

# Horarios

## Teoría:

- Martes de 16:00 a 18:00

## Práctica

- Martes de 18:00 a 19:00

Reuniones  
virtuales

Además contamos con un foro en [Ideas](#) para consultas de práctica y otro para consultas de teoría (disponible 24/7)

# Fechas importantes

30/11/2021: Examen primera fecha

7/12/2021: Examen segunda fecha

17/12/2021: Examen tercera fecha

Los coloquios sobre los trabajos de codificación obligatorios se tomarán en el período entre las fechas 9/11/2021 y 7/12/2021

Para este curso vamos a necesitar descargar e instalar ...



Descargar e instalar la última versión estable  
de .NET

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

## Supported versions

Version	Status	Latest release	Latest release date	End of support
<a href="#">.NET 6.0</a>	Preview ⓘ	6.0.0-preview.6	July 14, 2021	
<a href="#">.NET 5.0 (recommended)</a>	Current ⓘ	5.0.8	July 13, 2021	
<a href="#">.NET Core 3.1</a>	LTS ⓘ	3.1.17	July 13, 2021	December 03, 2022
<a href="#">.NET Core 2.1</a>	LTS ⓘ	2.1.28	May 11, 2021	August 21, 2021

Para este curso vamos a necesitar descargar e instalar ...



# Descargar e instalar Visual Studio Code (<https://code.visualstudio.com/>)

The screenshot shows the official Visual Studio Code website. At the top, there's a navigation bar with links to 'Visual Studio Code', 'Docs', 'Updates', 'Blog', 'API', 'Extensions', and 'FAQ'. To the right of the navigation is a search bar labeled 'Search Docs' and a blue 'Download' button. A message at the top states 'Version 1.48 is now available! Read about the new features and fixes from July.' Below this, a large white text on a dark background reads 'Code editing. Redefined.' followed by 'Free. Built on open source. Runs everywhere.' On the left, there's a download section for Windows, macOS, and Linux. The Windows section is highlighted with a blue box around the 'Download for Windows' button. It shows 'Stable Build' and 'Insiders' options. The macOS section shows 'Package' download links. The Linux section shows '.deb' and '.rpm' download links. Below these are 'Other downloads' links. The main content area on the right displays the Visual Studio Code interface, including the Extensions Marketplace, a code editor with several files (App.js, index.js, serviceWorker.js), and a terminal window. The bottom status bar shows 'master', '0 0 ▲ 0', and file details like 'Ln 43, Col 19, Spaces: 2, UTF-8, LF, JavaScript'.



# .NET

## Teoría 1

# Generalidades de .NET

## Generalidades de .NET

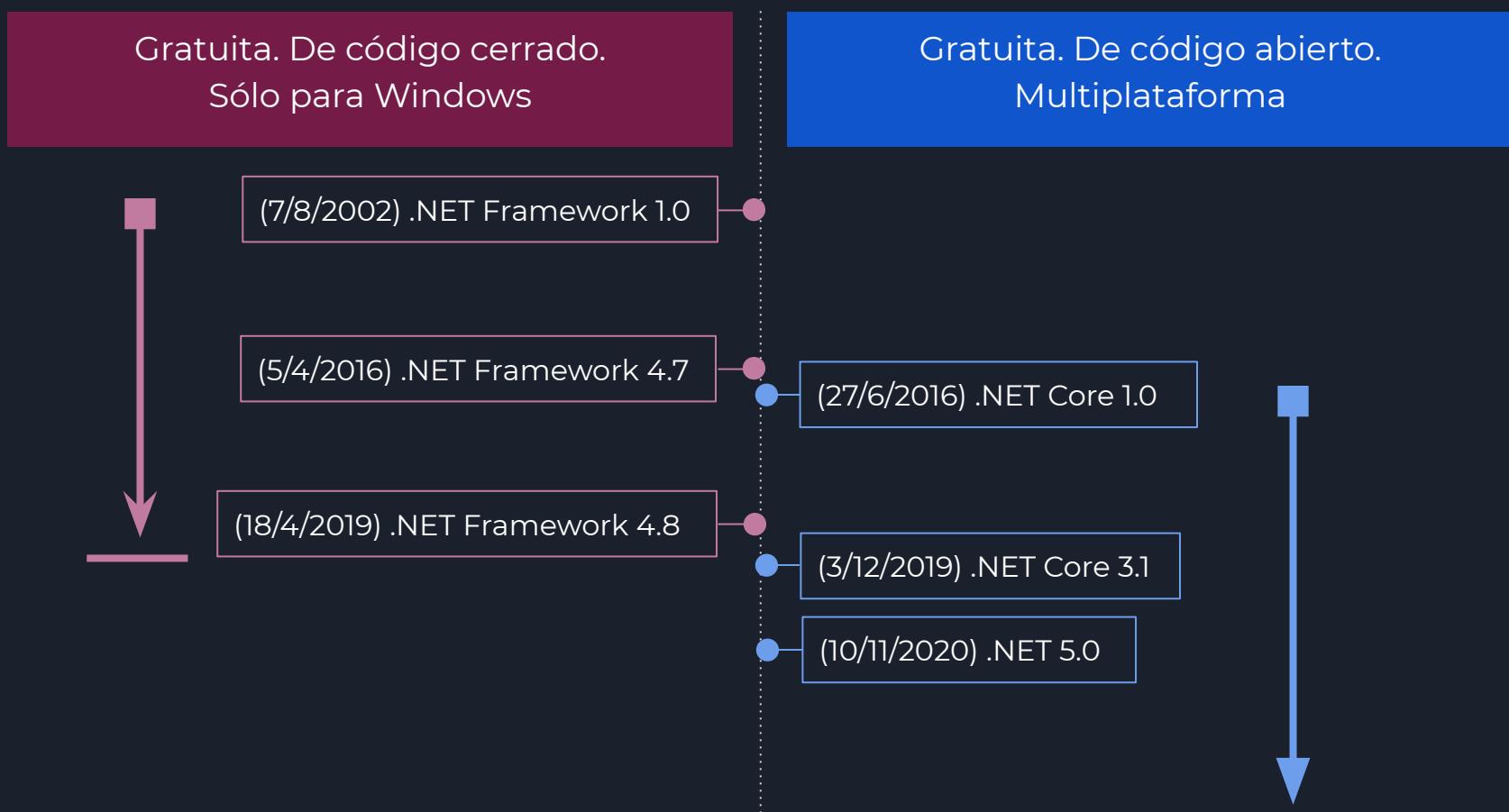
- Actualmente .NET es una plataforma de desarrollo gratuita, de código abierto y de uso general

### Tipos de aplicaciones que se pueden construir con .NET

 <b>Web</b> Build web apps and services for Linux, Windows, macOS, and Docker.	 <b>Mobile</b> Use a single codebase to build native mobile apps for iOS, Android, and Windows.	 <b>Desktop</b> Create beautiful and compelling desktop apps for Windows and macOS.	 <b>Microservices</b> Create independently deployable microservices that run on Docker containers.
 <b>Cloud</b> Consume existing cloud services, or create and deploy your own.	 <b>Machine Learning</b> Add vision algorithms, speech processing, predictive models, and more to your apps.	 <b>Game Development</b> Develop 2D and 3D games for the most popular desktops, phones, and consoles.	 <b>Internet of Things</b> Make IoT apps, with native support for the Raspberry Pi and other single-board computers.

## Generalidades de .NET

### Evolución



# Generalidades de .NET

- Soporta varios lenguajes de programación. Microsoft desarrolla activamente C#, Visual Basic y F#, pero también existen otros.
- Actualmente existen 4 implementaciones distintas de .NET, pero está en marcha el proyecto de unificarlas

# Implementaciones de .NET

1. **.NET Framework**: es la implementación original, optimizado para aplicaciones de escritorio de Windows.
2. **Mono**: para entornos de ejecución pequeños (**Xamarin** en **Android**, **macOS**, **iOS**, **tvOS** y **watchOS**) y juegos de **Unity**.
3. **Plataforma Universal de Windows (UWP)**: para aplicaciones **Windows** y de Internet de las cosas (**IoT**).
4. **.Net Core**: es una implementación multiplataforma de **.NET** (ahora se denomina simplemente **.NET**)

# Un poco de historia Todo comenzó con .NET Framework

- La versión 1.0 de .NET Framework fue lanzada oficialmente por Microsoft para su sistema operativo Windows en enero del 2002.
- Junto con .NET Framework, Microsoft liberó la primera versión del lenguaje C# y una nueva versión de Visual Basic.
- F# es un lenguaje funcional más reciente, fue desarrollado por Microsoft en 2005.
- .NET Framework es gratuito pero no es *open source*. La última versión es la 4.8 (última actualización abril de 2019)

# .NET Core y su evolución a .Net 5 Microsoft y el open source

- La versión **multiplataforma** (Windows, macOS y Linux) y **open-source** de .NET se inició con el nombre de **.NET Core** (v 1.0 liberado en junio 2016)
- La última versión con el nombre **.NET Core** es la 3.1 (liberado en diciembre 2019). A partir de noviembre de 2020 cambió su nombre a **.NET 5.0**
- La única rama que seguirá evolucionando en el futuro es la que se inicia con **.NET 5.0**

.NET Core 3.1 evolucionó a .NET 5.0

.NET 5.0 NO es la  
continuación de .NET  
Framework 4.8 sino la  
continuación de  
.NET Core 3.1

### .NET Core 3.1 evolucionó a .NET 5.0

.NET 5.0 pudo haberse llamado .NET Core 5.0 pero se prefirió cambiar el nombre a .NET 5.0 para dar idea de unificación.

En el futuro sólo se hablará de .NET

## Hoja de ruta de .NET



- **LTS = *Long Term Support*** (Soporte a largo plazo)
- **GA = *General Availability*** (Disponibilidad general)

# Common Language Runtime (CLR)

- El CLR es el **motor de ejecución (runtime)** de .NET
- Es un **entorno virtual** independiente de la arquitectura de hardware en el que se **ejecutan las aplicaciones** escritas con cualquier lenguaje .NET

# Microsoft Intermediate Language (MSIL)

Cuando se compila una aplicación escrita en un lenguaje de .NET (VB, C# u otro de los soportados), el compilador genera código MSIL (también conocido como código CIL).

