

Imagen que contiene firmar, parada, reloj

Descripción generada automáticamente

**C#**

**Indice**

1. **Historia de C#**
2. **Anders Hejlsberg**
3. **Clasificación**
4. **Versiones de C#**
5. **Principales caracteristicas**
6. **Curva de aprendizaje**
7. **Ecosistema/ Soporte/ Cominudad**
8. **Entorno de Desarrollo que soporta**
9. **Nivel de demanda de C#**

**10.Fuentes consultadas**

1. **HISTORIA C#**

Imagen que contiene dibujo, plato

Descripción generada automáticamenteC#, desarrollado por Anders Hejlsberg, nace para ser rival de Java, ya que Sun (que luego compró Oracle) no quería que Microsoft hiciera cambios en Java, por lo que Microsoft decidió crear su propio lenguaje.

**En enero de 1999**, **Anders Hejlsberg** formó un equipo para construir un nuevo lenguaje en ese momento llamado Cool, que en inglés significaba «lenguaje orientado a objetos tipo C». Microsoft había considerado mantener el nombre «Cool» como el nombre final del lenguaje, pero decidió no hacerlo por razones de marca registrada.

Por su cercanía y espíritu de competencia de C# con Java recibió fuertes críticas de James James Gosling, quien creó el lenguaje de programación Java en 1994; y Bill Joy, cofundador de Sun Microsystems; quienes lo llamaron una «imitación».

Pese a todo esto, sí han logrado tomar caminos propios. **Con el lanzamiento de C # 2.0 en noviembre de 2005**, los lenguajes C # y Java han evolucionado en trayectorias cada vez más divergentes, volviéndose menos similares.

**Una de las primeras vino con la reificación de C# para proporcionar objetos genéricos de «primera clase» que se pueden usar como cualquier otra clase, con la generación de código realizada en tiempo de carga de clase**. Por el contrario, los genéricos de Java son esencialmente una característica de sintaxis de lenguaje, y no afectan el código de bytes generado porque el compilador realiza el borrado de tipo en la información de tipo genérico después de haber verificado su corrección.

Finalmente, C # es un lenguaje compilado, lo que significa que el código almacenado en un servidor público está en forma binaria. Si el servidor es pirateado, el hacker no tendrá acceso automático al código fuente. Con otros lenguajes comunes como PHP, el atacante obtiene acceso al código fuente, que luego podría darle acceso a las contraseñas de la base de datos. Con C #, el hacker debe descompilar o «crackear» su software antes de poder ver los componentes críticos.

1. **FABRICANTE Y LICENCIAS**

**Anders Hejlsberg** nació en 1960 (algunas fuentes señalan que fue en 1961, pero durante el TechEd 2006 en Barcelona, el propio Hejlsberg confirmó que nació en diciembre de 1960). Es un destacado ingeniero de software danés que co-diseñó varios lenguajes de programación e instrumentos de desarrollo populares y comercialmente acertados. Actualmente trabaja para Microsoft, donde es el arquitecto jefe del lenguaje de programación TypeScript.

Hejlsberg se convirtió en uno de los objetivos principales de Microsoft y, tras una sucesión de ofertas, Borland no pudo igualar la oferta hecha por Microsoft. En 1996 Hejlsberg abandonó Borland y se unió a Microsoft. **Uno de sus primeros logros fue el lenguaje de programación J++**. También se convirtió en un Ingeniero distinguido y Técnico de Microsoft. **Desde 2000, ha sido el arquitecto principal del equipo que desarrolla el lenguaje de programación C#**. En 2012 Hejsberg anunció su nuevo proyecto TypeScript

1. **CLASIFICACIÓN**

C# es un lenguaje de alto nivel, y pertenece a la tercera genreacion.

Es un tipo de lenguaje compilado y soporta mas de un paradigma,

Multiparadigma: estructurado, imperativo, orientado a objetos, funcional …(no los hemos visto en clase).

1. **VERSIONES DE C#**

C# versión 1.0

Si echa la vista atrás, la versión 1.0 de C#, publicada con Visual Studio .NET 2002, se parecía mucho a Java. Como [parte de sus objetivos de diseño indicados para ECMA](https://feeldotneteasy.blogspot.com/2011/01/c-design-goals.html), intentaba ser un "lenguaje orientado a objetos que fuera sencillo, moderno y para fines generales". En aquel momento, parecerse a Java significaba que conseguía esos primeros objetivos de diseño.

