java_(programming_language)) programming language in 1994, and [Bill Joy](https://en.wikipedia.org/wiki/Bill_Joy), a co-founder of Sun Microsystems, the originator of Java, called C# an "imitation" of Java; Gosling further said that "[C# is] sort of Java with reliability, productivity and security deleted." Klaus Kreft and Angelika Langer (authors of a C++ streams book) stated in a blog post that "Java and C# are almost identical programming languages. Boring repetition that lacks innovation, Hardly anybody will claim that Java or C# are revolutionary programming languages that changed the way we write programs," and "C# borrowed a lot from Java - and vice versa. Now that C# supports [boxing](https://en.wikipedia.org/wiki/Boxing_(Computer_Science)) and unboxing, we'll have a very similar feature in Java." In July 2000, Anders Hejlsberg said that C# is "not a Java clone" and is "much closer to C++" in its design.

Since the release of C# 2.0 in November 2005, the C# and Java languages have evolved on increasingly divergent trajectories, becoming somewhat less similar. One of the first major departures came with the addition of [generics](https://en.wikipedia.org/wiki/Generic_programming) to both languages, with vastly different implementations. C# makes use of [reification](https://en.wikipedia.org/wiki/Reification_(computer_science)) to provide "first-class" generic objects that can be used like any other class, with code generation performed at class-load time. By contrast, Java's generics are essentially a language syntax feature, and they do not affect the generated byte code, because the compiler performs [type erasure](https://en.wikipedia.org/wiki/Type_erasure) on the generic type information after it has verified its correctness.

Furthermore, C# has added several major features to accommodate functional-style programming, culminating in the [LINQ](https://en.wikipedia.org/wiki/Language_Integrated_Query) extensions released with C# 3.0 and its supporting framework of [lambda expressions](https://en.wikipedia.org/wiki/Lambda_expressions), [extension methods](https://en.wikipedia.org/wiki/Extension_method), and anonymous types. These features enable C# programmers to use functional programming techniques, such as [closures](https://en.wikipedia.org/wiki/Closure_(computer_science)), when it is advantageous to their application. The LINQ extensions and the functional imports help developers reduce the amount of "boilerplate" code that is included in common tasks like querying a database, parsing an xml file, or searching through a data structure, shifting the emphasis onto the actual program logic to help improve readability and maintainability.

C# used to have a [mascot](https://en.wikipedia.org/wiki/Mascot) called Andy (named after Anders Hejlsberg). It was retired on January 29, 2004.

C# was originally submitted for review to the ISO subcommittee JTC 1/SC 22 under ISO/IEC 23270:2003, which is now withdrawn. It was then approved under ISO/IEC 23270:2006.

**Syntax**

The core syntax of C# language is similar to that of other C-style languages such as C, C++ and Java. In particular:

* Semicolons are used to denote the end of a statement.
* [Curly brackets](https://en.wikipedia.org/wiki/Curly_brackets) are used to group statements. Statements are commonly grouped into methods (functions), methods into classes, and classes into [namespaces](https://en.wikipedia.org/wiki/Namespaces).
* Variables are assigned using an [equals sign](https://en.wikipedia.org/wiki/Equals_sign), but compared using [two consecutive equals signs](https://en.wikipedia.org/wiki/%3D%3D).
* [Square brackets](https://en.wikipedia.org/wiki/Square_brackets) are used with [arrays](https://en.wikipedia.org/wiki/Array_data_structure), both to declare them and to get a value at a given index in one of them.

**Distinguishing features**

By design, C# is the programming language that most directly reflects the underlying [Common Language Infrastructure](https://en.wikipedia.org/wiki/Common_Language_Infrastructure) (CLI). Most of its intrinsic types correspond to value-types implemented by the CLI framework. However, the language specification does not state the code generation requirements of the compiler: that is, it does not state that a C# compiler must target a Common Language Runtime, or generate [Common Intermediate Language](https://en.wikipedia.org/wiki/Common_Intermediate_Language) (CIL), or generate any other specific format. Theoretically, a C# compiler could generate machine code like traditional compilers of C++ or [Fortran](https://en.wikipedia.org/wiki/Fortran). Some notable features of C# that distinguish it from C and C++ (and Java, where noted) are:

