

Manual de C#



Miguel Angel Alvarez



desarrolloweb.com/manuales/manual-jquery.html

Introducción: Manual de C

Este es el Manual de C#, el lenguaje más popular creado por Microsoft para su plataforma .NET. C Sharp es un lenguaje avanzado y moderno, que ofrece unas prestaciones excelentes para los programadores y las aplicaciones.

De entre todos los lenguajes que se pueden usar dentro de la plataforma .NET de Microsoft, C# es el más extendido, del que encontrarás mayor cantidad de documentación y una nutrida comunidad publicando guías y tutoriales.

En este manual pretendemos enseñar las características de C# pensando en lectores que ya tienen algunos conocimientos sobre programación, incluso sobre el paradigma de la programación orientada a objetos. Por tanto, iremos rápidamente abordando los puntos básicos, sin detenernos demasiado en explicar las bases de la programación, con el objetivo de realizar prácticas de un poco más avanzadas en las que cubrir aspectos más específicos sobre el lenguaje.

Como nuestro objetivo principal es abordar el lenguaje C#, Vamos a realizar únicamente aplicaciones de consola, que podrás ejecutar en cualquier plataforma, como Windows, Linux o Mac.

Encuentras este manual online en:

<http://desarrolloweb.com/manuales/manual-c-sharp>

Autores del manual

Las siguientes personas han participado como autores escribiendo artículos de este manual.

Miguel Angel Alvarez

Fundador de DesarrolloWeb.com y la plataforma de formación online EscuelaIT. Comenzó en el mundo del desarrollo web en el año 1997, transformando su hobby en su trabajo.



Obtener .NET y configurar tu IDE o editor para trabajar con C

Primeros pasos para el desarrollo con C#. Cómo instalar la plataforma .NET, alternativas de editores y IDE. Cómo configurar un primer proyecto tanto en Visual Studio Code como en Visual Studio.



Podemos usar C# en muchos contextos, como desarrollo de programas para consola, servicios web, aplicaciones para móviles o juegos. Dependiendo del tipo de programa que quieras hacer podrás necesitar unos u otros complementos. Por ejemplo para el desarrollo de aplicaciones móviles usarás Xamarin o para juegos tienes Unity, entre otros motores. Sin embargo, si quieres empezar con C# lo más adecuado es descargarse la plataforma .NET de Microsoft.

.NET es una plataforma libre de código abierto para desarrollar todo tipo de aplicaciones. Algunos de estos tipos, por ejemplo las aplicaciones de escritorio con interfaz gráfica, únicamente las vamos a poder desarrollar sobre Windows, pero con .NET podemos desarrollar programas de consola o servicios web que corren perfectamente en Windows, Linux y MacOS. Es decir, tengas el sistema que tengas, puedes aprender y usar C#.

En principio vamos a aprender C# desarrollando sencillos programas de consola, puesto que lo más importante es soltarnos con el lenguaje, y más adelante, que cada cual realice sus tipos de aplicaciones con el framework que requiera su actividad.

Descarga de .NET multiplataforma

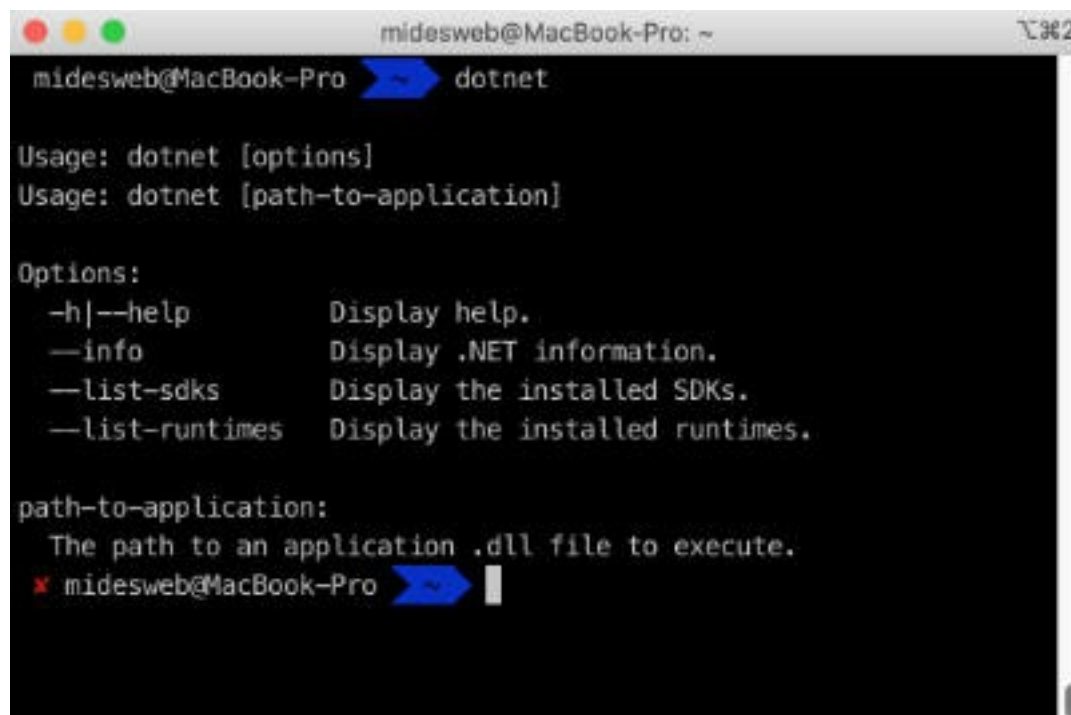
Así que nos vamos a ir a la pagina de "Dotnet" y allí vamos a descargar .NET para el sistema sobre el que estemos trabajando. Desde la página de descargas encontrarás la versión adecuada para tu sistema operativo.

<https://dotnet.microsoft.com/download>

Recomiendo descargar la versión más reciente que encuentres del SDK de .NET. En el momento de escribir estas líneas estamos en .NET 5.0, sin embargo, los conocimientos que vamos a abordar en este manual servirán perfectamente para cualquier versión de .NET, ya que vamos a tocar las cosas básicas y esenciales para trabajar.

El proceso de descarga puede depender del sistema operativo, pero en la página de descargas explican los pasos que necesitarás para tu sistema.