Pero si volvemos a echarle un vistazo a C# 1.0 ahora, no lo verá tan claro. Carecía de capacidades asincrónicas integradas y de algunas funcionalidades útiles de genéricos que se dan por sentado. De hecho, carecía por completo de genéricos. ¿Y [LINQ](https://docs.microsoft.com/es-es/dotnet/csharp/linq/)? Aún no estaba disponible. Esas características tardarían unos años más en agregarse.

C# 1.0 parecía estar privado de características, en comparación con la actualidad. Lo normal era tener que escribir código detallado. Pero aun así, hay que empezar por algo. C# 1.0 era una alternativa viable a Java en la plataforma Windows.

Las principales características de C# 1.0 incluían lo siguiente:

* [Clases](https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/classes-and-structs/classes)
* [Structs](https://docs.microsoft.com/es-es/dotnet/csharp/language-reference/builtin-types/struct)
* [Interfaces](https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/interfaces/)
* [Eventos](https://docs.microsoft.com/es-es/dotnet/csharp/events-overview)
* [Propiedades](https://docs.microsoft.com/es-es/dotnet/csharp/properties)
* [Delegados](https://docs.microsoft.com/es-es/dotnet/csharp/delegates-overview)
* [Operadores y expresiones](https://docs.microsoft.com/es-es/dotnet/csharp/language-reference/operators/)
* [Instrucciones](https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/statements-expressions-operators/statements)
* [Atributos](https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/concepts/attributes/)

Versión 1.2 de C#

Versión 1.2 de C# incluida en Visual Studio .NET 2003. Contenía algunas pequeñas mejoras del lenguaje. Lo más notable es que, a partir de esa versión, el código se generaba en un bucle foreach llamado [Dispose](https://docs.microsoft.com/es-es/dotnet/api/system.idisposable.dispose) en un [IEnumerator](https://docs.microsoft.com/es-es/dotnet/api/system.collections.ienumerator) cuando ese [IEnumerator](https://docs.microsoft.com/es-es/dotnet/api/system.collections.ienumerator) implementaba [IDisposable](https://docs.microsoft.com/es-es/dotnet/api/system.idisposable).

C# versión 2.0

Aquí las cosas empiezan a ponerse interesantes. Echemos un vistazo a algunas de las principales características de **C# 2.0**, que se publicó en 2005 junto con Visual Studio 2005:

* [Genéricos](https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/generics/)
* [Tipos parciales](https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/classes-and-structs/partial-classes-and-methods#partial-classes)
* [Métodos anónimos](https://docs.microsoft.com/es-es/dotnet/csharp/language-reference/operators/delegate-operator)
* [Tipos de valores que aceptan valores NULL](https://docs.microsoft.com/es-es/dotnet/csharp/language-reference/builtin-types/nullable-value-types)
* [Iteradores](https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/concepts/iterators)
* [Covarianza y contravarianza](https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/concepts/covariance-contravariance/)

Otras características de C# 2.0 agregaron capacidades a las características existentes:

* Accesibilidad independiente de captador o establecedor
* Conversiones de grupos de métodos (delegados)
* Clases estáticas
* Inferencia de delegados

Aunque puede que C# haya comenzado como un lenguaje genérico orientado a objetos, la versión 2.0 de C# cambió esto enseguida. En cuanto se pusieron con ella, se centraron en algunos puntos problemáticos graves para los desarrolladores. Y lo hicieron a lo grande.

Con los genéricos, los tipos y métodos pueden operar en un tipo arbitrario a la vez que conservan la seguridad de tipos. Por ejemplo, tener [List<T>](https://docs.microsoft.com/es-es/dotnet/api/system.collections.generic.list-1) nos permite tener List<string> o List<int> y realizar operaciones de tipo seguro en esas cadenas o en enteros mientras los recorremos en iteración. Usar genéricos es mejor que crear un ListInt que derive de ArrayList o se convierta desde Object en cada operación.

**C# 2.0** incorporó los iteradores. Para explicarlo brevemente, los iteradores permiten examinar todos los elementos de List (u otros tipos enumerables) con un bucle de foreach. Tener iteradores como una parte de primera clase del lenguaje mejoró drásticamente la facilidad de lectura del lenguaje y la capacidad de las personas de razonar sobre el código.