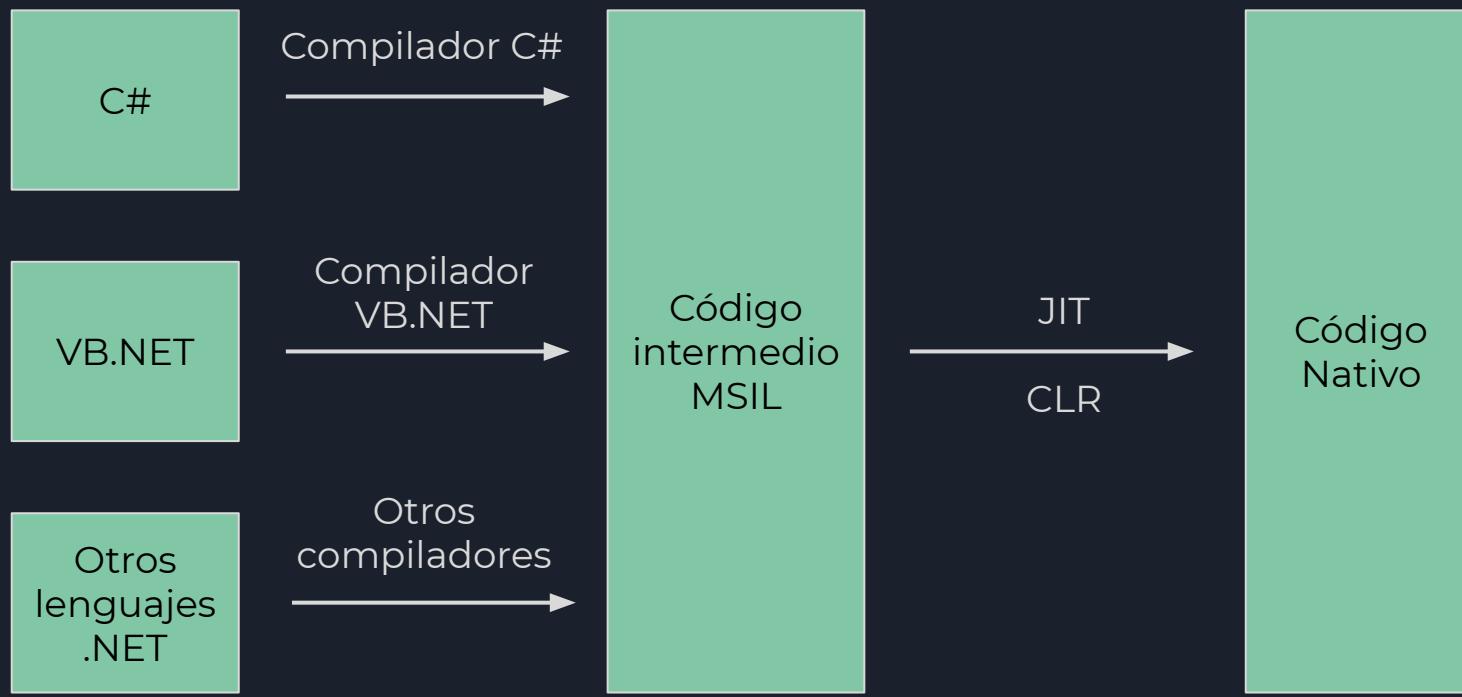
MSIL es un lenguaje similar a un código ensamblador pero de más alto nivel, creado para un hipotético procesador virtual que no está atado a una arquitectura determinada.

# Microsoft Intermediate Language (MSIL)

De manera transparente el código MSIL se traduce al lenguaje nativo del procesador físico en tiempo de ejecución, por intermediación de un compilador bajo demanda “Just In Time” (JIT)

# Microsoft Intermediate Language (MSIL)

Código Fuente



Tiempo de compilación

Tiempo de ejecución

# Base Classes Library (BCL)

- La **BCL** es la biblioteca de clases base de **.NET** que da soporte a infinidad de funcionalidades a través de las **clases** y otros **tipos** definidos en ella.
- Las clases en la **BCL** se organizan en espacios de nombres (**namespaces**) que agrupan a clases y a otros **namespaces**. La **BCL** incluye los tipos definidos en los espacios **System.\*** y **Microsoft.\***

# Base Classes Library (BCL)

- Por ejemplo, los **tipos integrados** que veremos más adelante en esta teoría están en el espacio de nombres **System**
- En **System.Collections** (namespace **Collections** dentro del namespace **System**) encontramos clases que representan colecciones
- Las clases relacionadas con el manejo de estructuras de datos **XML** se encuentra en el espacio de nombres **System.Xml** (namespace **Xml** dentro del namespace **System**)
- Las clases para manejar la comunicación entre servidor web y cliente están en el espacio de nombres **System.Web**

# Algunos namespaces de la BCL

Nombre	Descripción
System	El espacio de nombres contiene clases fundamentales y clases base que definen tipos de datos de valor y de referencia usados comúnmente, eventos y controladores de eventos, interfaces, atributos y excepciones de procesamiento.
System.Web	El espacio de nombres proporciona clases e interfaces que permiten la comunicación entre el explorador y el servidor.
System.Media	El espacio de nombres contiene clases para reproducir archivos de sonido y obtener acceso a los sonidos que proporciona el sistema.
System.Printing	Proporciona clases que permiten automatizar la administración de servidores, colas y trabajos de impresión.
System.Xml	El espacio de nombres proporciona compatibilidad basada en estándares para procesar XML.
System.Data	El espacio de nombres proporciona acceso a las clases que representan la arquitectura de ADO.NET. ADO.NET permite crear componentes que administran datos de varios orígenes de datos con eficacia.
System.Collections	El espacio de nombres contiene interfaces y clases que definen varias colecciones de objetos, como listas, colas, matrices de bits, tablas hash y diccionarios.
System.Text	El espacio de nombres contiene clases que representan codificaciones de caracteres ASCII y Unicode; clases base abstractas para convertir bloques de caracteres a y desde bloques de bytes y una clase del asistente que manipula y da formato a objetos sin crear instancias intermedias de .

# Introducción a C#



## Abrir una terminal (consola) del sistema operativo y ...

The image shows a computer monitor with a terminal window titled "leo : bash — Konsole". The window has a menu bar with "Archivo", "Editar", "Ver", "Marcadores", "Preferencias", and "Ayuda". The command "mkdir proyectosDotnet" is typed into the terminal, with a blue selection bar highlighting it. A white arrow points from a text box below the terminal up towards the selected command. The text box contains the Spanish note: "Crear el directorio proyectosDotnet". To the right of the monitor, a person's hands are shown typing on a white keyboard.

leo : bash — Konsole

Archivo Editar Ver Marcadores Preferencias Ayuda

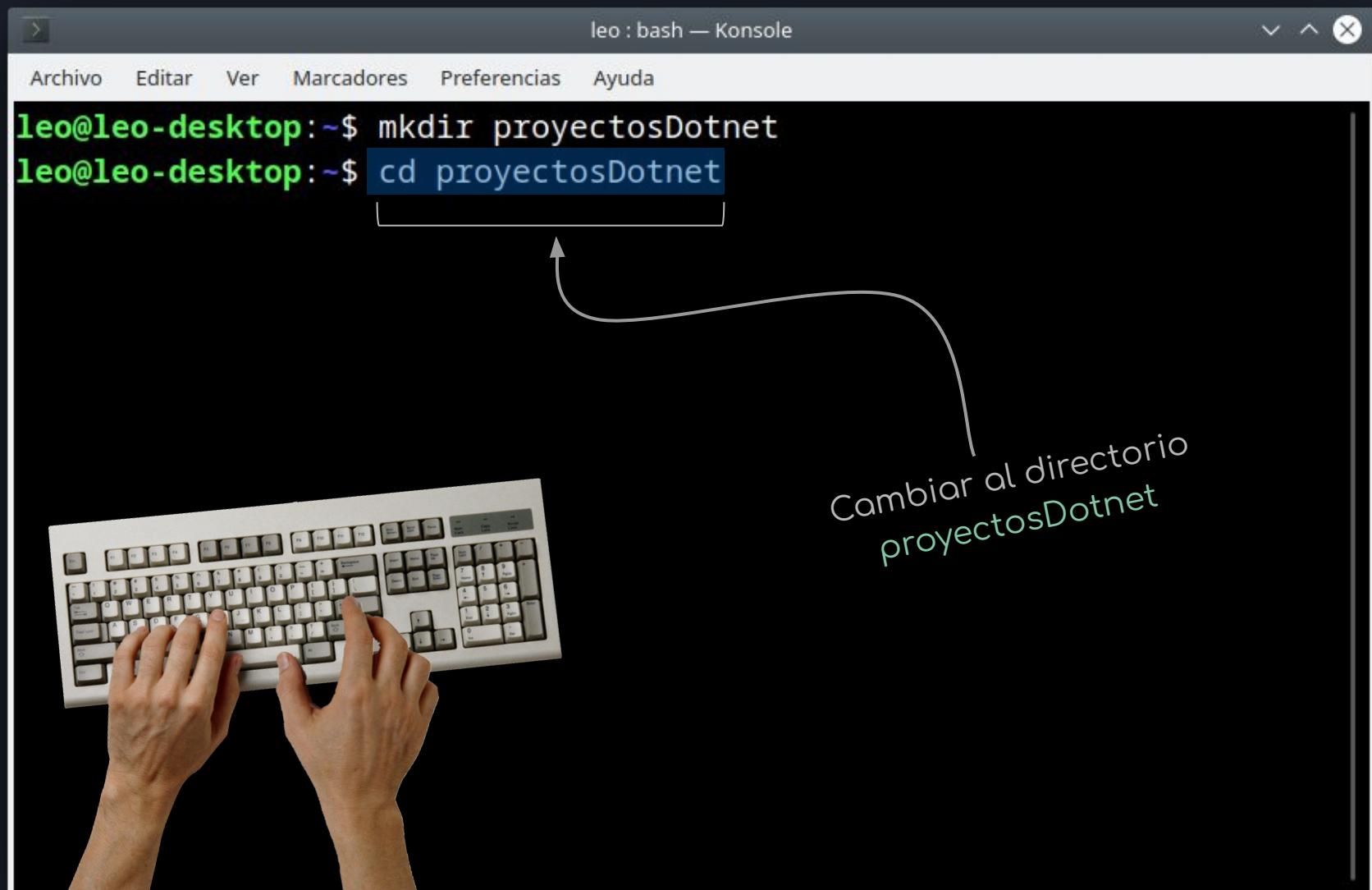
leo@leo-desktop:~\$ mkdir proyectosDotnet

Crear el directorio  
proyectosDotnet

NOTA: Los términos  
carpeta y directorio  
son sinónimos en  
este contexto



## Abrir una terminal (consola) del sistema operativo y ...



## Command-line interface (CLI)

A cartoon illustration of a teacher with brown hair and red-rimmed glasses, wearing a brown sweater over a white collared shirt. He is pointing a red pointer towards a chalkboard with his right hand. The chalkboard has a wooden frame and contains text.