Puedes verificar tu instalación (al menos en Mac) abriendo una consola de comandos y luego ejecutando el comando "dotnet". Verás una salida como la de la siguiente imagen.



```
midesweb@MacBook-Pro: ~  
midesweb@MacBook-Pro ~$ dotnet  
Usage: dotnet [options]  
Usage: dotnet [path-to-application]  
  
Options:  
-h|--help          Display help.  
-info              Display .NET information.  
-list-sdks          Display the installed SDKs.  
-list-runtimes      Display the installed runtimes.  
  
path-to-application:  
The path to an application .dll file to execute.  
x midesweb@MacBook-Pro ~$
```

Realmente esa salida nos está indicando que el comando se conoce en nuestro sistema, aunque no lo hemos usado como se debe, porque deberíamos indicar qué es lo que se quiere ejecutar. De momento lo dejamos ahí.

Si quieres también puedes ejecutar el comando "dotnet --version" para obtener la versión que se está ejecutando en tu máquina.



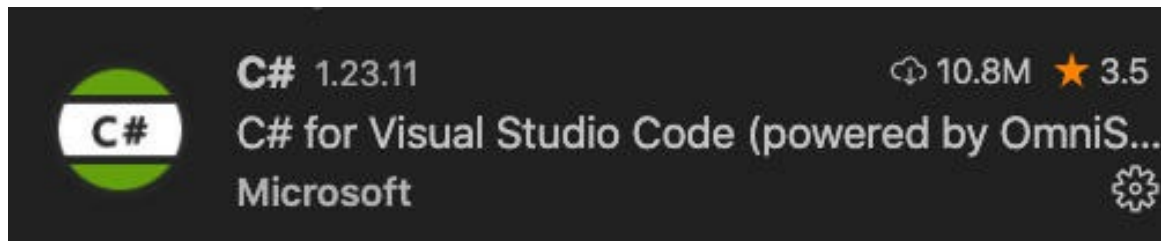
```
midesweb@MacBook-Pro ~$ dotnet --version  
5.0.202
```

Qué IDE o editor de código escoger para el desarrollo en .NET

Realmente podrías usar cualquier editor de código para escribir tus programas en C# y .NET. Un editor muy conocido y ligero que recomendamos es Visual Studio Code, que además tiene una serie de complementos y extensiones muy poderosos para trabajar con .NET.

Extensiones de Visual Studio Code recomendadas

La extensión que debemos instalar, sí o sí, para trabajar en buenas condiciones con Visual Studio Code se llama "C#" y está creada por la propia Microsoft.



Esta extensión nos permite ejecutar y depurar aplicaciones .NET dentro del propio Visual Studio Code.

Alternativa más profesional con Visual Studio

Como decimos, realmente todo lo que vamos a hacer es perfectamente posible realizarlo con cualquier editor para programadores, porque vamos a trabajar con cosas sencillas. Sin embargo, otra posibilidad muy interesante es usar Visual Studio, que es un IDE un tanto más pesado, pero que también nos aporta ciertas ventajas a la hora de editar código.

Visual Studio en Windows tiene una versión "Community" que es gratuita. Para MacOS tienes "Visual Studio para Mac" que también es un IDE gratuito. Para Linux no está disponible Visual Studio, pero tienes Visual Studio Code, que como decía, con las extensiones adecuadas funcionará bastante bien.

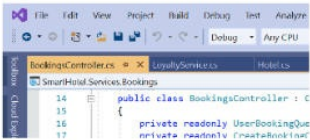
Los enlaces de descarga pueden cambiar, porque con el tiempo Microsoft siempre anda cambiando todas sus páginas, pero en esta dirección encuentras actualmente una buena referencia para saber cuáles son las versiones de Visual Studio gratuitas dependiendo del sistema operativo.

<https://visualstudio.microsoft.com/es/free-developer-offers/>

Todo lo necesario para crear aplicaciones geniales. Gratis.

Visual Studio Community

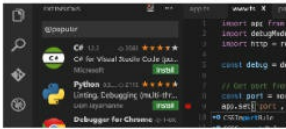
IDE completo para programar, depurar, probar e implementar soluciones en cualquier plataforma. [Más información](#)



[Descarga gratuita](#)

Visual Studio Code

Edición y depuración en cualquier sistema operativo. [Más información](#)

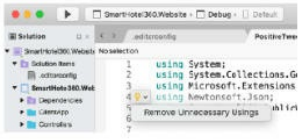


Al usar Visual Studio Code, confirma que acepta la licencia y declaración de privacidad

[Descarga gratuita](#)

Visual Studio Community para Mac

Desarrollo de aplicaciones y juegos para iOS, Android y la Web mediante .NET. [Más información](#)



[Más información acerca de la activación de la licencia](#)

[Descarga gratuita](#)

Yo personalmente uso Visual Studio para Mac, porque vengo desarrollando juegos con Unity y es la herramienta que ofrece la mejor experiencia de desarrollo con este motor de juegos. Aunque para este manual estoy usando más Visual Studio Code, porque es un editor más común, disponible en cualquier plataforma. La decisión de elegir uno u otro programa, no obstante es tuya.

Puedes descargar el IDE específico para tu sistema operativo, que dependerá de cada caso, pero realmente no es más complicado que instalar cualquier otro programa en tu ordenador. Lo que sí cambiará un poco es la manera en la que comenzamos un proyecto con uno u otro IDE/editor, por eso lo vamos a explicar a continuación.

Primeros pasos para programar en C# con Visual Studio Code

Si usas Visual Studio Code para comenzar a desarrollar con C# simplemente necesitarás crear una carpeta y trabajar en ella, creando tus archivos con código en C#, que tendrán la extensión ".cs".

Además, generalmente necesitarás un proyecto, que se consigue mediante un fichero de configuración que indica diversas configuraciones que se van a aplicar. Este proyecto lo puedes crear dentro de una carpeta vacía de tu disco duro.