Aun así, C# seguía yendo por detrás de Java. Java ya había publicado versiones que incluían genéricos e iteradores. Pero esto cambiaría pronto a medida que los idiomas siguieran evolucionando.

**C# versión 3.0**

**La versión 3.0 de C#** llegó a finales de 2007, junto con Visual Studio 2008, aunque la cartera completa de características de lenguaje no llegaría realmente hasta la versión 3.5 de .NET Framework. Esta versión marcó un cambio importante en el crecimiento de C#. Estableció C# como un lenguaje de programación realmente formidable. Echemos un vistazo a algunas de las principales características de esta versión:

* [Propiedades implementadas automáticamente](https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/classes-and-structs/auto-implemented-properties)
* [Tipos anónimos (Guía de programación de C#)](https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/classes-and-structs/anonymous-types).
* [Expresiones de consulta](https://docs.microsoft.com/es-es/dotnet/csharp/linq/query-expression-basics)
* [Expresiones lambda](https://docs.microsoft.com/es-es/dotnet/csharp/language-reference/operators/lambda-expressions)
* [Árboles de expresión](https://docs.microsoft.com/es-es/dotnet/csharp/expression-trees)
* [Métodos de extensión](https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/classes-and-structs/extension-methods)
* [Variables locales con asignación implícita de tipos](https://docs.microsoft.com/es-es/dotnet/csharp/language-reference/keywords/var)
* [Métodos parciales](https://docs.microsoft.com/es-es/dotnet/csharp/language-reference/keywords/partial-method)
* [Inicializadores de objeto y colección](https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/classes-and-structs/object-and-collection-initializers)

En retrospectiva, muchas de estas características parecen inevitables e indivisibles. Todas ellas encajan estratégicamente. Por lo general se considera que la mejor característica de la versión de C# fue la expresión de consulta, también conocida como Language-Integrated Query (LINQ).

Una vista más matizada examina árboles de expresión, expresiones lambda y tipos anónimos como la base sobre la que se construye LINQ.

Sin embargo, en cualquier caso, **C# 3.0** presentó un concepto revolucionario. **C# 3.0** había comenzado a sentar las bases para convertir C# en un lenguaje híbrido funcional y orientado a objetos.

En concreto, permitía escribir consultas declarativas en estilo de SQL para realizar operaciones en colecciones, entre otras cosas. En lugar de escribir un bucle de for para calcular el promedio de una lista de enteros, permitía hacerlo fácilmente como list.Average(). La combinación de métodos de extensión y expresiones de consulta hizo que esa lista de enteros pareciera haberse vuelto más inteligente.

Llevó tiempo hasta que los usuarios realmente captaron e integraron el concepto, pero ocurrió gradualmente. Ahora, años más tarde, el código es mucho más conciso, sencillo y funcional.

**C# versión 4.0**

**La versión 4.0 de C#**, publicada con Visual Studio 2010, tuvo que lidiar con el carácter innovador que había adquirido la versión 3.0. Con la versión 3.0, el lenguaje de C# dejó de estar a la sombra de Java y alcanzó una posición prominente. El lenguaje se estaba convirtiendo rápidamente en algo elegante.

La siguiente versión introdujo algunas nuevas características interesantes:

* [Enlace dinámico](https://docs.microsoft.com/es-es/dotnet/csharp/language-reference/builtin-types/reference-types)
* [Argumentos opcionales/con nombre](https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/classes-and-structs/named-and-optional-arguments)
* [Covariante y contravariante de genéricos](https://docs.microsoft.com/es-es/dotnet/standard/generics/covariance-and-contravariance)
* [Tipos de interoperabilidad insertados](https://docs.microsoft.com/es-es/dotnet/framework/interop/type-equivalence-and-embedded-interop-types)

Los tipos de interoperabilidad insertados solucionaron un problema de implementación. La covarianza y contravarianza de genéricos proporcionan más capacidad para usar genéricos, pero son más bien académicos y probablemente más valorados por autores de bibliotecas y Framework. Los parámetros opcionales y con nombre permiten eliminar muchas sobrecargas de métodos y proporcionan mayor comodidad. Pero ninguna de esas características está modificando el paradigma exactamente.