Para crear nuestra aplicación vamos a utilizar .NET CLI

La interfaz de la línea de comandos de .NET es un conjunto de herramientas multiplataforma que sirve para desarrollar, compilar, ejecutar y publicar aplicaciones .NET.

Todos los comandos comienzan con dotnet que es el controlador genérico de la CLI de .NET.



## Abrir una terminal (consola) del sistema operativo y ...

The image shows a terminal window titled "proyectosDotnet : bash — Konsole". The window contains the following command history:

```
leo@leo-desktop:~$ mkdir proyectosDotnet
leo@leo-desktop:~$ cd proyectosDotnet
leo@leo-desktop:~/proyectosDotnet$ dotnet new console -o teoria1
```

A blue rectangular callout box highlights the last command: "dotnet new console -o teoria1". A white arrow points from the text "Crear la aplicación de consola teoria1" (Create the console application teoria1) up towards this highlighted command.

Crear la aplicación de consola teoria1

A photograph of a person's hands typing on a white computer keyboard. The hands are positioned as if they are entering the command shown in the terminal window above.

## Comando dotnet

```
leo@leo-desktop:~$ mkdir proyectosDotnet
leo@leo-desktop:~$ cd proyectosDotnet
leo@leo-desktop:~/proyectosDotnet$ dotnet new console -o teoria1
```

**NOTA:** Para ver los distintos tipos de proyectos que se pueden crear se debe tipar:

**dotnet new --list**  
o simplemente  
**dotnet new**

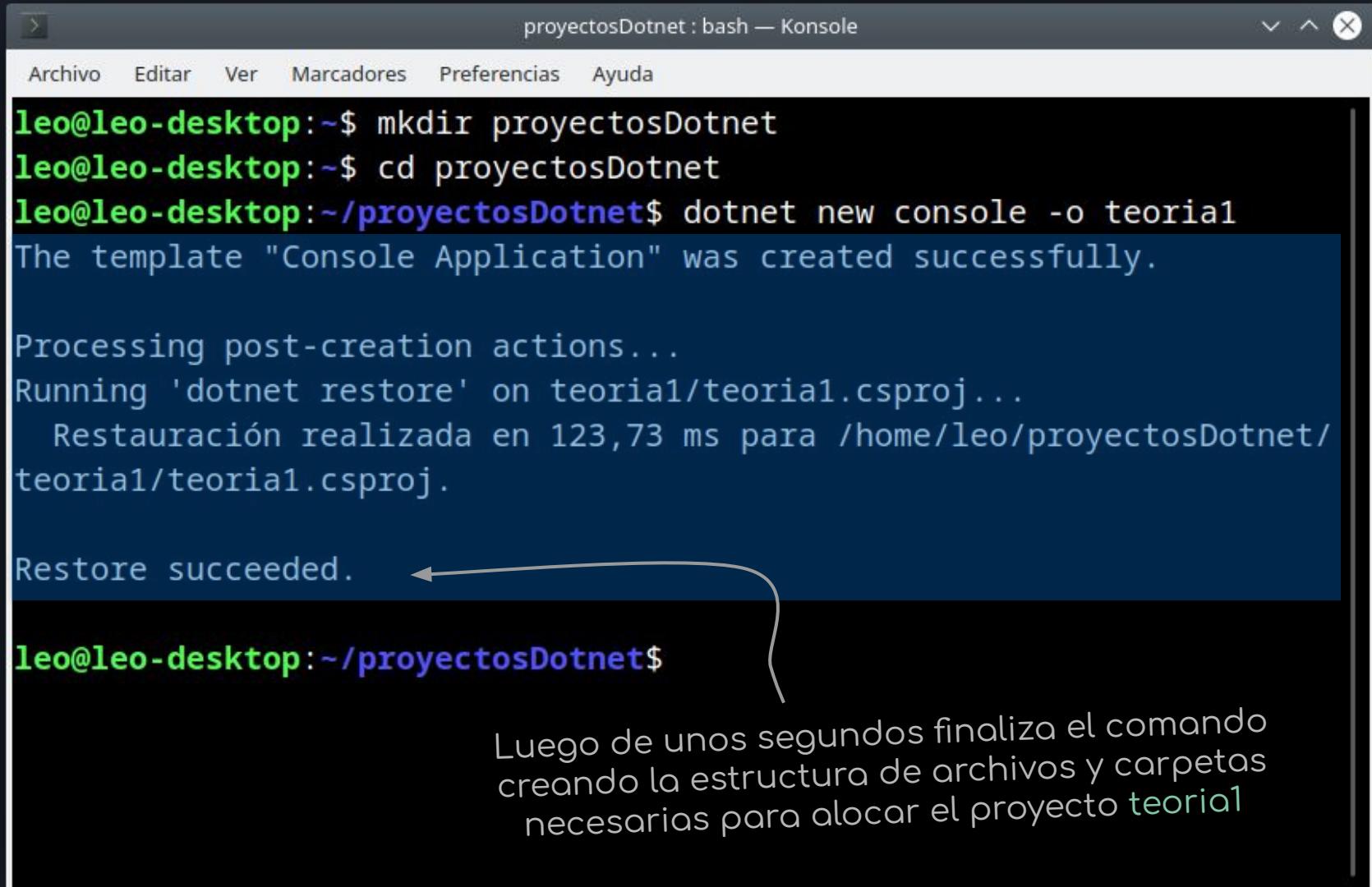
comando para  
crear un  
proyecto basado  
en una plantilla

Nombre de la plantilla  
para crear una  
aplicación de consola

-o (output)  
establece la carpeta (se  
crea si es necesario)  
donde se genera el  
proyecto



## Comando dotnet



```
leo@leo-desktop:~$ mkdir proyectosDotnet
leo@leo-desktop:~$ cd proyectosDotnet
leo@leo-desktop:~/proyectosDotnet$ dotnet new console -o teoria1
The template "Console Application" was created successfully.

Processing post-creation actions...
Running 'dotnet restore' on teoria1/teoria1.csproj...
  Restauración realizada en 123,73 ms para /home/leo/proyectosDotnet/
  teoria1/teoria1.csproj.

Restore succeeded.

leo@leo-desktop:~/proyectosDotnet$
```

Luego de unos segundos finaliza el comando creando la estructura de archivos y carpetas necesarias para alojar el proyecto `teoria1`



En la terminal del sistema operativo hacer:

```
Processing post-creation actions...
Running 'dotnet restore' on teoria1/teoria1.csproj...
Restauración realizada en 123,73 ms para /home/leo/proyectosDotnet/teoria1/teoria1.csproj.
```

```
Restore succeeded.
```

```
leo@leo-desktop:~/proyectosDotnet$ cd teoria1
leo@leo-desktop:~/proyectosDotnet/teoria1$ ls
```

Cambiar al  
directorio teoria1 ...

... y listar los  
archivos  
generados

En Windows (Símbolo  
del Sistema) se puede  
usar el comando **dir** en  
lugar de **ls**  
Windows PowerShell  
soporta el comando **ls**

### Contenido del proyecto generado por el comando dotnet new

```
Processing post-creation actions...
Running 'dotnet restore' on teoria1/teoria1.csproj...
Restauración realizada en 123,73 ms para /home/leo/proyectosDotnet/
teoria1/teoria1.csproj.

Restore succeeded.
```

```
leo@leo-desktop:~/proyectosDotnet$ cd teoria1
leo@leo-desktop:~/proyectosDotnet/teoria1$ ls
obj  Program.cs  teoria1.csproj
leo@leo-desktop:~/proyectosDotnet/teoria1$
```

En el archivo  
Program.cs hay  
código C Sharp



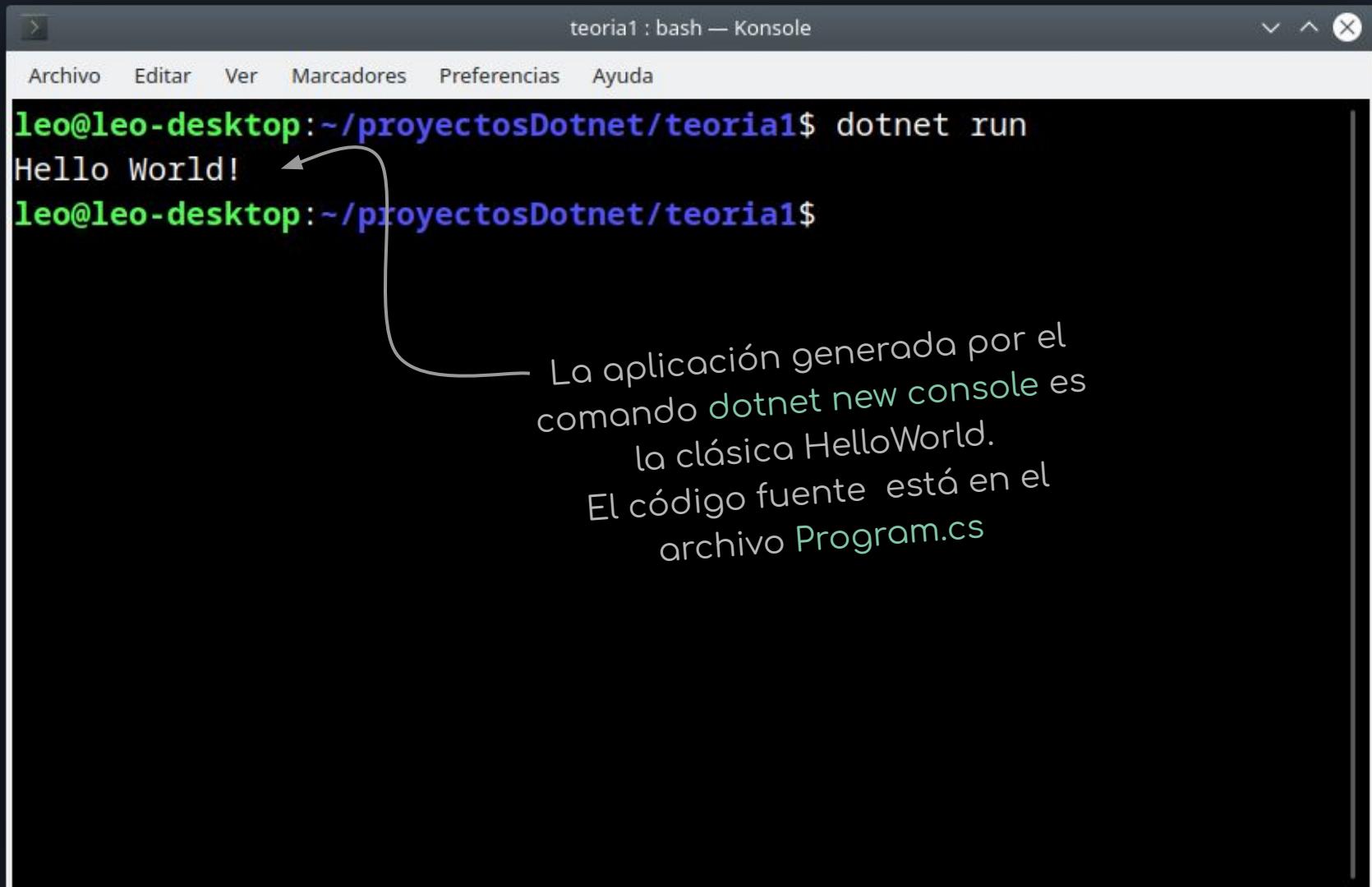
En la terminal del sistema operativo hacer:



```
teoria1 : bash — Konsole
Archivo Editar Ver Marcadores Preferencias Ayuda
leo@leo-desktop:~/proyectosDotnet/teoria1$ dotnet run
```