Podemos crear proyectos cómodamente con la herramienta de línea de comandos "dotnet" y el comando siguiente:

```
dotnet new
```

Entonces te pregunta qué tipo de proyecto es el que quieres crear, para especificarlo cuando haces el "new", ofreciendo varias alternativas para que puedas elegir la que necesites. Para los programas que vamos a realizar en este manual, que son básicamente programas de consola, usaremos concretamente este comando:

```
dotnet new console
```



```
midesweb@MacBook-Pro: ~/sites/sandbox/primeros-pasos-c-sharp
midesweb@MacBook-Pro ~$ cd sites/sandbox
midesweb@MacBook-Pro ~/sites/sandbox$ clear
midesweb@MacBook-Pro ~/sites/sandbox$ mkdir primeros-pasos-c-sharp
midesweb@MacBook-Pro ~/sites/sandbox$ cd primeros-pasos-c-sharp
midesweb@MacBook-Pro ~/sites/sandbox/primeros-pasos-c-sharp$ dotnet new console
The template "Console Application" was created successfully.

Processing post-creation actions...
Running 'dotnet restore' on /Users/midesweb/sites/sandbox/primeros-pasos-c-sharp/primeros-pasos-c-sharp.csproj...
  Determinando los proyectos que se van a restaurar...
  Se ha restaurado /Users/midesweb/sites/sandbox/primeros-pasos-c-sharp/primeros-pasos-c-sharp.csproj (en 70 ms).
Restore succeeded.

midesweb@MacBook-Pro ~/sites/sandbox/primeros-pasos-c-sharp$
```

Una vez realizado este comando se crearán varios archivos sobre la carpeta. En concreto nos interesará uno en particular con extensión ".csproj", que tiene el nombre de la carpeta donde lo hemos creado. Este archivo .csproj es donde se indica qué tipo de proyecto es y otras cuestiones de configuración.

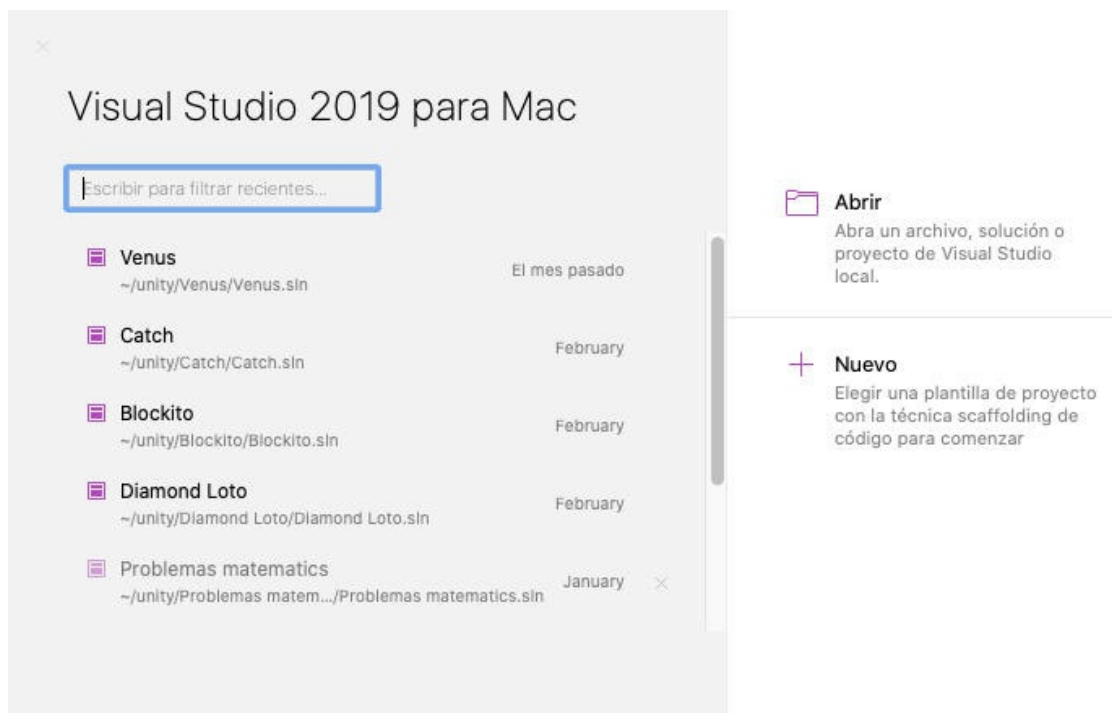
Ahora mismo en la carpeta de nuestro proyecto podremos realizar acciones adicionales sobre el mismo, con comandos como:

- dotnet build: para compilar el proyecto
- dotnet run: para ejecutar el proyecto.

Volveremos más adelante sobre estos comandos y la operativa de trabajo con los proyectos cuando pongamos manos en el código.

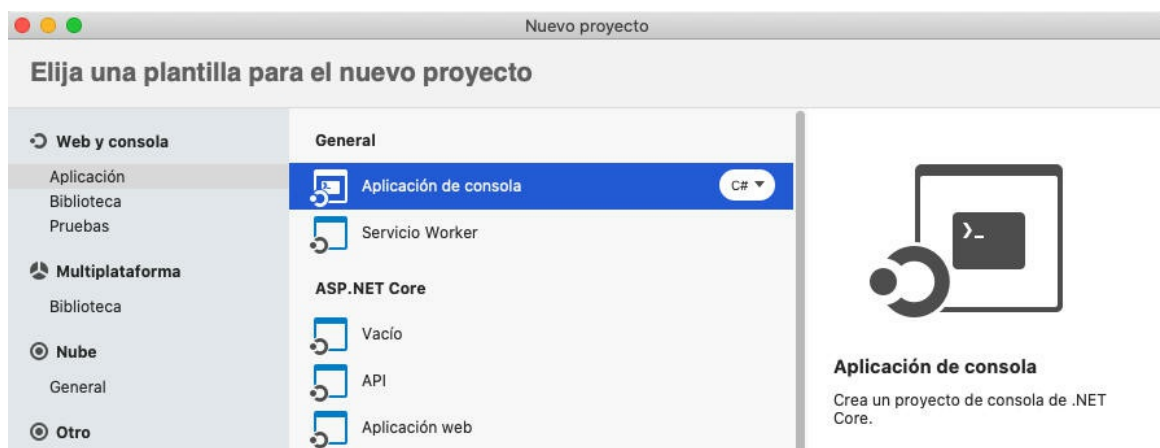
Primeros pasos con Visual Studio

Una vez que tenemos instalado Visual Studio nos sale una pantalla inicial con los proyectos recientes donde podremos simplemente recuperar un uno de ellos o crear un nuevo proyecto nuevo.