La característica más importante fue la introducción de la palabra clave dynamic. Con la palabra clave dynamic, en la versión 4.0 de C# se introdujo la capacidad de invalidar el compilador durante la escritura en tiempo de compilación. Al usar la palabra clave dinámica, puede crear constructos similares a los lenguajes tipados dinámicamente, como JavaScript. Puede crear dynamic x = "a string" y luego agregarle seis, dejando que el runtime decida qué debería suceder después.

Los enlaces dinámicos pueden dar lugar a errores, pero también otorgan un gran poder sobre el lenguaje.

**C# versión 5.0**

**La versión 5.0** de C#, publicada con Visual Studio 2012, era una versión centrada del lenguaje. Casi todo el trabajo de esa versión se centró en otro concepto de lenguaje innovador: el modelo async y await para la programación asincrónica. Estas son las principales características:

* [Miembros asincrónicos](https://docs.microsoft.com/es-es/dotnet/csharp/async)
* [Atributos de información del llamador](https://docs.microsoft.com/es-es/dotnet/csharp/language-reference/attributes/caller-information)

Vea también

* [Proyecto de código: Atributos de información del autor de llamada en C# 5.0](https://www.codeproject.com/Tips/606379/Caller-Info-Attributes-in-Csharp)

El atributo de información del autor de la llamada permite recuperar fácilmente información sobre el contexto donde se está ejecutando sin tener que recurrir a una gran cantidad de código de reflexión reutilizable. Tiene muchos usos en tareas de registro y diagnóstico.

Pero async y await son los auténticos protagonistas de esta versión. Cuando estas características salieron a la luz en 2012, C# cambió de nuevo las reglas del juego al integrar la asincronía en el lenguaje como un participante de primera clase. Si alguna vez ha trabajado con operaciones de larga duración y la implementación de sitios web de devoluciones de llamada, probablemente le haya encantado esta característica del lenguaje.

**C# versión 6.0**

Con las versiones 3.0 y 5.0, C# había agregado nuevas características destacables a un lenguaje orientado a objetos. Con la versión 6.0, publicada con Visual Studio 2015, en lugar de introducir una característica innovadora y predominante, se publicaron muchas características menores que aumentaron la productividad de la programación de C#. Estas son algunas de ellas:

* [Importaciones estáticas](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-6#using-static)
* [Filtros de excepciones](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-6#exception-filters)
* [Inicializadores de propiedades automáticas](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-6#auto-property-initializers)
* [Miembros de cuerpo de expresión](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-6#expression-bodied-function-members)
* [Propagador de null](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-6#null-conditional-operators)
* [Interpolación de cadenas](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-6#string-interpolation)
* [operador nameof](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-6#the-nameof-expression)
* [Inicializadores de índice](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-6#extension-add-methods-in-collection-initializers)

Entre las otras características nuevas se incluyen estas:

* Await en bloques catch y finally
* Valores predeterminados para las propiedades solo de captador

Cada una de estas características es interesante en sí misma. Pero si las observamos en su conjunto, vemos un patrón interesante. En esta versión, C# eliminó lenguaje reutilizable para que el código fuera más fluido y fácil de leer. Así que, para los que adoran el código simple y limpio, esta versión del lenguaje fue una gran aportación.

En esta versión también se hizo otra cosa, aunque no es una característica de lenguaje tradicional: publicaron el [compilador Roslyn como un servicio](https://github.com/dotnet/roslyn). Ahora, el compilador de C# está escrito en C# y puede usarlo como parte de su trabajo de programación.

**C# versión 7.0**

**C# versión 7.0** se comercializó con Visual Studio 2017. Esta versión tiene algunas cosas interesantes y evolutivas en la misma línea que C# 6.0, pero sin el compilador como servicio. Estas son algunas de las nuevas características:

* [Variables out](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-7#out-variables)
* [Tuplas y deconstrucción](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-7#tuples)
* [Coincidencia de patrones](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-7#pattern-matching)
* [Funciones locales](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-7#local-functions)
* [Miembros con forma de expresión expandidos](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-7#more-expression-bodied-members)
* [Devoluciones y variables locales ref](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-7#ref-locals-and-returns)

Otras características incluidas:

* [Descartes](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-7#discards)
* [Literales binarios y separadores de dígitos](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-7#numeric-literal-syntax-improvements)
* [Expresiones throw](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-7#throw-expressions)

Todas estas características ofrecen capacidades nuevas e interesantes para los desarrolladores y la posibilidad de escribir un código de manera más clara que nunca. De manera destacada, condensan la declaración de variables que se van a usar con la palabra clave out y permiten varios valores devueltos a través de tuplas.