* C# supports strongly typed implicit variable declarations with the keyword **var**, and implicitly typed arrays with the keyword new[] followed by a collection initializer.
* Meta programming via C# attributes is part of the language. Many of these attributes duplicate the functionality of GCC's and VisualC++'s platform-dependent preprocessor directives.
* Like C++, and unlike Java, C# programmers must use the keyword **virtual** to allow methods to be overridden by subclasses.
* *Extension methods* in C# allow programmers to use static methods as if they were methods from a class's method table, allowing programmers to add methods to an object that they feel should exist on that object and its derivatives.
* The type **dynamic** allows for run-time method binding, allowing for JavaScript-like method calls and run-time object composition.
* C# has support for strongly-typed function pointers via the keyword **delegate**.
* Like the Qt framework's pseudo-C++ *signal* and *slot*, C# has semantics specifically surrounding publish-subscribe style events, though C# uses delegates to do so.
* C# offers Java-like **synchronized** method calls, via the attribute [MethodImpl(MethodImplOptions.Synchronized)], and has support for [mutually-exclusive locks](https://en.wikipedia.org/wiki/Mutual_exclusion) via the keyword **lock**.
* The C# language does not allow for global variables or functions. All methods and members must be declared within classes. Static members of public classes can substitute for global variables and functions.
* Local variables cannot shadow variables of the enclosing block, unlike C and C++.
* A C# **namespace** provides the same level of code isolation as a Java **package** or a C++ namespace, with very similar rules and features to a **package**.
* C# supports a strict [Boolean data type](https://en.wikipedia.org/wiki/Boolean_data_type), **bool**. Statements that take conditions, such as **while** and **if**, require an expression of a type that implements the **true** operator, such as the Boolean type. While C++ also has a Boolean type, it can be freely converted to and from integers, and expressions such as **if**(a) require only that a is convertible to bool, allowing a to be an int, or a pointer. C# disallows this "integer meaning true or false" approach, on the grounds that forcing programmers to use expressions that return exactly **bool** can prevent certain types of programming mistakes common in C or C++ such as **if** (a = b) (use of assignment = instead of equality ==).
* In C#, memory address pointers can only be used within blocks specifically marked as *unsafe*, and programs with unsafe code need appropriate permissions to run. Most object access is done through safe object references, which always either point to a "live" object or have the well-defined [null](https://en.wikipedia.org/wiki/Nullable_type) value; it is impossible to obtain a reference to a "dead" object (one that has been garbage collected), or to a random block of memory. An unsafe pointer can point to an instance of a value-type, array, string, or a block of memory allocated on a stack. Code that is not marked as unsafe can still store and manipulate pointers through the System.IntPtr type, but it cannot dereference them.
* Managed memory cannot be explicitly freed; instead, it is automatically garbage collected. Garbage collection addresses the problem of [memory leaks](https://en.wikipedia.org/wiki/Memory_leak) by freeing the programmer of responsibility for releasing memory that is no longer needed.
* In addition to the **try**...**catch** construct to [handle exceptions](https://en.wikipedia.org/wiki/Exception_handling), C# has a **try**...**finally** construct to guarantee execution of the code in the **finally** block, whether an exception occurs or not. This is present in other programming languages like [Java](https://en.wikipedia.org/wiki/Java_(programming_language)) and [Python](https://en.wikipedia.org/wiki/Python_(programming_language)).
* C#, supports connections to various types and providers of databases, from [SQLite](https://en.wikipedia.org/wiki/SQLite), [MySQL](https://en.wikipedia.org/wiki/MySQL), [Microsoft SQL Server](https://en.wikipedia.org/wiki/Microsoft_SQL_Server) (MSSQL), [Oracle](https://en.wikipedia.org/wiki/Oracle_Database), etc. You might need to import 3rd party libraries to use some of the database, except MSSQL which is supported by default.
* [Multiple inheritance](https://en.wikipedia.org/wiki/Multiple_inheritance) is not supported, although a class can implement any number of interfaces. This was a design decision by the language's lead architect to avoid complication and simplify architectural requirements throughout CLI. When implementing multiple interfaces that contain a method with the same signature, C# allows the programmer to implement each method depending on which interface that method is being called through, or, like Java, allows the programmer to implement the method once and have that be the single invocation on a call through any of the class's interfaces.
* C#, unlike Java, supports [operator overloading](https://en.wikipedia.org/wiki/Operator_overloading). Only the most commonly overloaded operators in C++ may be overloaded in C#.
* C# is more [type safe](https://en.wikipedia.org/wiki/Type_safety) than C++. The only implicit conversions by default are those that are considered safe, such as widening of integers. This is enforced at compile-time, during [JIT](https://en.wikipedia.org/wiki/Just-in-time_compilation), and, in some cases, at runtime. No implicit conversions occur between Booleans and integers, nor between enumeration members and integers (except for literal 0, which can be implicitly converted to any enumerated type). Any user-defined conversion must be explicitly marked as explicit or implicit, unlike C++ [copy constructors](https://en.wikipedia.org/wiki/Copy_constructor) and conversion operators, which are both implicit by default.
* C# has explicit support for [covariance and contravariance](https://en.wikipedia.org/wiki/Covariance_and_contravariance_(computer_science)) in generic types, unlike C++ which has some degree of support for contravariance simply through the semantics of return types on virtual methods.
* [Enumeration](https://en.wikipedia.org/wiki/Enumerated_type) members are placed in their own [scope](https://en.wikipedia.org/wiki/Scope_(programming)).
* C# provides [properties](https://en.wikipedia.org/wiki/Property_(programming)) as [syntactic sugar](https://en.wikipedia.org/wiki/Syntactic_sugar) for a common pattern in which a pair of methods, [accessor (getter) and mutator (setter)](https://en.wikipedia.org/wiki/Mutator_method) encapsulate operations on a single [attribute](https://en.wikipedia.org/wiki/Attribute_(computing)) of a class. No redundant method signatures for the getter/setter implementations need be written, and the property may be accessed using attribute syntax rather than more verbose method calls.
* [Checked exceptions](https://en.wikipedia.org/wiki/Checked_exceptions) are not present in C# (in contrast to Java). This has been a conscious decision based on the issues of scalability and versionability.
* Though primarily an imperative language, C# 2.0 offered limited support for functional programming through [first-class functions](https://en.wikipedia.org/wiki/First-class_functions) and closures in the form of anonymous delegates. C# 3.0 expanded support for functional programming with the introduction of a light weight syntax for lambda expressions, extension methods (an affordance for modules), and a [list comprehension](https://en.wikipedia.org/wiki/List_comprehension) syntax in the form of a "query comprehension" language.