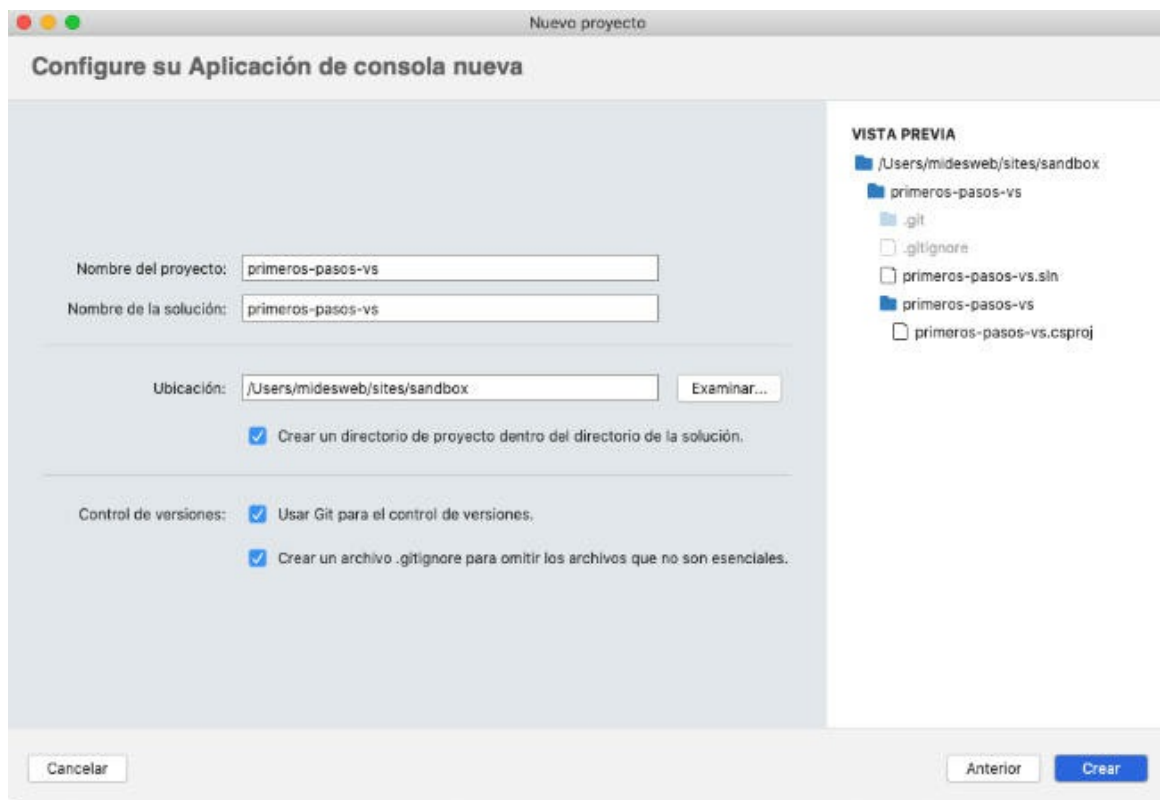
Al elegir la opción de crear un proyecto nuevo nos aparecerán las plantillas del proyecto que queremos realizar. Es básicamente lo mismo que podríamos ejecutar con los comandos de consola que hemos comentado anteriormente para Visual Studio Code, pero con interfaz gráfica.

Vamos a elegir la plantilla que nos permitirá realizar un programa de consola con C#.



En la pantalla siguiente nos pregunta la plataforma de destino, pudiendo ser cualquiera de las versiones de .NET que tengas instaladas en tu sistema.

En la última pantalla nos solicitará el nombre del proyecto que vamos a crear, la carpeta donde lo vamos a guardar y algunas cosas extra como si queremos usar control de versiones con Git. Esto lo puedes configurar a tu gusto.



Archivos del proyecto de consola

Tanto si hemos usado Visual Studio como Visual Studio Code los archivos de la carpeta de proyecto se parecerán bastante.

Dentro de estas carpetas de proyecto esta archivos como:

- El archivo `.csproj`: que es un XML de configuración. De momento no hace falta preocuparse mucho de él, pero puedes abrirlo para ver qué contenido tiene.
- Archivo `Program.cs`, que es el archivo de código de nuestro programa, es donde vamos a programar a continuación.

Luego verás otras carpetas que se han creado en el directorio del proyecto. Generalmente no las vamos a necesitar tocar nosotros manualmente, así que no vamos a nombrarlas siquiera. De momento no nos importan mucho.

Conclusión

Con esto hemos completado el primer paso para comenzar a trabajar con C#, simplemente creando un proyecto inicial donde, más adelante, iremos escribiendo el código de nuestro programa.

A continuación iremos aprendiendo más temas específicos de programación y menos de configuración del entorno, que como digo puede ser realizado de diversas maneras dependiendo del sistema operativo en el que te encuentres y de las preferencias que tengas a la hora de elegir tu editor. Visual Studio Code tiene la ventaja de ser más ligero, completamente gratuito y multiplataforma. Visual Studio tiene la ventaja de ofrecerte un entorno más

especializado en el desarrollo para .NET y C#, con algunas ayudas extra para los desarrolladores, que harán la diferencia en proyectos más avanzados de los que vamos a ver en este manual.

Este artículo es obra de *Miguel Angel Alvarez*

Fue publicado / actualizado en 28/05/2021

Disponible online en <https://desarrolloweb.com/articulos/obtener-dotnet-configurar-ide-editor-c>

Primer programa en C

Este sería nuestro Hola Mundo en C#, en un proyecto de consola. Veremos cómo crear un primer programa, conoceremos el método main y los medios para poder ejecutar y compilar el código de este primer programa en C#.



En el artículo anterior estuvimos estudiando el [inicio de los proyectos en C#](#), desde la instalación de la plataforma .NET a los primeros pasos para crear proyectos, que cambian dependiendo del editor o IDE que estás usando. Esa parte no la vamos a repetir, por lo que partimos de un proyecto recién creado con el comando:

```
dotnet new console
```

Esto nos ha dejado una estructura de archivos y carpetas en el directorio donde hemos iniciado el proyecto. Entre todos ellos encuentras uno llamado "Program.cs". Ese es el archivo que está configurado como arranque de nuestro proyecto y donde vamos a comenzar a poner las manos en el código.