Pero C# se está usando cada vez más. .NET Core ahora tiene como destino cualquier sistema operativo y tiene puesta la mirada en la nube y la portabilidad. Por supuesto, esas nuevas capacidades ocupan las ideas y el tiempo de los diseñadores de lenguaje, además de ofrecer nuevas características.

**C# versión 7.1**

C# empezó a publicar *versiones de punto* con C# 7.1. Esta versión agregó el elemento de configuración de [selección de versión de lenguaje](https://docs.microsoft.com/es-es/dotnet/csharp/language-reference/configure-language-version), tres nuevas características de lenguaje y un nuevo comportamiento del compilador.

Las nuevas características de lenguaje de esta versión son las siguientes:

* [Método async Main](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-7-1#async-main)
  + El punto de entrada de una aplicación puede tener el modificador async.
* [Expresiones literales default](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-7-1#default-literal-expressions)
  + Se pueden usar expresiones literales predeterminadas en expresiones de valor predeterminadas cuando el tipo de destino se pueda inferir.
* [Nombres de elementos de tupla inferidos](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-7-1#inferred-tuple-element-names)
  + En muchos casos, los nombres de elementos de tupla se pueden deducir de la inicialización de la tupla.
* [Coincidencia de patrones en parámetros de tipo genérico](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-7-1#pattern-matching-on-generic-type-parameters)
  + Puede usar expresiones de coincidencia de patrones en variables cuyo tipo es un parámetro de tipo genérico.

Por último, el compilador tiene dos opciones, -refout y -refonly, que controlan la [generación de ensamblados de referencia](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-7-1#reference-assembly-generation).

**C# versión 7.2**

**C#7.2** agregó varias características de lenguaje pequeñas:

* [Técnicas para escribir código eficiente seguro](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-7-2#safe-efficient-code-enhancements)
  + Una combinación de mejoras en la sintaxis que permiten trabajar con tipos de valor mediante la semántica de referencia.
* [Argumentos con nombre no finales](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-7-2#non-trailing-named-arguments)
  + Los argumentos con nombre pueden ir seguidos de argumentos posicionales.
* [Caracteres de subrayado iniciales en literales numéricos](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-7-2#leading-underscores-in-numeric-literals)
  + Los literales numéricos ahora pueden tener caracteres de subrayado iniciales antes de los dígitos impresos.
* [Modificador de acceso private protected](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-7-2#private-protected-access-modifier)
  + El modificador de acceso private protected permite el acceso de clases derivadas en el mismo ensamblado.
* [Expresiones ref condicionales](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-7-2#conditional-ref-expressions)
  + El resultado de una expresión condicional (?:) ahora puede ser una referencia.

**C# versión 7.3**

Hay dos temas principales para la versión C# 7.3. Un tema proporciona características que permiten al código seguro ser tan eficaz como el código no seguro. El segundo tema proporciona mejoras incrementales en las características existentes. Además, se han agregado nuevas opciones de compilador en esta versión.

Las siguientes características nuevas admiten el tema del mejor rendimiento para código seguro:

* [Puede acceder a campos fijos sin anclar.](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-7-3#indexing-fixed-fields-does-not-require-pinning)
* [Puede reasignar variables locales ref.](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-7-3#ref-local-variables-may-be-reassigned)
* [Puede usar inicializadores en matrices stackalloc.](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-7-3#stackalloc-arrays-support-initializers)
* [Puede usar instrucciones fixed con cualquier tipo que admita un patrón.](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-7-3#more-types-support-the-fixed-statement)
* [Puede usar restricciones genéricas adicionales.](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-7-3#enhanced-generic-constraints)

Se hicieron las mejoras siguientes a las características existentes:

* [Puede probar == y != con tipos de tupla.](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-7-3#tuples-support--and-)
* [Puede usar variables de expresión en más ubicaciones.](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-7-3#extend-expression-variables-in-initializers)
* [Puede asociar atributos al campo de respaldo de las propiedades autoimplementadas.](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-7-3#attach-attributes-to-the-backing-fields-for-auto-implemented-properties)
* [Se ha mejorado la resolución de métodos cuando los argumentos difieren en in.](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-7-3#in-method-overload-resolution-tiebreaker)
* [La resolución de sobrecarga tiene ahora menos casos ambiguos.](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-7-3#improved-overload-candidates)