**Common type system**

C# has a *unified type system*. This unified type system is called [Common Type System](https://en.wikipedia.org/wiki/Common_Type_System) (CTS).

A unified type system implies that all types, including primitives such as integers, are subclasses of the System.**Object** class. For example, every type inherits a ToString() method.

**Categories of data types**

CTS separates data types into two categories:

1. Reference types
2. Value types

Instances of value types do not have referential identity nor referential comparison semantics - equality and inequality comparisons for value types compare the actual data values within the instances, unless the corresponding operators are overloaded. Value types are derived from System.ValueType, always have a default value, and can always be created and copied. Some other limitations on value types are that they cannot derive from each other (but can implement interfaces) and cannot have an explicit default (parameterless) constructor. Examples of value types are all primitive types, such as **int** (a signed 32-bit integer), **float** (a 32-bit IEEE floating-point number), **char** (a 16-bit Unicode code unit), andSystem.DateTime (identifies a specific point in time with nanosecond precision). Other examples are **enum** (enumerations) and **struct** (user defined structures).

In contrast, reference types have the notion of referential identity - each instance of a reference type is inherently distinct from every other instance, even if the data within both instances is the same. This is reflected in default equality and inequality comparisons for reference types, which test for referential rather than structural equality, unless the corresponding operators are overloaded (such as the case for System.**String**). In general, it is not always possible to create an instance of a reference type, nor to copy an existing instance, or perform a value comparison on two existing instances, though specific reference types can provide such services by exposing a public constructor or implementing a corresponding interface (such as ICloneable or IComparable). Examples of reference types are **object** (the ultimate base class for all other C# classes), System.**String** (a string of Unicode characters), and System.Array (a base class for all C# arrays).

Both type categories are extensible with user-defined types.

**Boxing and unboxing**

*Boxing* is the operation of converting a value-type object into a value of a corresponding reference type. Boxing in C# is implicit.

*Unboxing* is the operation of converting a value of a reference type (previously boxed) into a value of a value type.[[41]](https://en.wikipedia.org/w/index.php?title=C_Sharp_(programming_language)&printable=yes#cite_note-insidecsharpp2ch4-45) Unboxing in C# requires an explicit [type cast](https://en.wikipedia.org/wiki/Type_conversion). A boxed object of type T can only be unboxed to a T (or a nullable T).