Clase de arranque de un proyecto de consola en C

Observarás que el archivo Program.cs ya viene con el esqueleto de una clase, de [Programación Orientada a Objetos](#). Esta clase ya contiene el código de un "Hola mundo" en C#, por lo que nos han adelantado trabajo para esta presentación práctica de C#.

El código que encontramos es el siguiente:

```
using System;

namespace manual_c_sharp
{
    class Program
    {
        static void Main(string[] args)
```

```
{  
    Console.WriteLine("Hello World!");  
}  
}
```

Lo único que podrá cambiar en el código que habrás generado en tu caso es el namespace de la clase, que en mi caso se llaman "manual_c_sharp", que es el nombre de la carpeta donde había generado este nuevo proyecto.

Este archivo de arranque tiene estas particularidades:

Primero se requiere un namespace dentro de este componente, uno que nos viene proporcionado por la propia plataforma .NET, que es System.

```
using System;
```

Luego se declara el namespace de esta clase que acabamos de crear.

```
namespace manual_c_sharp
```

Este namespace se podría cambiar perfectamente de nombre, a voluntad del desarrollador, si se juzgara necesario, para algo como "ManualCSharp" o cualquier cosa que queramos. Es independiente el nombre de la carpeta donde se encuentre la clase, es decir, no es necesario que coincida el nombre del namespace con el nombre de la carpeta donde has colocado ese archivo. Sí es importante que la clase esté dentro de un namespace, pues es un requisito del lenguaje. No puede haber ninguna clase fuera de un namespace.

Un namespace viene a ser lo que la propia palabra indica, un espacio de nombres. Es la manera de organizar las clases de un proyecto, en diversos paquetes para conseguir formar la arquitectura de la aplicación. También permite que las clases de un namespace se nombren libremente, sin pensar en que esos nombres se hayan utilizado en otros espacios de nombres, ya que no colisionarían por formar parte de paquetes distintos.

Dentro del namespace encontramos ya la declaración de de la clase que estamos programando.

```
class Program {  
    // ...  
}
```

La clase tiene el mismo nombre que el archivo que se había generado, con la primera letra en mayúscula, como es convención en los nombres de clases.

Método Main

Dentro de la clase encontramos el método Main. Este es el método que se ejecutará al ponerse en marcha la aplicación, que debe hacer el arranque del programa.

```
static void Main(string[] args)
{
    Console.WriteLine("Hello World!");
}
```

Main es un método estático que no devuelve nada y que recibe un array de argumentos. Un método estático significa que es un método que se ejecuta sobre la clase. La declaración "void" es la que indica que no va a devolver nada el método.

Los que ya habéis tenido experiencia con otros lenguajes de programación similares, como Java, seguramente os parecerá todo muy familiar. Para los que no, simplemente decir que toda clase puede tener un método Main que se utiliza para ejecutarla. En ella generalmente se instanciarán objetos de esa clase y se pondrán en funcionamiento. Como esta es la clase principal del programa, este método es fundamental, porque servirá de arranque a todo el proyecto.

Dentro de este método estamos utilizando la clase Console que nos provee el namespace System. Es una clase que contiene diversos métodos, en este caso estamos haciendo uso de su método estático WriteLine() que se encarga de enviar texto a la consola.

En eso consiste nuestro primer programa en C#. Como has visto, todo el código nos lo aporta el propio generador del proyecto.

Ejecutar un programa de C

Ahora vamos a poner en marcha este proyecto. Para ello podemos usar la consola. Nos vamos a la carpeta raíz del proyecto y hacemos un comando de arranque.

```
dotnet run
```

Ese comando lo tendremos que ejecutar desde la carpeta raíz de este proyecto, donde está el archivo Program.cs.

Al ejecutarse nos saldrá en la consola el texto que hayamos indicado en el método Console.WriteLine().

```
midesweb@MacBook-Pro ~/sites/sandbox/manual-c-sharp dotnet run
Hello World!
```

Compilar la aplicación C

El otro comando de consola que vamos a ver es el que nos permite la compilación del proyecto.

```
dotnet build
```

Cuando ejecutes ese comando verás que se crea una carpeta nueva en el directorio del proyecto llamada "bin". Dentro de esa carpeta encontraremos varios archivos y carpetas, entre ellos hay un archivo "dll" que es el formato con el que se ha compilado la aplicación.

Podríamos pensar que esta compilación generase un .exe u otro tipo de archivo autónomo, que puedas ejecutar en cualquier lugar, sin embargo esto no es así. Para poder poner en marcha este tipo de compilaciones necesitas el runtime de C# instalado en la máquina donde los pienses ejecutar.

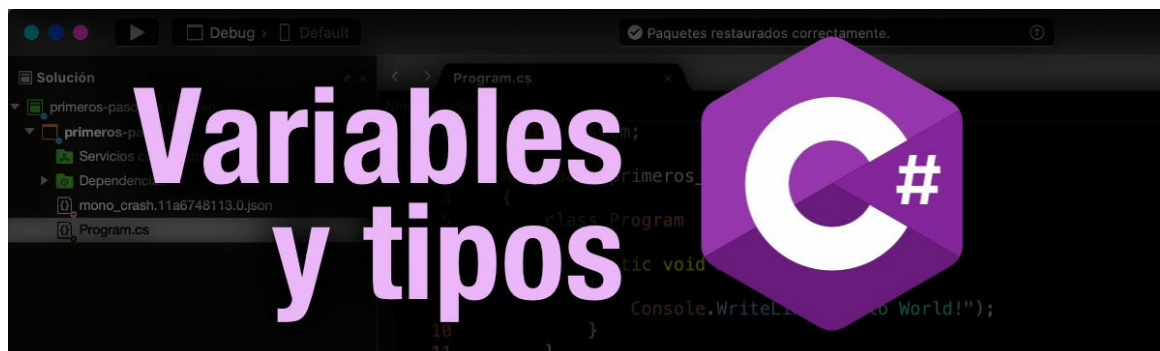
En realidad sí que hay una manera de hacer que la compilación incluya el runtime de C# en los archivos generados, lo que aumentaría considerablemente el peso de la compilación y generalmente no se suele usar, al menos para programas de consola, donde esperamos que C# (.NET) generalmente esté instalado en la máquina donde pensamos poner los programas en funcionamiento. Recuerda que actualmente .NET y C# es multiplataforma y por tanto lo puedes instalar en cualquier sistema donde lo requieras.