Compilar y ejecutar la aplicación con el comando `dotnet run`  
Este comando asume que el proyecto está en la carpeta actual

## Ejecución de la aplicación



The screenshot shows a terminal window titled "teoria1 : bash — Konsole". The window has a menu bar with "Archivo", "Editar", "Ver", "Marcadores", "Preferencias", and "Ayuda". The terminal output is as follows:

```
leo@leo-desktop:~/proyectosDotnet/teoria1$ dotnet run
Hello World!
leo@leo-desktop:~/proyectosDotnet/teoria1$
```

A curly brace is drawn around the command "dotnet run" and its output "Hello World!". A callout bubble points from this brace to the explanatory text below.

La aplicación generada por el comando `dotnet new console` es la clásica `HelloWorld`. El código fuente está en el archivo `Program.cs`



## Descargar e instalar Visual Studio Code (<https://code.visualstudio.com/>)

The screenshot shows the official Visual Studio Code website. At the top, there's a navigation bar with links to 'Visual Studio Code', 'Docs', 'Updates', 'Blog', 'API', 'Extensions', and 'FAQ'. To the right of the navigation is a search bar labeled 'Search Docs' and a blue 'Download' button. A message at the top of the main content area says 'Version 1.48 is now available! Read about the new features and fixes from July.' Below this, there's a large heading 'Code editing. Redefined.' followed by the text 'Free. Built on open source. Runs everywhere.' On the left, there's a download section for Windows, macOS, and Linux, with options for 'Stable Build' and 'Insiders'. The download links are for User Installers or Packages. On the right, a screenshot of the VS Code interface shows an open file named 'serviceWorker.js' with some JavaScript code. The interface includes tabs for 'App.js', 'index.js', and 'serviceWorker.js'. The code editor has several lines of code, including comments and function definitions. Below the code editor, there's a terminal window showing a command prompt and some output. At the bottom, there's a status bar with information like 'Local: http://localhost:3000/' and 'On Your Network: http://10.211.55.3:3000/'. A note at the bottom right says 'Note that the development build is not optimized.'



En la terminal del sistema operativo hacer:

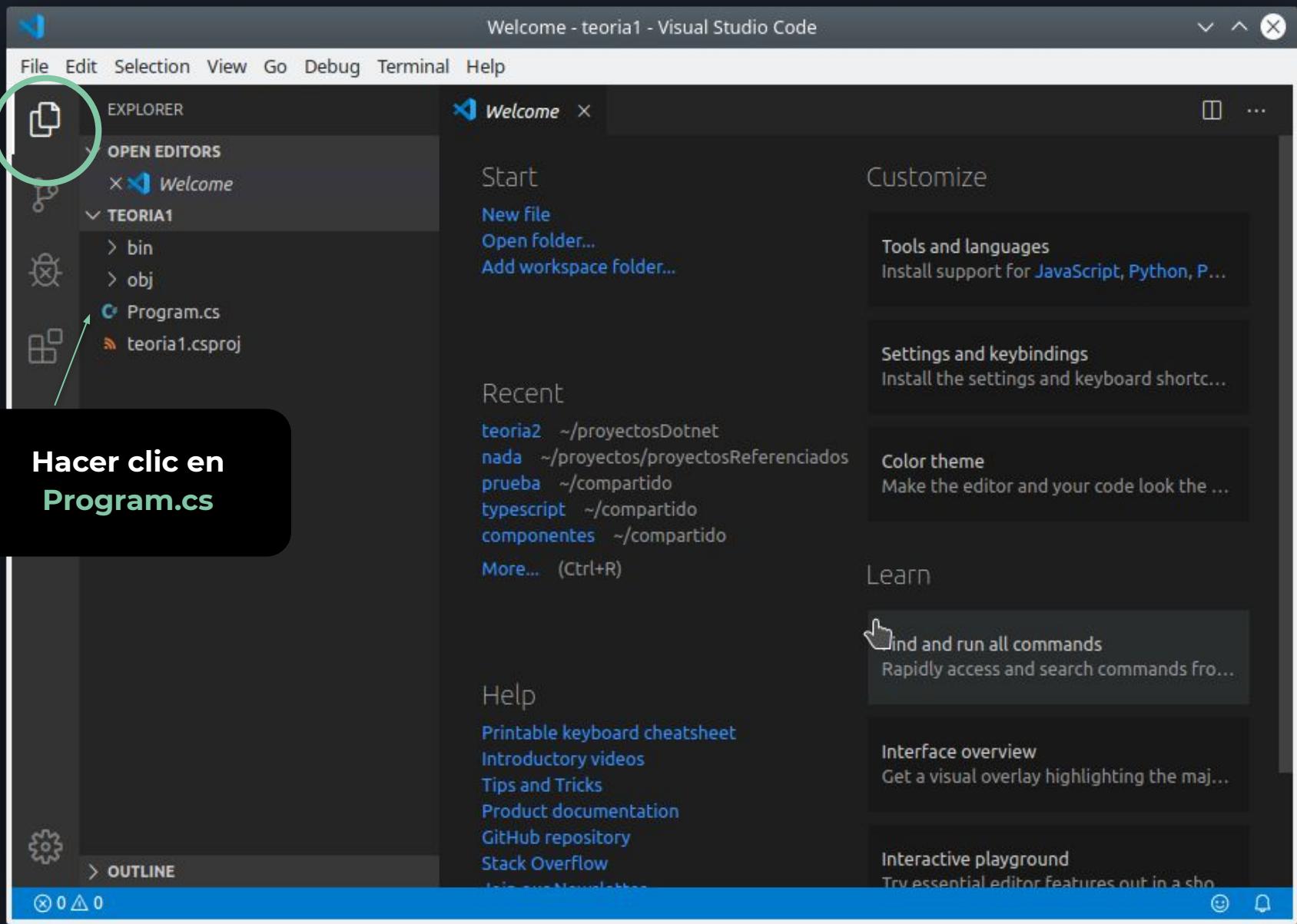


```
teoria1 : bash — Konsole
Archivo Editar Ver Marcadores Preferencias Ayuda
leo@leo-desktop:~/proyectosDotnet/teoria1$ dotnet run
Hello World!
leo@leo-desktop:~/proyectosDotnet/teoria1$ code .
```