Las nuevas opciones del compilador son:

* [-publicsign para habilitar la firma de ensamblados de software de código abierto (OSS).](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-7-3#public-or-open-source-signing)
* [-pathmap para proporcionar una asignación para los directorios de origen.](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-7-3#pathmap)

**C# versión 8.0**

**C# 8.0** es la primera versión C# principal que tiene como destino específicamente .NET Core. Algunas características se basan en nuevas funcionalidades de CLR, otras en tipos de biblioteca agregados solo a .NET Core. C# 8.0 agrega las siguientes características y mejoras al lenguaje C#:

* [Miembros de solo lectura](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-8#readonly-members)
* [Métodos de interfaz predeterminados](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-8#default-interface-methods)
* [Mejoras de coincidencia de patrones](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-8#more-patterns-in-more-places):
  + [Expresiones switch](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-8#switch-expressions)
  + [Patrones de propiedades](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-8#property-patterns)
  + [Patrones de tupla](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-8#tuple-patterns)
  + [Patrones posicionales](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-8#positional-patterns)
* [Declaraciones using](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-8#using-declarations)
* [Funciones locales estáticas](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-8#static-local-functions)
* [Estructuras ref descartables](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-8#disposable-ref-structs)
* [Tipos de referencia que aceptan valores null](https://docs.microsoft.com/es-es/dotnet/csharp/language-reference/builtin-types/nullable-reference-types)
* [Secuencias asincrónicas](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-8#asynchronous-streams)
* [Índices y rangos](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-8#indices-and-ranges)
* [Asignación de uso combinado de NULL](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-8#null-coalescing-assignment)
* [Tipos construidos no administrados](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-8#unmanaged-constructed-types)
* [Stackalloc en expresiones anidadas](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-8#stackalloc-in-nested-expressions)
* [Mejora de las cadenas textuales interpoladas](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-8#enhancement-of-interpolated-verbatim-strings)

Los miembros de interfaz predeterminados requieren mejoras en CLR. Estas características se agregaron en CLR para .NET Core 3.0. Los intervalos y los índices, y los flujos asincrónicos requieren nuevos tipos en las bibliotecas de .NET Core 3.0. Los tipos de referencia que aceptan valores NULL, aunque se implementan en el compilador, son mucho más útiles cuando se anotan bibliotecas para proporcionar información semántica relativa al estado NULL de los argumentos y los valores devueltos. Esas anotaciones se agregan a las bibliotecas de .NET Core.

**C# 9.0** agrega las siguientes características y mejoras al lenguaje C#:

* [Registros](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-9#record-types)
* [Establecedores de solo inicialización](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-9#init-only-setters)
* [Instrucciones de nivel superior](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-9#top-level-statements)
* [Mejoras de coincidencia de patrones](https://docs.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-9#pattern-matching-enhancements)
* Enteros con tamaño nativos
* Punteros de función
* Supresión de la emisión de la marca localsinit
* Expresiones nuevas con tipo de destino
* Funciones anónimas estáticas
* Expresiones condicionales con tipo de destino
* Tipos de valor devueltos de covariante
* Compatibilidad con extensiones GetEnumerator para bucles foreach
* Parámetros de descarte lambda
* Atributos en funciones locales
* Inicializadores de módulo
* Nuevas características para métodos parciales

C# 9.0 es compatible con **.NET 5**. Para obtener más información, vea [Control de versiones del lenguaje C#](https://docs.microsoft.com/es-es/dotnet/csharp/language-reference/configure-language-version).