No obstante lo anterior, los archivos compilados en formato .dll sí que son perfectamente portables. Los puedes copiar tal cual en cualquier ordenador que tenga instalado .NET y ejecutarlos sin más.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado / actualizado en 11/06/2021
Disponible online en <https://desarrolloweb.com/articulos/primer-programa-csharp>

Variables y Tipos en C

Explicamos cómo crear variables en C#, cómo indicar el tipo de datos de las variables y las implicaciones del lenguaje, que es de tipado estático. Veremos también cuáles son los tipos de datos primitivos disponibles en C#.



C# es un lenguaje moderno, siendo una de sus principales características su tipado estático. En otras palabras podemos decir también que C Sharp es fuertemente tipado, lo que quiere decir que los tipos de datos tienen un papel protagonista a la hora de escribir programas. Por ejemplo, para crear una variable en C# necesitamos indicar el tipo de dato que va a contener. Además, una vez creada una variable con un tipo, no podremos cambiarla y asignarle un tipo distinto en el futuro.

En este artículo vamos a hablar un poco de las cuestiones alrededor de los tipos, viendo algunos ejemplos de variables y cosas que podemos y no podemos hacer.

Existe toda una familia de lenguajes de tipado dinámico muy populares, como Javascript o PHP, que seguro son el día a día de muchos otros lectores. Para ellos puede que les resulte todo un incordio esto de tener que tipar las variables, pero ya vamos a adelantar que el tipado forma toda una red de seguridad en la programación y permite al compilador alertarnos sobre muchos posibles errores que cometemos, evitando que lleguen alguna vez a producción. Dicho de otra manera, tipar las variables puede resultar un poco más de trabajo, pero es toda una ventaja para las aplicaciones profesionales.

Declaración de una variable en C

Una variable se declara en C# indicando el tipo que va a contener y luego el nombre de la variable que estamos declarando.

```
string nombre;
```

Esta es una variable de tipo "string", que consiste en una cadena de caracteres.

Si lo deseamos, podemos declarar variables y asignar el valor que contendrán, de una vez;

```
string saludo = "Hola Miguel";
```

Esta variable "saludo" es de tipo string y el valor que hemos asignado es la cadena "Hola Miguel".

Tipos primitivos en C

Se llaman tipos primitivos a los tipos de datos simples que pueden tener las variables, es decir, todos los tipos de valores básicos que podemos usar, sin entrar en estructuras más complejas como podrían ser los objetos o arrays.

En C# disponemos de los siguientes tipos de datos primitivos:

Booleanos

El booleano es el tipo que guarda un valor positivo o negativo, un sí o un no.

- bool: cuyos posibles valores puede ser true o false.

```
bool esVerdad = true;
```

Números enteros

Existen varios tipos de datos enteros, dependiendo de la capacidad. Además puedes tenerlos con y sin signo. Para los tipos de datos enteros y con signo, de menor capacidad a mayor capacidad son tenemos los siguientes:

- short: Números enteros pequeños (16 bits)
- int: Números enteros (32 bits)
- long: Números enteros grandes (64 bits)

```
int numeroEntero = -33;
```

Además, podemos tener los números enteros con signo, que serían los mismos, pero con una "u" delante:

- ushort: Números enteros sin signo pequeños (16 bits)
- uint: Números enteros sin signo (32 bits)
- ulong: Números enteros sin signo grandes (64 bits)

Como tipos de datos enteros menos usados tenemos también el byte y sbyte, que tienen solo capacidad de 8 bit.

- byte: número entero sin signo de 0 a 255.
- sbyte: número entero con signo de -128 a 127

Números con coma flotante

Los números en coma flotante son lo que llamamos comúnmente en matemáticas como enteros con decimales. Existen de tres tipos:

- float: Números con coma flotante de 32 bits
- double: Números con coma flotante de 64 bits

```
double flotanteDouble = 13.5;
```

Aquí es interesante mencionar que los literales expresados en coma flotante para almacenar el variables de tipo float deben contener una "f" al final.

```
float flotante = 13.5f;
```

Un literal en términos de lenguajes de programación es un valor escrito en el código directamente, que generalmente asignaremos a una variable o entregaremos como parámetro a una función. Es simplemente eso, un valor. Por ejemplo literal de entero sería cualquiera de los valores que podemos asignar a una variable de las de tipo entero (un número sin decimales). Un literal de string es cualquier cadena de caracteres expresada entre comillas, etc.

Estos son los tipos de datos en coma flotante más comunes, sin embargo también tenemos un tipo de datos llamado "decimal", que sería un float aporta todavía más alcance de valores (más capacidad para valores grandes) que se puede usar para cálculos financieros y monetarios.

- decimal: Números con decimales de 128 bits

Otros tipos de datos en C

Existen otros tipos de datos en C# que merece la pena comentar, comenzando por las cadenas o "string" que en C# son objetos, pero a la hora de declararlas el aspecto es idéntico a la declaración de un tipo de datos simple.

- string: una cadena de caracteres Unicode

Además tenemos también el tipo de datos char, que sirve para almacenar un único carácter Unicode, si fuera necesario.

También tenemos un tipo de datos llamado `object`, que es el tipo básico de todos los objetos, el `DateTime` que se encarga de trabajar con fechas y tiempo que veremos ejemplos si acaso más adelante. Para consultar más detalles y la lista completa puedes acceder a la [documentación de C# para tipos de datos incorporados](#).

Tipado estático

Volvemos sobre el concepto de tipado estático para ver qué cosas no podríamos hacer en el código C#, a modo de ejemplo.

Por ejemplo, si declaramos un entero, en C# no podremos colocarle una cadena con valor numérico:

```
int entero = "333"; // Esto dará un error al compilar
```

Si tenemos una variable flotante, no podremos asignarle un literal de double:

```
float conDecimales = 45.9;
```

Si una variable nace con un tipo de datos, como por ejemplo `string`, no podremos más adelante asignarle cualquier cosa que no sea del tipo `string`.

```
int num = 44;
string txt = "";
txt = num; // Esto dará un error
txt = 55; // Esto tampoco es correcto
```

Evidentemente, pueden surgir miles de situaciones en las que queramos introducir un número en una cadena y el lenguaje es perfectamente capaz, sin embargo necesitaremos realizar una conversión explícita del valor numérico a un valor cadena de caracteres.

```
int num = 44;
string txt = "";
txt = num.ToString(); // Esto sí funcionará
```

Valores nulos (null)

Los tipos simples de C# no incluyen `null` como uno de sus posibles valores. Esto es porque son tipos de datos primitivos y no tipos de datos complejos como los objetos o cadenas (recordemos que los valores de `string` son en realidad objetos aunque las creamos como si fueran valores primitivos).