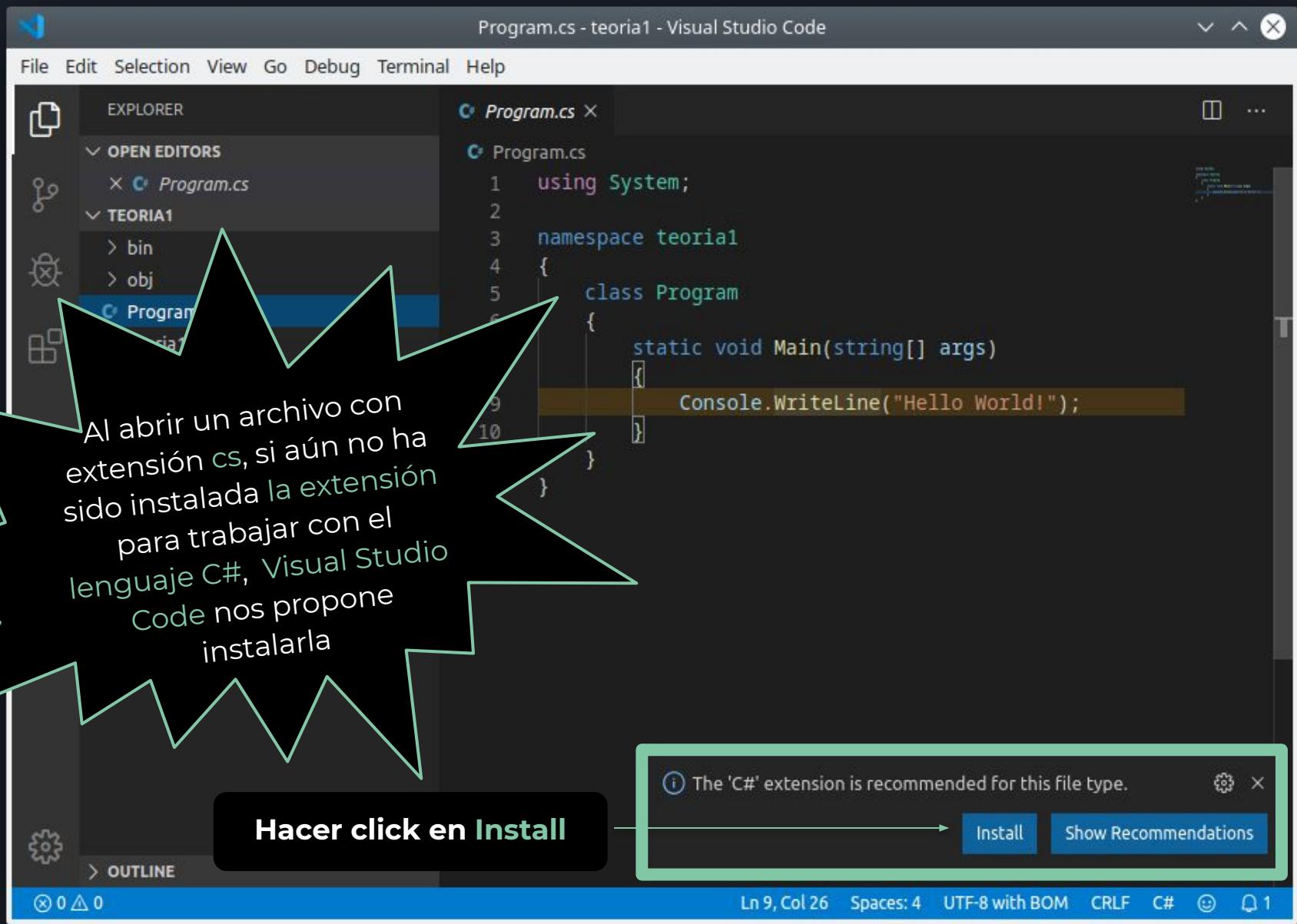
El punto (.) indica la carpeta actual

Abrir Visual Studio Code en la carpeta actual

# Introducción a C# - Primera aplicación en C#



# Introducción a C# - Primera aplicación en C#



# Introducción a C# - Primera aplicación en C#

The screenshot shows the Visual Studio Code interface with the title bar "Program.cs - teoria1 - Visual Studio Code". The menu bar includes File, Edit, Selection, View, Go, Debug, Terminal, and Help. In the top left, there's a "EXTENSIONS: MARKETPLACE" button. A tooltip box with rounded corners contains the text: "Luego de unos segundos se habrá completado la instalación de la extensión C# for Visual Studio Code". An arrow points from this tooltip to a green circle around the "C# 1.21.9" extension entry in the Marketplace sidebar. The main editor area shows a "Program.cs" file with the following code:

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

The status bar at the bottom shows "Ln 1, Col 14" and "Spaces: 4". The bottom right corner of the status bar has a small icon with the number "1".

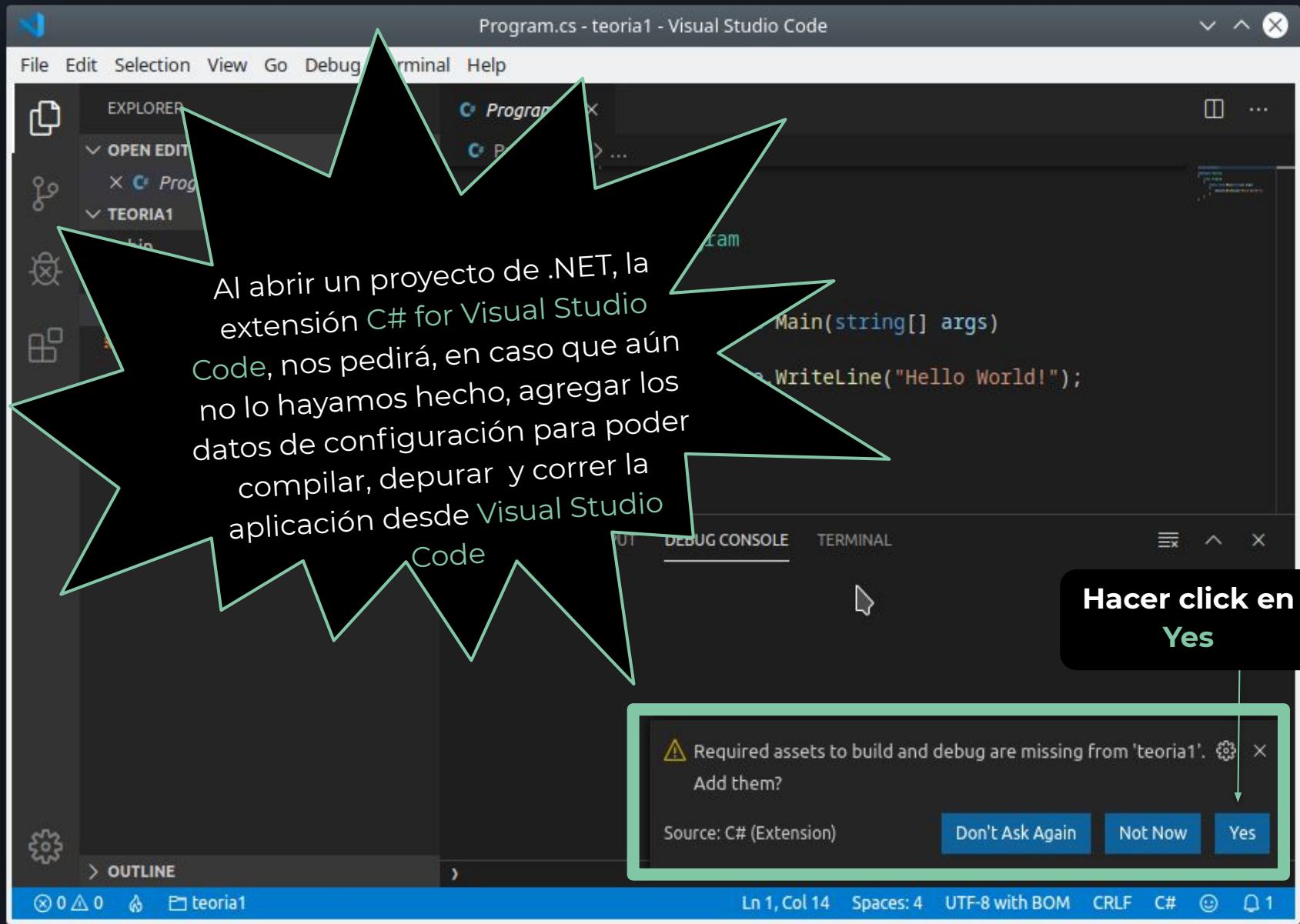
PROBLEMS    OUTPUT    DEBUG CONSOLE    ...    C#   

```
..... Done!
Validating download...
Integrity Check succeeded.
Installing package '.NET Core Debugger (linux / x64)'

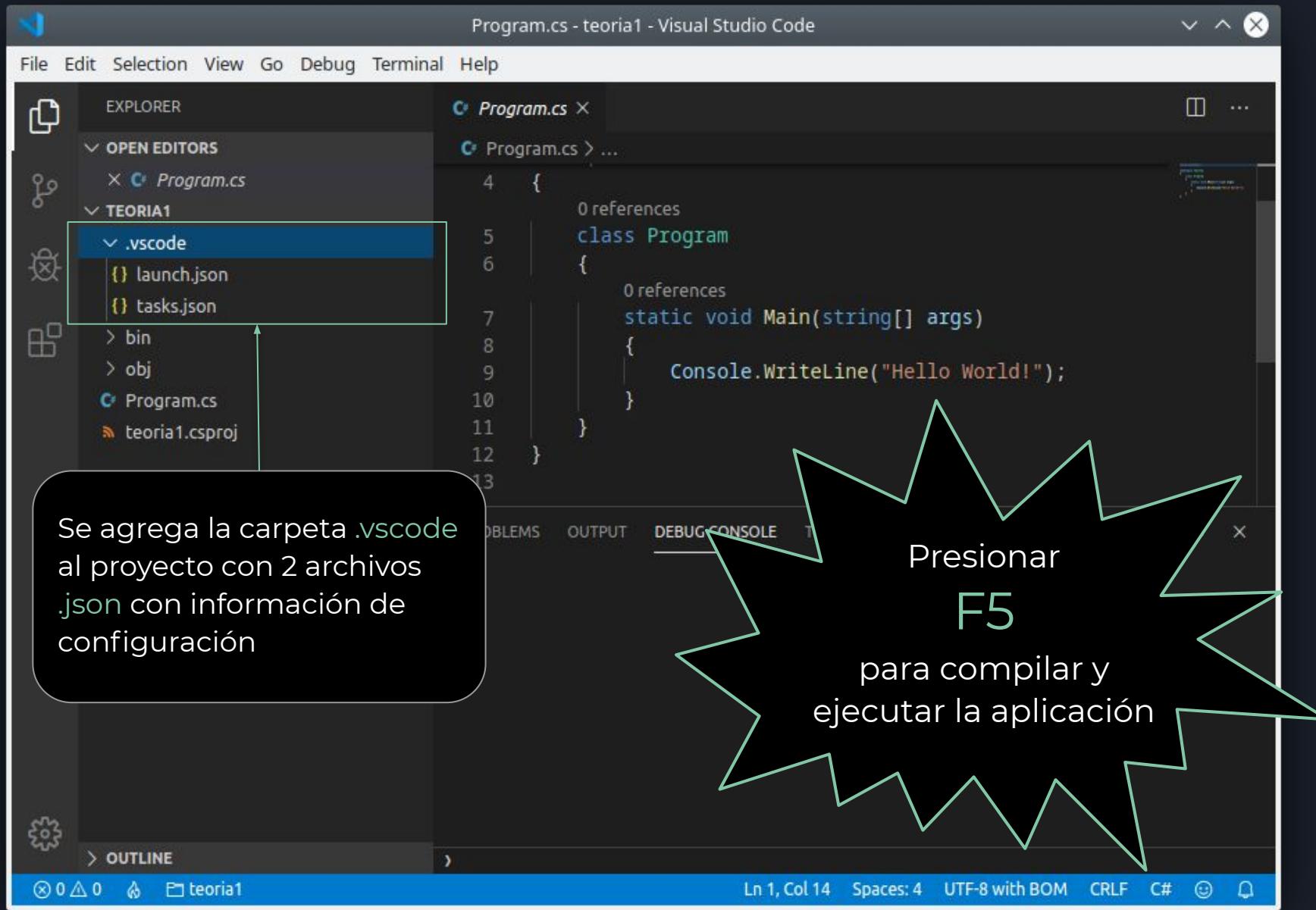
Downloading package 'Razor Language Server (Linux / x64)' (51492
KB)..... Done!
Installing package 'Razor Language Server (Linux / x64)'

Finished
```