Example:

**int** foo1 = 42; *// Value type.*

**object** bar = foo1; *// foo is boxed to bar.*

**int** foo2 = (**int**)bar; *// Unboxed back to value type.*

**Generics**

[Generics](https://en.wikipedia.org/wiki/Generic_programming) were added to version 2.0 of the C# language. Generics use type parameters, which make it possible to design classes and methods that do not specify the type used until the class or method is instantiated. The main advantage is that one can use generic type parameters to create classes and methods that can be used without incurring the cost of runtime casts or boxing operations, as shown here:

*// Declare the generic class.*

**public** **class** GenericList<T>

{

**void** **Add**(T input) { }

}

**class** TestGenericList

{

**private** **class** ExampleClass { }

**static** **void** Main()

{

*// Declare a list of type int.*

GenericList<**int**> list1 = new GenericList<**int**>();

*// Declare a list of type string.*

GenericList<**string**> list2 = new GenericList<**string**>();

*// Declare a list of type ExampleClass.*

GenericList<ExampleClass> list3 = new GenericList<ExampleClass>();

}

}

When compared with [C++ templates](https://en.wikipedia.org/wiki/C%2B%2B_templates), C# generics can provide enhanced safety, but also have somewhat limited capabilities. For example, it is not possible to call arithmetic operators on a C# generic type.

**Preprocessor**

C# features "preprocessor directives" (though it does not have an actual preprocessor) based on the [C preprocessor](https://en.wikipedia.org/wiki/C_preprocessor) that allow programmers to define [symbols](https://en.wikipedia.org/wiki/Symbol_(programming)), but not macros. Conditionals such as #if, #endif, and #else are also provided. Directives such as #region give hints to editors for [code folding](https://en.wikipedia.org/wiki/Code_folding).

**public** **class** Foo

{

#region Constructors

**public** Foo() {}

**public** Foo(**int** firstParam) {}

#endregion

#region Procedures

**public** **void** IntBar(**int** firstParam) {}

**public** **void** StrBar(**string** firstParam) {}

**public** **void** BoolBar(**bool** firstParam) {}

#endregion

}

**Code comments**

C# utilizes a double slash (*//*) to indicate the rest of the line is a [comment](https://en.wikipedia.org/wiki/Comment_(computer_programming)). This is inherited from [C++](https://en.wikipedia.org/wiki/C%2B%2B).

**public** **class** Foo

{

*// a comment*

**public** **static** **void** Bar(**int** firstParam) {} *// also a comment*

}

Multi-line comments can start with slash-asterisk (*/\**) and end asterisk-slash (\*/). This is inherited from standard [C](https://en.wikipedia.org/wiki/C_(programming_language)).

**public** **class** Foo

{

*/\* A Multi-Line*

*comment \*/*

**public** **static** **void** Bar(**int** firstParam) {}

}

**XML documentation system**

C#'s documentation system is similar to Java's [Javadoc](https://en.wikipedia.org/wiki/Javadoc), but based on [XML](https://en.wikipedia.org/wiki/Extensible_Markup_Language). Two methods of documentation are currently supported by the C# [compiler](https://en.wikipedia.org/wiki/Compiler).

Single-line documentation comments, such as those commonly found in [Visual Studio](https://en.wikipedia.org/wiki/Microsoft_Visual_Studio) generated code, are indicated on a line beginning with *///*.

**public** **class** Foo

{

*/// <summary>A summary of the method.</summary>*

*/// <param name="firstParam">A description of the parameter.</param>*

*/// <remarks>Remarks about the method.</remarks>*

**public** **static** **void** Bar(**int** firstParam) {}

}

Multi-line documentation comments, while defined in the version 1.0 language specification, were not supported until the [.NET](https://en.wikipedia.org/wiki/.NET_Framework) 1.1 release. These comments start with slash-asterisk-asterisk (*/\*\**) and end asterisk-slash (\*/).

**public** **class** Foo

{

*/\*\* <summary>A summary of the method.</summary>*

*\* <param name="firstParam">A description of the parameter.</param>*

*\* <remarks>Remarks about the method.</remarks> \*/*

**public** **static** **void** Bar(**int** firstParam) {}

}

Note there are some stringent criteria regarding white space and XML documentation when using the slash/asterisk/asterisk (*/\*\**) technique.