Por tanto, una cadena podría tener el valor `null`:

```
string nombre = null;
```

Pero no podríamos asignar el valor null a una variable de tipo int, por ejemplo.

```
int num = null; // Esto da error
```

Conclusión

Con todo esto hemos tenido un recorrido sobre los tipos de datos en C#. Si vienes de lenguajes como Javascript habrás observado que C Sharp es mucho más rico en lo que respecta a tipos de datos. En este artículo hemos conocido los tipos de datos primitivos, o tipos de datos básicos, pero luego descubrirás que por medio de las clases puedes crear cualquier tipo de datos complejo, aumentando todavía más las posibilidades del lenguaje.

Trabajar con tipos de datos puede costar un poco al principio, pero como todo es cuestión de acostumbrarse. C# siempre te alertará en el momento de compilación sobre cualquier problema que surja con el tipado de tus variables, aportando una gran ayuda a la hora de trabajar con el lenguaje.

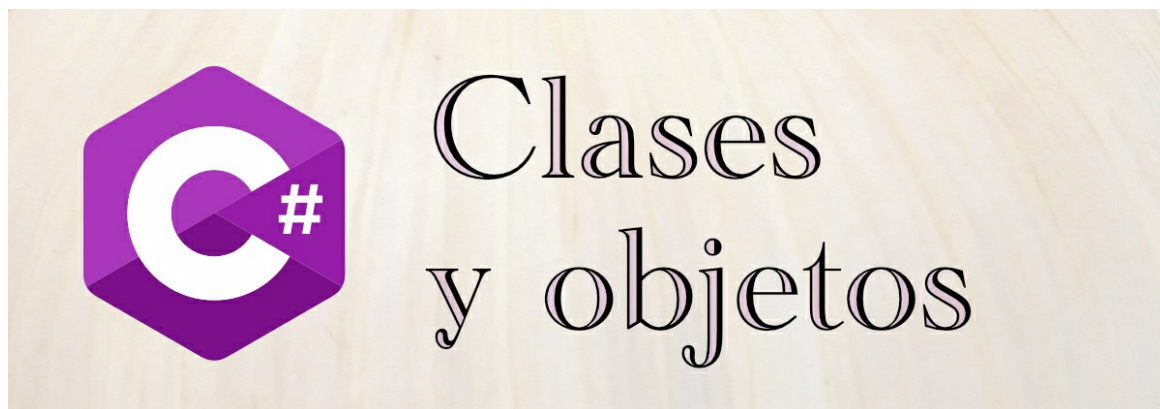
Este artículo es obra de *Miguel Angel Alvarez*

Fue publicado / actualizado en 23/07/2021

Disponible online en <https://desarrolloweb.com/articulos/variables-tipos-csharp>

Clases y objetos en C

Conceptos de programación orientada a objetos con C#. Veremos qué son las clases, los objetos, propiedades y métodos de las clases. Aprenderemos a crear constructores e instanciar objetos de una clase, con ejemplos.



C# es un lenguaje de programación orientado a objetos, por lo tanto, los programas y aplicaciones realizados con este lenguaje están compuestos por clases, a partir de las cuales se crean objetos, que colaboran entre sí para resolver los problemas o necesidades de las aplicaciones.

Dicho esto, en un programa con C#, incluso en el más básico, lo que nos vamos a encontrar son clases de programación orientada a objetos, en un número que dependerá de la complejidad de la aplicación. Unas clases más generales dependerán de otras más específicas mediante cualquier tipo de relación, como asociación, uso, composición y por supuesto herencia.

Aunque en este manual vamos a aportar bastantes nociones sobre la programación orientada a objetos no pretendemos dar un curso completo de este paradigma de la programación. Si te lías con los conceptos, eso es algo que se ha abordado con bastante detalle en el [Manual de Teoría de la Programación Orientada a Objetos](#). Si ves que no tienes suficientes conocimientos y te fallan aspectos básicos como clases y objetos te recomendamos que leas ese manual.

Nuestra primera clase en C

Realmente, incluso para los ejemplos más básicos de C# que hemos visto en el [Manual de C#](#), hemos tenido que desarrollar clases, ya que este lenguaje está completamente orientado a objetos.

El aspecto básico de una clase en C# lo podemos ver a continuación:

```
using System;

public class Program
{
    public static void Main()
    {
        Console.WriteLine("Hola Mundo!");
    }
}
```

Como puedes ver una clase se define con la palabra reservada "class". Además, se antepone la palabra "public" si queremos que esta clase se pueda acceder desde fuera de este programa.

Luego ponemos las llaves que engloban el código de la clase. Dentro de este código podemos indicar diversos miembros, como propiedades y métodos:

- Las propiedades son los datos que debe de manejar una clase
- Los métodos son las funcionalidades que podrá realizar la clase.

Método Main de las clases

El método Main en una clase de C# contiene código que se puede ejecutar al arrancar un programa. Dentro de todo lenguaje orientado a objetos existe una clase principal, que es la que se sitúa arriba en la jerarquía de clases del programa.

Esta clase principal es desde la que empieza la ejecución del programa y en ella tendremos un método Main. Al ejecutar la clase desde la consola, tal como explicamos en el artículo del [Primer programa en C#](#).

Este método ya lo conocíamos, aunque todavía no hemos desvelado todos sus detalles, como que es un método *estático*. De todos modos, nos disculparás porque va a permanecer aparcado todavía un poco más, ya que para explicarlo necesitamos conocer nuevos detalles que no hemos contado todavía.

Así que vamos a ver ahora cómo podemos hacer otras cosas, como crear propiedades y métodos adicionales.

Propiedades en las clases

En C# podemos declarar propiedades de las clases simplemente indicando su nombre y su tipo. Haremos esto generalmente al principio del código de la clase.

```
public class Program
{
    int numero;
    string cadena;
}
```


En este código hemos definido dos propiedades en la clase Program. La primera se llama "numero" y es de tipo entero (int) y la segunda se llama "cadena" y es de tipo string.