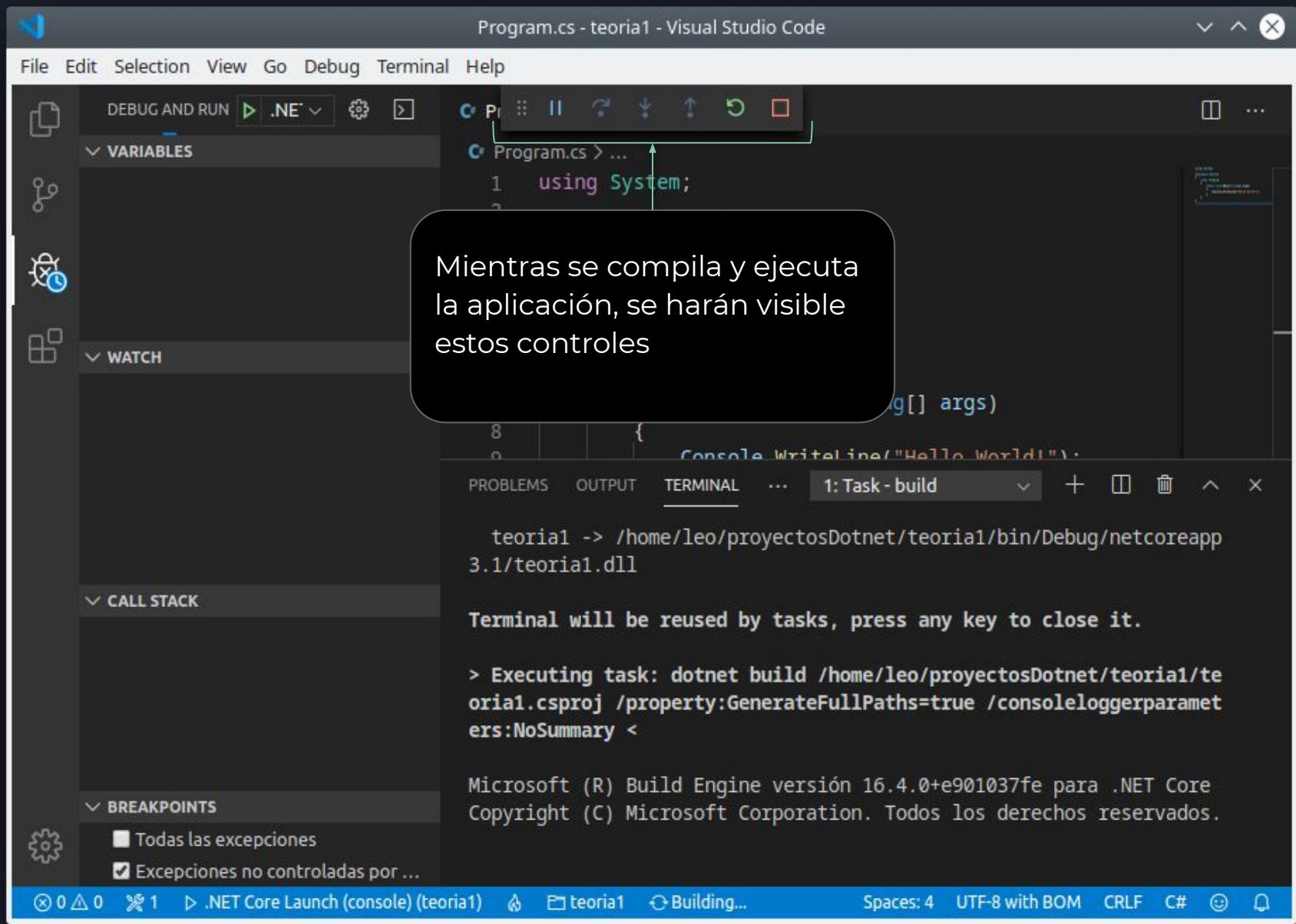
# Introducción a C# - Primera aplicación en C#



# Introducción a C# - Primera aplicación en C#



# Introducción a C# - Primera aplicación en C#



# Introducción a C# - Primera aplicación en C#

The screenshot shows the Visual Studio Code interface with a C# file named `Program.cs` open. The code contains a simple `Hello World` application. A green arrow points from a callout box to the `DEBUG CONSOLE` tab at the bottom of the interface.

En DEBUG CONSOLE, entre mensajes del debugger se observa la salida de la aplicación

```
1  using System;
2
3  namespace teoria1
4  {
5      class Program
6      {
7          static void Main(string[] args)
8          {
9              Console.WriteLine("Hello World!");
10         }
11     }
12 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
ng.dll". Se omitió la carga de símbolos. El módulo está optimizado y la opción del depurador 'Sólo mi código' está habilitada. cargado
0
"/home/leo/dotnet/shared/Microsoft.NETCore.App/3.1.0/System.Runtime.Extensions.dll". Se omitió la carga de símbolos. El módulo está optimizado y la opción del depurador 'Sólo mi código' está habilitada. cargado
"/home/leo/dotnet/shared/Microsoft.NETCore.App/3.1.0/System.Text.Encoding.Extensions.dll". Se omitió la carga de símbolos. El módulo está optimizado y la opción del depurador 'Sólo mi código' está habilitada. cargado
Hello World!
El programa "[8088] teoria1.dll" terminó con el código 0 (0x0).
```

BREAKPOINTS

- Todas las excepciones
- Excepciones no controladas por ...

0 △ 0 .NET Core Launch (console) (teoria1) teoria1 Ln 13, Col 1 Spaces: 4 UTF-8 with BOM CRLF C# ☺ 🔔

# Aplicación HelloWorld generada

```
using System;  
  
namespace teorial1  
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            Console.WriteLine("Hello World!");  
        }  
    }  
}
```

Espacio de nombre **teorial1** coincidente con el nombre del proyecto

Clase **Program** que contiene el método principal

Método especial denominado **Main** que represente el punto de inicio de la aplicación

### Aplicación HelloWorld generada

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

Es una clase

Es un método

Argumento pasado al método WriteLine

Se está invocando el método `WriteLine` de la clase `Console` pasando como parámetro el `string` que se desea imprimir en la consola.





## Comentar la primera línea utilizando //

```
// using System;  
  
namespace teorial1  
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            Console.WriteLine("Hello World!");  
        }  
    }  
}
```

El compilador ignora las líneas que comienzan con //





Presionar F5 para compilar y ejecutar

```
// using System;

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



## Error de compilación!

```
// using System;

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

El nombre 'Console' no existe en  
el contexto actual





## Corregir y probar nuevamente

```
// using System;

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

Agregar

# Introducción a C# - Primera aplicación en C#

```
// using System;

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



# Directiva using

- El nombre **extenso** de una clase lleva como prefijo el espacio de nombres en la que se ha definido.
- El nombre **extenso** de la clase **Console** definida en el espacio de nombres **System** es **System.Console**
- La directiva **using X** al comienzo del archivo fuente permite omitir el prefijo **X.** para todos los miembros del espacio de nombres **X .** El alcance de la directiva **using** es el archivo fuente donde se especifica
- En el ejemplo anterior fue necesario referirse a la clase por su nombre extenso **System.Console** porque se omitió la directiva **using System**

# Tipos integrados y operadores

# Tipos integrados en C#

- C# proporciona un conjunto estándar de tipos numéricos integrados para representar números enteros y valores de punto flotante
- Además de los tipos numéricos, C# proporciona tipos integrados para representar expresiones booleanas, caracteres de texto, cadenas de texto y objetos. El tipo `dynamic`, que veremos luego, también se considera integrado a C#
- Todos los tipos integrados de C# son tipos de .NET y pueden referenciarse por el nombre del tipo en la plataforma o por un alias propio de C# (a excepción de `dynamic`)

# Tipos enteros integrados en C#

Alias de C#	Intervalo	Tamaño	Tipo de .NET
<b>sbyte</b>	De -128 a 127	8 bits con signo	System.SByte
<b>byte</b>	De 0 a 255	8 bits sin signo	System.Byte
<b>short</b>	De -32 768 a 32 767	16 bits con signo	System.Int16
<b>ushort</b>	De 0 a 65.535	16 bits sin signo	System.UInt16
<b>int</b>	De -2.147.483.648 a 2.147.483.647	32 bits con signo	System.Int32
<b>uint</b>	De 0 a 4.294.967.295	32 bits sin signo	System.UInt32
<b>long</b>	De -9.223.372.036.854.775.808 a 9.223.372.036.854.775.807	64 bits con signo	System.Int64
<b>ulong</b>	De 0 a 18.446.744.073.709.551.615	bits sin signo	System.UInt64

# Tipos de punto flotante integrados en C#

Alias C#	Intervalo aproximado	Tamaño	Tipo de .NET	Precisión
<b>float</b>	De $\pm 1,5 \times 10^{-45}$ a $\pm 3,4 \times 10^{38}$	4 bytes	System.Single	6 a 9 dígitos aprox.
<b>double</b>	De $\pm 5,0 \times 10^{-324}$ a $\pm 1,7 \times 10^{308}$	8 bytes	System.Double	15 a 17 dígitos aprox.
<b>decimal</b>	De $\pm 1,0 \times 10^{-28}$ a $\pm 7,9228 \times 10^{28}$	16 bytes	System.Decimal	6 a 9 dígitos aprox.

# Demás tipos integrados en C#

Alias C#	Tipo de .NET
<b>char</b>	System.Char
<b>bool</b>	System.Boolean
<b>string</b>	System.String
<b>object</b>	System.Object
<b>dynamic</b>	System.Object

C# es sensible a mayúsculas y minúsculas

### ¡Atención!

C# es sensible a las mayúsculas y minúsculas, por lo tanto `bool` es un tipo válido de C#, en cambio `Bool` es inválido.