This code block:

*/\*\**

*\* <summary>*

*\* A summary of the method.</summary>\*/*

produces a different XML comment from this code block:

*/\*\**

*\* <summary>*

*A summary of the method.</summary>\*/*

Syntax for documentation comments and their [XML](https://en.wikipedia.org/wiki/XML) markup is defined in a non-normative annex of the [ECMA](https://en.wikipedia.org/wiki/Ecma_International) C# standard. The same standard also defines rules for processing of such comments, and their transformation to a plain [XML](https://en.wikipedia.org/wiki/XML)document with precise rules for mapping of [CLI](https://en.wikipedia.org/wiki/Common_Language_Infrastructure) identifiers to their related documentation elements. This allows any C# [IDE](https://en.wikipedia.org/wiki/Integrated_development_environment) or other development tool to find documentation for any symbol in the code in a certain well-defined way.

**Libraries**

The C# specification details a minimum set of types and class libraries that the compiler expects to have available. In practice, C# is most often used with some implementation of the [Common Language Infrastructure](https://en.wikipedia.org/wiki/Common_Language_Infrastructure) (CLI), which is standardized as ECMA-335 *Common Language Infrastructure (CLI)*.

**"Hello, world" example**

The following is a very simple C# program, a version of the classic "[Hello world](https://en.wikipedia.org/wiki/Hello_world)" example:

**using** System;

**class** Program

{

**static** **void** Main()

{

Console.WriteLine("Hello, world!");

}

}

The effect is to write the following text to the output console:

Hello, world!

Each line has a purpose:

**using** System;

The above line of code tells the compiler to use System as a candidate prefix for types used in the source code. In this case, when the compiler sees use of the Console type later in the source code, it tries to find a type named Console, first in the current assembly, followed by all referenced assemblies. In this case the compiler fails to find such a type, since the name of the type is actually System.Console. The compiler then attempts to find a type named System.Console by using the Systemprefix from the **using** statement, and this time it succeeds. The **using** statement allows the programmer to state all candidate prefixes to use during compilation instead of always using full type names.

**class** Program

Above is a [class](https://en.wikipedia.org/wiki/Class_(computer_science)) definition. Everything between the following pair of braces describes Program.

**static** **void** Main()

This declares the class member method where the program begins execution. The .NET runtime calls the Main method. (Note: Main may also be called from elsewhere, like any other method, e.g. from another method of Program.) The [**static**keyword](https://en.wikipedia.org/wiki/Static_(keyword)) makes the method accessible without an instance of Program. Each console application's Main entry point must be declared **static**. Otherwise, the program would require an instance, but any instance would require a program. To avoid that irresolvable [circular dependency](https://en.wikipedia.org/wiki/Circular_dependency), C# compilers processing [console applications](https://en.wikipedia.org/wiki/Console_application) (like that above) report an error, if there is no **static** Main method. The **void** keyword declares that Main has no [return value](https://en.wikipedia.org/wiki/Return_value).

Console.WriteLine("Hello, world!");

This line writes the output. Console is a static class in the System namespace. It provides an interface to the standard input, output, and error streams for console applications. The program calls the Console method WriteLine, which displays on the console a line with the argument, the string "Hello world!".

A [GUI](https://en.wikipedia.org/wiki/GUI) example:

**using** System.Windows.Forms;

**class** Program

{

**static** **void** Main()

{

MessageBox.Show("Hello, world!");

}

}

This example is similar to the previous example, except that it generates a [dialog box](https://en.wikipedia.org/wiki/Dialog_box) that contains the message "Hello, world!" instead of writing it to the console.

**Standardization and licensing**

In August 2000, Microsoft Corporation, Hewlett-Packard and Intel Corporation co-sponsored the submission of specifications for C# as well as the Common Language Infrastructure (CLI) to the standards organization [Ecma International](https://en.wikipedia.org/wiki/Ecma_International). In December 2001, ECMA released ECMA-334 *C# Language Specification*. C# became an [ISO](https://en.wikipedia.org/wiki/International_Organization_for_Standardization) standard in 2003 (ISO/IEC 23270:2003 - *Information technology — Programming languages — C#*). ECMA had previously adopted equivalent specifications as the 2nd edition of C#, in December 2002.

In June 2005, ECMA approved edition 3 of the C# specification, and updated ECMA-334. Additions included partial classes, anonymous methods, nullable types, and [generics](https://en.wikipedia.org/wiki/Generic_programming) (somewhat similar to C++ [templates](https://en.wikipedia.org/wiki/Template_(programming))).