Además a las propiedades les podemos asignar una visibilidad, indicando si son públicas, privadas, etc. Eso lo veremos también más adelante.

Métodos de las clases

Además de las propiedades vamos a aprender a definir métodos en las clases. Los métodos son como funciones que pertenecen a las clases y que tienen el código que implementa las funcionalidades que esta clase debe desempeñar.

Para definir métodos usamos un código como el siguiente:

```
public class Program
{
    int numero;
    string cadena;

    void show() {
        Console.WriteLine(this.numero);
        Console.WriteLine(this.cadena);
    }
}
```

En este caso hemos definido un método llamado "show". Se antepone la palabra "void" porque en C# debemos indicar el tipo de datos que devuelve todo método. En caso que no devuelva ningún dato entonces indicamos "void" como tipo de devolución.

Luego colocamos los paréntesis y opcionalmente parámetros que el método necesite recibir en su invocación. En este caso nuestro método no recibe ningún parámetro, por lo que colocamos los paréntesis vacíos.

En el caso de los métodos también podemos indicar la visibilidad, como public o private. Lo explicaremos con detalle más adelante.

Variable this

Los métodos se ejecutan normalmente en el contexto del objeto sobre el que se ha invocado un método. Aún no hemos creado ningún objeto por lo que puede costar algo de ver. Enseguida lo haremos y quedará más claro, pero de momento tienes que saber:

- Las clases son definiciones, como un mapa o un libro que define las características de los objetos que está modelando.

- Los objetos son concreciones de un ejemplar de una clase.

Para invocar un método primero necesitamos crear un objeto de una clase. Por ejemplo tenemos la clase "Coche". El coche tiene un método que se llama "arrancar". Para poder ejecutar ese método necesitamos crear primero un objeto de la clase "Coche".

Solo para que lo sepas y por si lo vamos usando más adelante en este manual, en la jerga de la programación orientada a objetos, invocar un método sobre un objeto se denomina "lanzar un mensaje".

Enseguida veremos cómo crear objetos. pero supongamos que hemos creado un coche llamado "TeslaNegro". Si yo le pido "arrancar" al "TeslaNegro", entonces el objeto que arrancará será justamente aquel sobre el que he invocado el método "arrancar".

Pues bien, dentro de los métodos tenemos la palabra "this" para referirnos al objeto que recibió ese método. Por tanto, si le pedí arrancar al "Tesla negro", entonces `this` contendrá una referencia justamente a ese objeto Coche.

Dicho esto, en el método que acabamos de ver, usábamos la variable `this` para acceder a las propiedades del objeto que recibieron el método.

```
void show() {  
    Console.WriteLine(this.numero);  
    Console.WriteLine(this.cadena);  
}
```

Métodos constructores

Ahora vamos a aprender a crear constructores de los objetos. Los constructores, a pesar de su nombre, no construyen nada. En realidad ellos se dedican a inicializar aquellas propiedades que deban ser inicializadas durante la construcción de los objetos.

En C# los constructores tienen el mismo nombre que se otorgó a la clase. Por ejemplo aquí podríamos tener un constructor de la clase `Program`.

```
Program(int numero, string cadena) {  
    this.numero = numero;  
    this.cadena = cadena;  
}
```

Los constructores son como otros métodos de la clase, pero como puedes apreciar son algo especiales, porque no se indica ningún valor de devolución en ellos.

En los constructores podemos recibir los parámetros que sean necesarios para inicializar los objetos. Estos parámetros pueden ser habitualmente los valores de las propiedades que deseamos inicializar, pero no tiene por qué ser siempre así.

En los constructores usamos `this` para referirnos al objeto que se está inicializando en ese momento dado. En nuestro constructor simplemente asignamos los valores de los parámetros recibidos a las propiedades del objeto que se está inicializando.

Cómo instanciar objetos

En Programación Orientada a Objetos usamos el verbo "instanciar" para referirnos al proceso mediante el cual se crea un objeto a partir de una clase. También decimos que tal objeto es una "instancia" de tal clase.

En C# instanciamos objetos con la palabra reservada "new", indicando a continuación el nombre de la clase del objeto que deseamos crear y luego los paréntesis, indicando en ellos los argumentos requeridos por el constructor de la clase.

```
Program objetoProgram = new Program(3, "hola");
```

En este caso hemos declarado una variable "objetoProgram" que es de tipo "Program" y se le ha asignado una nueva instancia de la clase "Program". Al crearse este objeto se invocará al constructor para realizar las tareas de inicialización y se le pasarán los parámetros que ese constructor requiere, en nuestro caso era un entero y una cadena.

Una vez creado un objeto, podemos invocar a sus métodos.

```
objetoProgram.show()
```

Eso hará que se ejecute el método `show` sobre el objeto `objetoProgram`, que acabamos de crear.

Programa completo de una clase con atributos y métodos

Con lo que sabemos ya podemos ver el siguiente código completo de un programa en C# que contiene las propiedades y métodos que hemos visto antes.

```
using System;

public class Program
{
    int numero;
    string cadena;

    public static void Main()
    {
        Program objetoProgram = new Program(3, "hola");
        objetoProgram.show();
    }
}
```

```
Program(int numero, string cadena) {  
    this.numero = numero;  
    this.cadena = cadena;  
}  
  
void show() {  
    Console.WriteLine(this.numero);  
    Console.WriteLine(this.cadena);  
}  
}
```

Como puedes ver, en nuestro programa, que es muy simple, las tareas de creación de los objetos se realizan dentro del método `Main()`. Como habíamos dicho, `Main` se usa para arrancar el programa y se encarga generalmente de instanciar un objeto de programa y pedirle que realice alguna acción que produzca la ejecución de la aplicación.

En este caso esa acción era crear un objeto y pedirle que muestre los valores de sus propiedades con el método `show()`.

Hasta aquí hemos dado un primer paseo por las cosas más básicas sobre clases y objetos en C#. Esperamos que te hayan resultado sencillas de entender y puedas experimentar algo más modificando el código de esta clase. En futuros artículos realizaremos cosas más interesantes y aprenderemos otros conceptos de programación orientada a objetos y su aplicación en C#.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado / actualizado en 01/03/2023
Disponible online en <https://desarrolloweb.com/articulos/clases-objetos-c-sharp>