Observar que los alias de C# para los tipos integrados comienzan siempre en minúscula



# Tipos integrados en C#

- Los tipos integrados son estructuras, salvo `string` y `object` que son clases
- Como tipos de la plataforma, están definidos en el espacio de nombres `System`
- Por lo tanto, si se utiliza la directiva `using System` es posible referirse al tipo `System.Boolean` simplemente como `Boolean`
- Sin embargo, para el caso de los tipos integrados, lo usual es utilizar el alias definido para el lenguaje correspondiente

# Declaración de variables y constantes

```
static void Main(string[] args)
{
    // Declaración variable de tipo char
    char a;

    // Declaración múltiple variables de tipo int
    int b, c, d;

    //Declaración y asignación de variable de tipo double
    double e = 5.2;

    //declaración y asignación múltiple variables de tipo int
    int f = 2, g = 3;

    //declaración y asignación de constante
    const double pi = 3.1416;
}
```

# Literales

```
static void Main(string[] args)
{
    // Comillas simples para un literal de tipo char
    char a = 'A';

    // Comillas dobles para un literal de tipo string
    string st = "hola mundo!";

    // Un literal numérico sin punto decimal es int por defecto
    int i = 27;

    // El prefijo 0x se utiliza con una expresión hexadecimal
    int j = 0x1AFAE1FF;

    // Un literal numérico con punto decimal es double por defecto
    double pi = 3.1416;
}
```



Codificar Main de la siguiente manera:

```
 . . .
static void Main(string[] args)
{
    double d = 15.1;
    float f = 21.2;
}
. . .
```





## Compilar

```
 . . .  
 static void Main(string[] args)  
{  
     double d = 15.1;  
     float f = 21.2;  
 }  
 . . .
```

¿ Cuál es el  
problema ?



# Literales

```
...  
static void Main(string[] args)  
{  
    double d = 15.1;  
    float f = 21.2;  
}  
...
```

Un literal con punto decimal es por defecto de tipo **double**, por lo tanto no puede asignarse a una variable de tipo **float**

# Literales

```
...  
static void Main(string[] args)  
{  
    double d = 15.1;  
    float f = 21.2f;  
}
```



Se soluciona agregando  
el sufijo **f** al literal.  
21.2f es un literal de tipo  
**float**

Observar que ...

```
static void Main(string[] args)  
{  
    int i = 15;  
    byte b = 21;  
}
```

No se necesita  
ningún sufijo

No ocurre lo  
mismo para el  
caso de literales  
enteros.



# Sufijos para literales

- Literales enteros
  - Sin sufijo: int
  - **L, l**: long
  - **U, u**: unsigned
- Literales reales
  - Sin sufijo: double
  - **F, f**: float
  - **D, d**: double
  - **M, m**: decimal

# Operadores aritméticos

Operador	Operación
+	Suma
-	Resta
*	Multiplicación
/	División
%	Residuo

```
double resultado = 1/2;
```



**¡ Cuidado  
con la división !**

O los resultados pueden  
ser dolorosos

# Cuidado con la división

Probar, analizar y responder sobre el siguiente código para la próxima clase: ¿Por qué las variables **r1** y **r2** difieren?

```
static void Main(string[] args)
{
    double r1 = 17 / 3;
    double r2 = 17 / 3.0;
    Console.WriteLine(r1);
    Console.WriteLine(r2);
}
```



# Operadores relacionales

Operador	Operación
<	Menor que
>	Mayor que
<=	Menor o igual que
>=	Mayor o igual que
!=	Diferente de
==	Igual a

# Operadores lógicos

Operador	Operación
&	AND
	OR
!	NOT
^	XOR
&&	AND en cortocircuito
	OR en cortocircuito

# Operadores &, &&, | y ||

Probar, analizar y responder para la próxima clase sobre el código siguiente:

1. ¿Cuál es la salida por consola?
2. ¿si se reemplaza `int x = 2;` por `int x = 0;`?

```
static void Main(string[] args)
{
    int x = 2;
    int y = 5;
    Console.WriteLine(x != 0 && y / x == 2);
    Console.WriteLine(x != 0 & y / x == 2);
    Console.WriteLine(x == 0 || y / x == 2);
    Console.WriteLine(x == 0 | y / x == 2);
}
```



# Operadores de asignación

Operador	Operación
<code>++</code>	Incremento
<code>--</code>	Decremento
<code>=</code>	Asignación simple
<code>*=</code>	Multiplicación más asignación
<code>/=</code>	División más asignación
<code>%=</code>	Residuo más asignación
<code>+=</code>	Suma más asignación
<code>-=</code>	Resta más asignación

# Operador incremento

Probar, analizar y responder para la próxima clase sobre el código siguiente:

1. ¿Cuál es la salida por consola?
2. ¿En qué difieren el pre y post incremento?

```
static void Main(string[] args)
{
    int x = 1;
    Console.WriteLine(x++); //post incremento
    Console.WriteLine(x);

    Console.WriteLine(++x); //pre incremento
    Console.WriteLine(x);
}
```



# Operador decremento

Probar, analizar y responder para la próxima clase sobre el código siguiente:

1. ¿Cuál es la salida por consola?
2. ¿En qué difieren el pre y post decremento?



```
static void Main(string[] args)
{
    int x = 10;
    int y = x--; //post decremento
    Console.WriteLine(y);
    Console.WriteLine(x);

    y = --x; //pre decremento
    Console.WriteLine(y);
    Console.WriteLine(x);
}
```

# Operadores incremento y decremento

Responder para la próxima clase

¿Cuál es la salida por consola del siguiente código?

```
static void Main(string[] args)
{
    int x = 10;
    Console.WriteLine(x++ == 10);
    Console.WriteLine(x-- == 10);
    Console.WriteLine(++x == 10);
    Console.WriteLine(--x == 10);
}
```





**Ejercicio práctico:** Codificar un programa que solicite al usuario ingresar por teclado su nombre y saludarlo de manera personalizada



La siguiente sentencia

```
st = Console.ReadLine();
```

Lee un **string** desde la consola y lo asigna a la variable **st**



**Ejercicio práctico:** Codificar un programa que solicite al usuario ingresar por teclado su nombre y saludarlo de manera personalizada

```
static void Main(string[] args)
{
    Console.WriteLine("Ingrese su nombre");
    string nombre = Console.ReadLine();
    Console.WriteLine("Hola " + nombre);
}
```





# Compilar el ejercicio en el Visual Studio Code

No es posible ingresar datos por teclado desde la consola interna de Visual Studio Code. Tenemos dos opciones:

1. Compilar y ejecutar desde una terminal del sistema operativo usando el comando `dotnet run`
2. Configurar las opciones de Visual Studio Code en el proyecto para que utilice una terminal del sistema operativo



# Introducción a C# - Ejercitación

File Edit Selection View Go Debug Terminal Help

EXPLORER OPEN EDITORS

- Program.cs
- launch.json
- TEORIA1
  - .vscode
    - launch.json
    - tasks.json
  - bin
  - obj
- Program.cs
- teoria1.csproj

```
1  {
2      // Use IntelliSense to learn about this JSON config
3      // Hover to view descriptions of the properties
4      // For more information, visit: https://go.microsoft.com/fwlink/?linkid=808688
5      "version": "0.2.0",
6      "configurations": [
7          {
8              "name": ".NET Core Launch (console)",
9              "type": "coreclr",
10             "request": "launch",
11             "preLaunchTask": "build",
12             "program": "${workspaceFolder}/bin/Debug/netcoreapp3.1/teoria1.dll",
13             "args": [],
14             "cwd": "${workspaceFolder}",
15             "console": "internalConsole",
16             "stopAtEntry": false
17         },
18     ]
19 }
```

En el archivo `launch.json` cambiar el valor `"internalConsole"` por `"externalTerminal"`

Add Configuration...

Ln 25, Col 2 Spaces: 4 UTF-8 LF JSON with Comments

DEBUG AND RUN ▶ .NET



Program.cs

launch.json

## VARIABLES

## WATCH

Guardar los cambios y presionar **F5**. Ahora se compilará y ejecutará la aplicación en una terminal del sistema operativo

## BREAKPOINTS

 Todas las excepciones Excepciones no controladas por ...

```
1  {
2      // Use IntelliSense to learn about possible attributes
3      // Hover to view descriptions of existing attributes
4      // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830302
5      "version": "0.2.0",
6      "configurations": [
7          {
8              "name": ".NET Core Launch (console)",
9              "type": "coreclr",
10             "request": "launch",
11             "preLaunchTask": "build",
12             "program": "${workspaceFolder}/bin/Debug/netcoreapp3.1/teoria1.dll",
13             "args": [],
14             "cwd": "${workspaceFolder}",
15             "console": "externalTerminal",
16             "stopAtEntry": false
17         },
18         {
19             "name": ".NET Core Attach",
20             "type": "coreclr",
21             "request": "attach",
22             "processId": "${command:pickProcess}"
23         }
24     ]
25 }
26 }
```

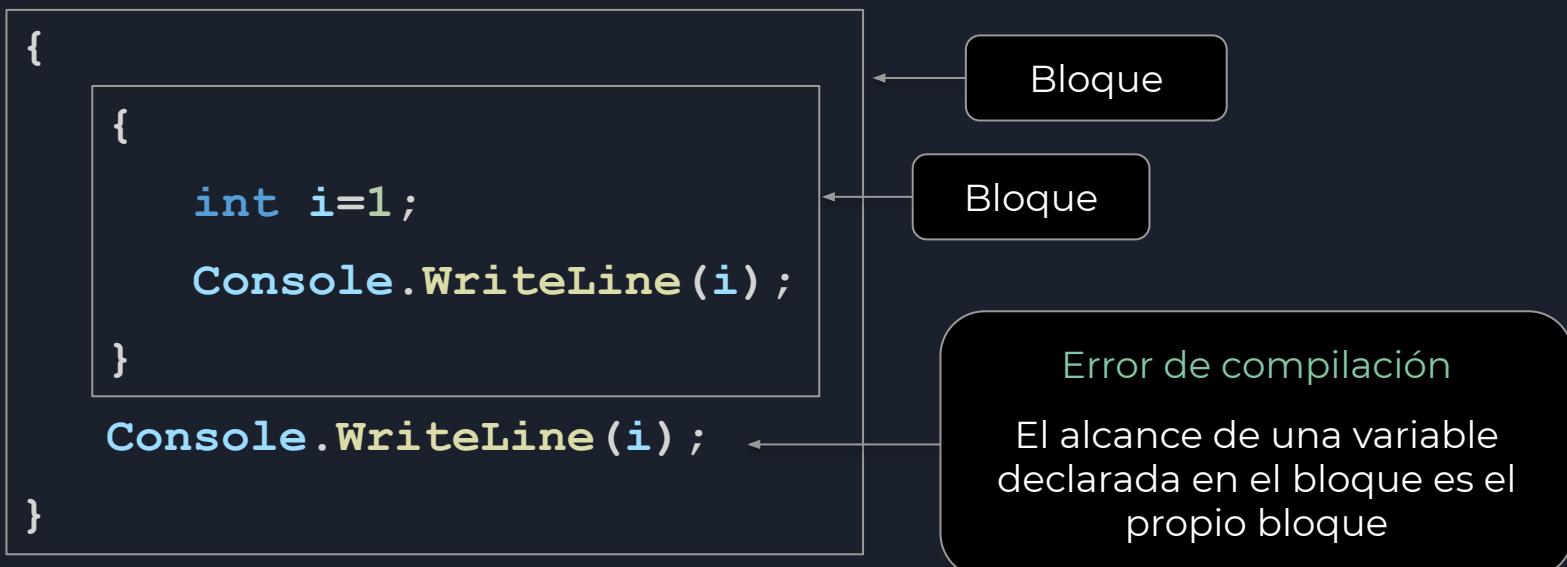
Add Configuration...