In July 2005, ECMA submitted the standards and related TRs to ISO/IEC JTC 1 via the latter's Fast-Track process. This process usually takes 6–9 months.

The C# language definition and the [CLI](https://en.wikipedia.org/wiki/Common_Language_Infrastructure) are standardized under [ISO](https://en.wikipedia.org/wiki/International_Organization_for_Standardization) and [Ecma](https://en.wikipedia.org/wiki/Ecma) standards that provide [reasonable and non-discriminatory licensing](https://en.wikipedia.org/wiki/Reasonable_and_Non_Discriminatory_Licensing) protection from patent claims.

Microsoft has agreed not to sue open source developers for violating patents in non-profit projects for the part of the framework that is covered by the OSP. Microsoft has also agreed not to enforce patents relating to [Novell](https://en.wikipedia.org/wiki/Novell) products against Novell's paying customers with the exception of a list of products that do not explicitly mention C#, .NET or Novell's implementation of .NET ([The Mono Project](https://en.wikipedia.org/wiki/Mono_(software))). However, Novell maintains that Mono does not infringe any Microsoft patents. Microsoft has also made a specific agreement not to enforce patent rights related to the [Moonlight browser plugin](https://en.wikipedia.org/wiki/Moonlight_(runtime)), which depends on Mono, provided it is obtained through [Novell](https://en.wikipedia.org/wiki/Novell).[[53]](https://en.wikipedia.org/w/index.php?title=C_Sharp_(programming_language)&printable=yes#cite_note-MsCovenant-57)

**Implementations**

The reference C# compiler is [Microsoft Visual C#](https://en.wikipedia.org/wiki/Microsoft_Visual_C_Sharp), which is closed-source.

Microsoft is leading the development of a new [open source](https://en.wikipedia.org/wiki/Open_source) C# compiler and set of tools, previously codenamed "[Roslyn](https://en.wikipedia.org/wiki/Microsoft_Roslyn)". The compiler, which is entirely written in [managed code](https://en.wikipedia.org/wiki/Managed_code) (C#), has been opened up and functionality surfaced as APIs. It is thus enabling developers to create refactoring and diagnostics tools.

Other C# compilers exist, often including an implementation of the [Common Language Infrastructure](https://en.wikipedia.org/wiki/Common_Language_Infrastructure) and the .NET class libraries up to .NET 2.0:

* The [Mono](https://en.wikipedia.org/wiki/Mono_(software)) project provides an [open source](https://en.wikipedia.org/wiki/Open_source) C# compiler, a complete open source implementation of the Common Language Infrastructure including the required framework libraries as they appear in the ECMA specification, and a nearly complete implementation of the Microsoft proprietary .NET class libraries up to .NET 3.5. As of Mono 2.6, no plans exist to implement [WPF](https://en.wikipedia.org/wiki/Windows_Presentation_Foundation); [WF](https://en.wikipedia.org/wiki/Windows_Workflow_Foundation) is planned for a later release; and there are only partial implementations of [LINQ to SQL](https://en.wikipedia.org/wiki/Language_Integrated_Query#LINQ_to_SQL) and[WCF](https://en.wikipedia.org/wiki/Windows_Communication_Foundation).
* The [DotGNU](https://en.wikipedia.org/wiki/DotGNU) project also provides an open source C# compiler, a nearly complete implementation of the Common Language Infrastructure including the required framework libraries as they appear in the ECMA specification, and subset of some of the remaining Microsoft proprietary .NET class libraries up to .NET 2.0 (those not documented or included in the ECMA specification, but included in Microsoft's standard .NET Framework distribution).
* Microsoft's Rotor project (currently called [Shared Source Common Language Infrastructure](https://en.wikipedia.org/wiki/Shared_Source_Common_Language_Infrastructure)) (licensed for educational and research use only) provides a [shared source](https://en.wikipedia.org/wiki/Shared_source) implementation of the CLR runtime and a C# compiler, and a subset of the required [Common Language Infrastructure](https://en.wikipedia.org/wiki/Common_Language_Infrastructure) framework libraries in the ECMA specification (up to C# 2.0, and supported on Windows XP only).\

Questions

1. What are main two data types in C# ?
2. What expressions and operations in C# do we have?
3. How C# provide exception handling?
4. Describe common body parts of class?
5. Why do we need properties in classes?
6. What is .NET Framework?
7. What are The main layers of .NET Framework?
8. What happens with code when programmer compiles it?
9. Which stage goes the application during runtime?
10. .NET framework platform structure.