1. **PRINCIPALES CARACTERISTICAS DE C#**

* **Sintaxis sencilla**. La sintaxis de C# es muy similar a Java, lo que simplifica al desarrollador a la hora de escribir código.
* **Escritura**. En C# tienes que declarar los tipos de datos con las variables, además que estas variables no pueden ser globales. De esta manera se reducen los errores de compilación. Además, es un **lenguaje orientado a objetos**, así que obliga que todos los métodos y propiedades estén dentro de una clase.
* **Orientación a componentes**. Además de ser orientado a objetos, como hemos dicho antes, es orientado a componentes. Podemos definir propiedades sin necesidad de crear métodos, como en Java, o usar eventos sin tratar con punteros a funciones. Escribir código en C# es mucho más sencillo y potente que en otros lenguajes de programación.
* **Sistema de tipos unificado**. Todos los tipos de datos sencillos de C# derivan de una clase común llamada **System.Object**. De esta manera, la conversión de tipos sólo es compatible entre los tipos más cercanos (números enteros con decimales, por ejemplo), además de facilitar la creación de colecciones.
* **CLR (Common Language Runtime)**. Es el núcleo de .NET, el motor que ejecuta las aplicaciones programadas en C#, lo que ayuda a la integración del sistema operativo o dlls externas.
* **Espacio de nombres.** Puedes agrupar o aislar código mediante namespaces, parecido al package de Java.
* **Polimorfismo**. En C++ se permite la herencia múltiple, pero en C# no, aunque puedes implementar interfaces. De esta manera se evitan complicaciones y simplifica la escritura.
* **Operadores sobrescritos**. En C# puedes recodificar un operador, como puede ser el sumatorio + o el incremento ++.
* **Bibliotecas**. Cualquier compilador de C# viene con un mínimo de biblioteca de clases disponibles y que puedes usar.
* **Eliminación de los problemas de DLLs**. Aunque puedes crear dlls en .NET, no te encuentras con la problemática de la sustitución de versiones antiguas de dlls compartidas con versiones nuevas. Pueden existir diferentes versiones de la misma dll, simplificando la instalación del softwasre.
* **Multiplataforma**. Actualmente, el lenguaje C# es de código abierto y se ha utilizado en otros IDEs, como el proyecto MoÇÇno o Xamarin, y en múltiples sistemas operativos, como puede ser OSx o Android.
* **Integración con otros lenguajes**. Cualquier lenguaje que se compile con .NET, como la nueva versión de visual basic, puede aprovecharse para usar en tu proyecto.
* **Mejora en la gestión de memoria**. Al igual que Java, en C# dispone de un recolector de basura que destruye los objetos que no se usan en memoria.
* **Tratamiento de errores**. Cualquier lenguaje de programacion moderno utiliza las excepciones para controlar los posibles errores en el código.
* **Multihilo**. Puedes dividir tu código en múltiples hilos de ejecución, trabajar en paralelo y sincronizándose al final.

**using System;**

**namespace HelloNameSpace{**

**public class HelloWorld{static void Main(string[] args) {**

**Console.WriteLine("Hola Mundo!");**

**}**

**}**

**}**

1. **CURVA DE APRENDIZAJE EN C#**

La sintaxis de C# se base en C++ (y en Java), por lo que a priori podría parecer un lenguaje complejo para principiantes. Sin embargo, las opciones de autocompletado de Visual Studio, la auto-creación de proyectos y la facilidad de uso de su entorno de desarrollo en general, son aspectos que hacen que este lenguaje sea una buena opción para las personas que se inician en la programación.

1. **PLATAFORMAS SOPORTADAS**

Es multiplataforma tanto a nivel de hardware como de sistemas operativos

1. **ECOSISTEMA / SOPORTE /**

**COMINIDAD**

C# a tenido una gran aceptacion en el ambito profesional pues es uno de los 10 lenguajes de mayor ranking mundial, tiene una extensa comunidad de programadores, tanto profesionales como amateurs.

Tiene un gran soporte, pues muchas empresas contrullen su software en este lenguaje, o se dedican a auditar empresas con software programado en este lenguaje.

1. **ENTORNO DE DESARROLLO DE C#**

* JetBrains Rider
* **Visual Studio**
* **Visual Studio Code**
* MonoDevelop

1. **NIVEL DE DEMANDA DE C#**

C# es el 4 lenguaje mas demandado en el mundo, pero en España no es tan demandado

Imagen que contiene Mapa

Descripción generada automáticamenteImagen que contiene Escala de tiempo

Descripción generada automáticamente

1. **WEBGRAFIA**

https://recluit.com/historia-del-lenguaje-c/#.X3nsLe3tZhF

https://es.wikipedia.org/wiki/Anders\_Hejlsberg

https://lenguajesdeprogramacion.net/c-sharp/

https://es.wikipedia.org/wiki/C\_Sharp

http://www.holamundo.es/lenguaje/c-sharp/

https://www.slant.co/topics/4118/~c-ides#8