# Estructuras de control

### Estructura de control: Bloque

Es una lista de cero, o más sentencias encerradas entre llaves {}

```
static void Main(string[] args)
```



# Estructura de control: Bloque

No se puede ocultar el nombre de una variable en un bloque más interno (anidado)

```
static void Main(string[] args)
{
    int i;
    {
        double j = 1.1;
    }
    {
        char j = 'A';
        int i = 1;
    }
}
```

### Válido

Es posible volver a utilizar el nombre de una variable en un bloque hermano. Estas dos variables j son variables distintas

### Error de compilación

La variable **i** no se puede declarar en este ámbito porque ese nombre se está usando en un ámbito local envolvente

# Estructura de control: Condicional (if)

```
if (condición)
{
    //bloque que se ejecuta
    //si condición es verdadera
}

else
{
    //bloque que se ejecuta
    //si condición es verdadera
}
```

La condición es una **expresión booleana**. Va siempre entre paréntesis

la parte del **else** puede ser omitida



### Operador condicional ?:

También conocido como operador condicional ternario.

```
condición ? consiguiente : alternativa
```

Ejemplo de uso:

```
static void Main(string[] args) {  
    int n = 10;  
    string st = (n < 0) ? "negativo" : "no negativo";  
    Console.WriteLine(st);  
}
```



### Estructura de control: Switch

```
switch (expresion) ←  
{  
    case valor1:  
        // lista de instrucciones para  
        // cuando expresion == valor1  
        break;  
    case valor2:  
        // lista de instrucciones para  
        // cuando expresion == valor2  
        break;  
    default:  
        // lista de instrucciones para  
        // cuando no hubo coincidencia  
        // con ninguno de los casos anteriores  
        break;  
}
```

La expresión cuyo valor se va a utilizar buscando coincidencias en las secciones  
case va siempre entre paréntesis

## Estructura de control: while

```
static void Main(string[] args)
{
    int i = 1;
    while (i <= 3)
    {
        Console.WriteLine(i);
        i++;
    }
}
```

Bucle

La condición siempre va entre paréntesis

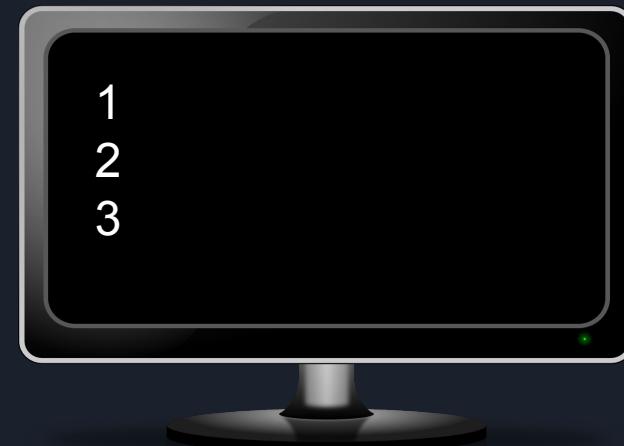


# Estructura de control: do-while

```
static void Main(string[] args)
{
    int i = 1;
    do
    {
        Console.WriteLine(i);
        i++;
    }
    while (i <= 3);
}
```

Bucle

La condición siempre va entre paréntesis



### Estructura de control: `for`

```
for (<inicialización>; <condición>; <iterador>)
{
    < código del bucle >
}
```

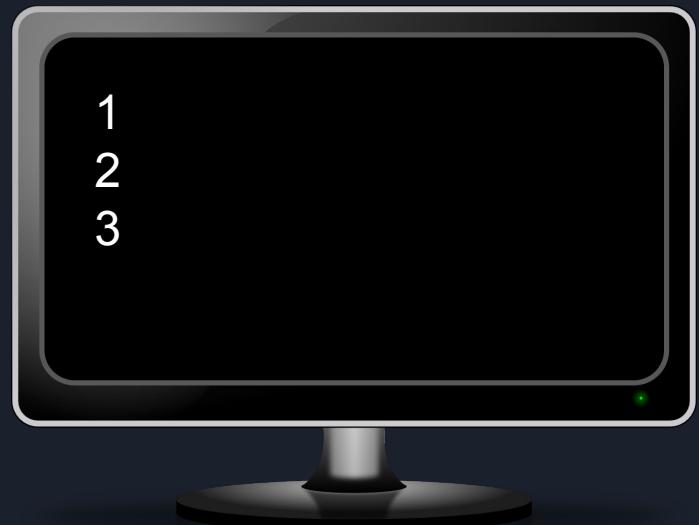
Es equivalente a:

```
<inicialización>
while (<condición>)
{
    < código del bucle >
    < iterador >
}
```

### Estructura de control: `for`

Ejemplo:

```
static void Main(string[] args)
{
    int i;
    for (i = 1; i <= 3; i++)
    {
        Console.WriteLine(i);
    }
}
```



## Estructura de control: `for`

Distintas maneras de hacer lo mismo

```
static void Main(string[] args)
{
    int i = 1;
    for (; i <= 3; i++)
    {
        Console.WriteLine(i);
    }
}
```

```
static void Main(string[] args)
{
    int i = 1;
    for (; i <= 3; )
    {
        Console.WriteLine(i++);
    }
}
```

```
static void Main(string[] args)
{
    int i = 1;
    for ( ; ; )
    {
        Console.WriteLine(i++);
        if (i > 3) break;
    }
}
```

Las secciones  
del `for` pueden  
omitirse

# Estructura de control: `for`

## Alternativa

```
static void Main(string[] args)
{
    for (int i = 1; i <= 3; i++)
    {
        Console.WriteLine(i);
    }
    . . .
}
```

Es muy común declarar la variable en la sección de inicialización.

De esta manera, el ámbito de la variable `i` es el bloque del `for`

En este punto la variable `i` no está accesible

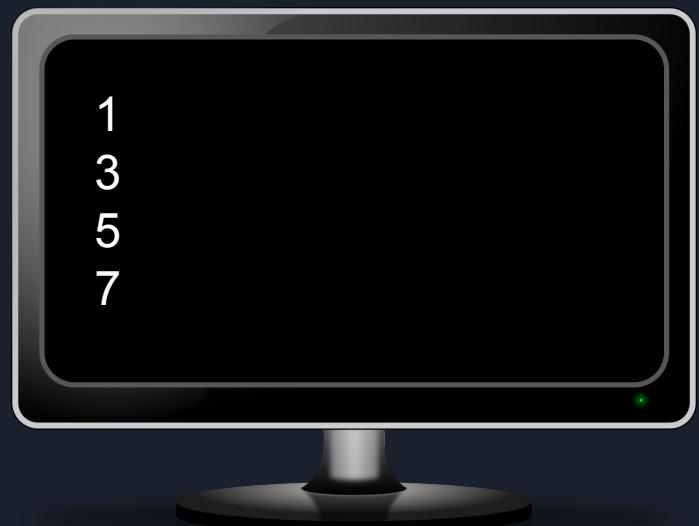
# Interrupción de bucles

- **break** – El bucle termina inmediatamente
- **continue** – Termina el ciclo corriente inmediatamente (la ejecución continúa con el próximo ciclo)
- **goto** – Permite saltar fuera del bucle (no recomendada)
- **return** – Salta fuera del bucle y del método que lo contiene

## Interrupción de bucles

Ejemplo:

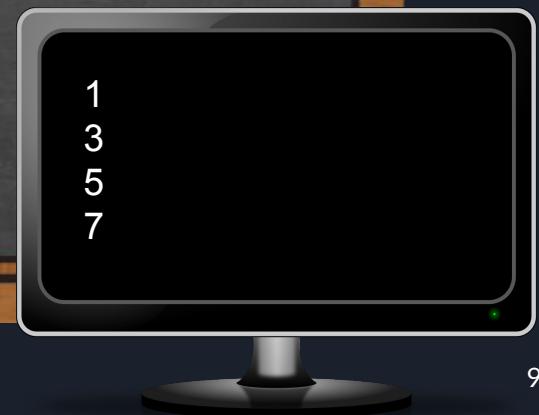
```
static void Main(string[] args)
{
    for (int i = 1; i <= 20; i++)
    {
        if (i == 8)
            break;
        if (i % 2 == 0)
            continue;
        Console.WriteLine(i);
    }
}
```



Una mejor y más legible solución que la presentada en la diapositiva anterior



```
static void Main(string[] args)
{
    for (int i = 1; i <= 7; i += 2)
    {
        Console.WriteLine(i);
    }
}
```





**Ejercicio práctico:** Solicitar al usuario que ingrese por teclado un número n y calcular la sumatoria desde 1 hasta n



La siguiente sentencia

```
n = int.Parse(Console.ReadLine());
```

Lee un **string** desde la consola, lo convierte a entero y lo asigna a la variable **n**



**Ejercicio práctico:** Solicitar al usuario que ingrese por teclado un número n y calcular la sumatoria desde 1 hasta n

```
static void Main(string[] args)
{
    Console.WriteLine("Ingrese n");
    int n = int.Parse(Console.ReadLine());
    int suma=0;
    for(int i = 1; i <= n; i++)
    {
        suma += i; //ídem a: suma = suma + i;
    }
    Console.WriteLine("Suma de 1 a " + n + " = " + suma);
}
```



Fin

