

Horarios

Primer Turno

- Teoría: Lunes de 15:30 a 18:00
- Práctica: Viernes de 16:30 a 18:30

Segundo Turno

- Teoría: Miércoles de 18:00 a 20:30
- Práctica: Viernes de 18:30 a 20:30

Método de Evaluación

Aprobación de la cursada

- Aprobar dos trabajos de programación obligatorios realizados en grupo.
- Aprobar un coloquio final e individual sobre los trabajos presentados.

Promoción

- El alumno que aprueba la cursada obtiene la promoción.

Fechas importantes

06/05/2022: Fecha límite para entregar el primer trabajo de programación solicitado

16/06/2022: Fecha límite para entregar el trabajo final de programación solicitado

Los coloquios sobre los trabajos entregados se tomarán a partir del 27/06/2022 en los horarios de práctica y teoría

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
.NET 6.0 (recommended)	LTS ⓘ	6.0.2	February 8, 2022	November 08, 2024
.NET 5.0	Current ⓘ	5.0.14	February 8, 2022	May 08, 2022
.NET Core 3.1	LTS ⓘ	3.1.22	December 14, 2021	December 03, 2022

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 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 for Windows' section with a dropdown menu showing 'Stable Build' selected. It lists download options for Windows x64 (User Installer), macOS, and Linux x64, along with links for '.deb' and '.rpm' files. Below this is a 'Other downloads' section. The main right-hand side of the screenshot shows the actual Visual Studio Code interface. It has a dark theme with several tabs open: 'src > JS serviceWorker.js > register > window.addEventListener('load') callback'. The code editor shows some JavaScript code related to service workers. In the bottom right corner of the interface, there's a terminal window with the command 'node' entered. The status bar at the bottom provides information like the local host URL (http://localhost:3000) and the file path 'create-react-app'. The status bar also indicates the current file is 'serviceWorker.js', the encoding is 'UTF-8', and the character count is 'Ln 43, Col 19'.



.NET

Teoría 1

Generalidades de .NET

Un poco de historia .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).

Un poco de historia .NET Core y .Net

- 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**
- En noviembre de 2021 se liberó **.NET 6.0**.

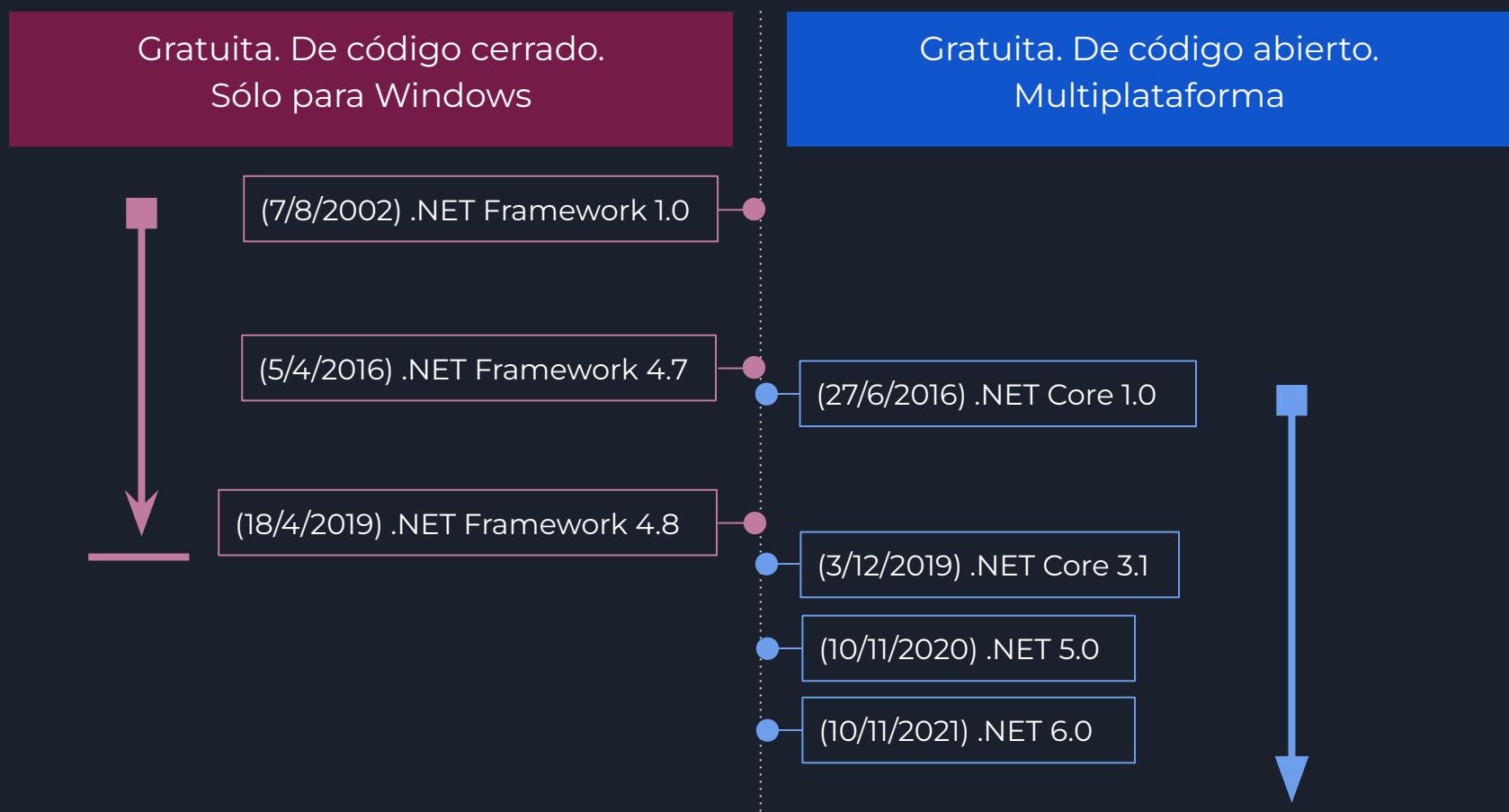
.NET Core 3.1 evolucionó a .NET 5.0

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

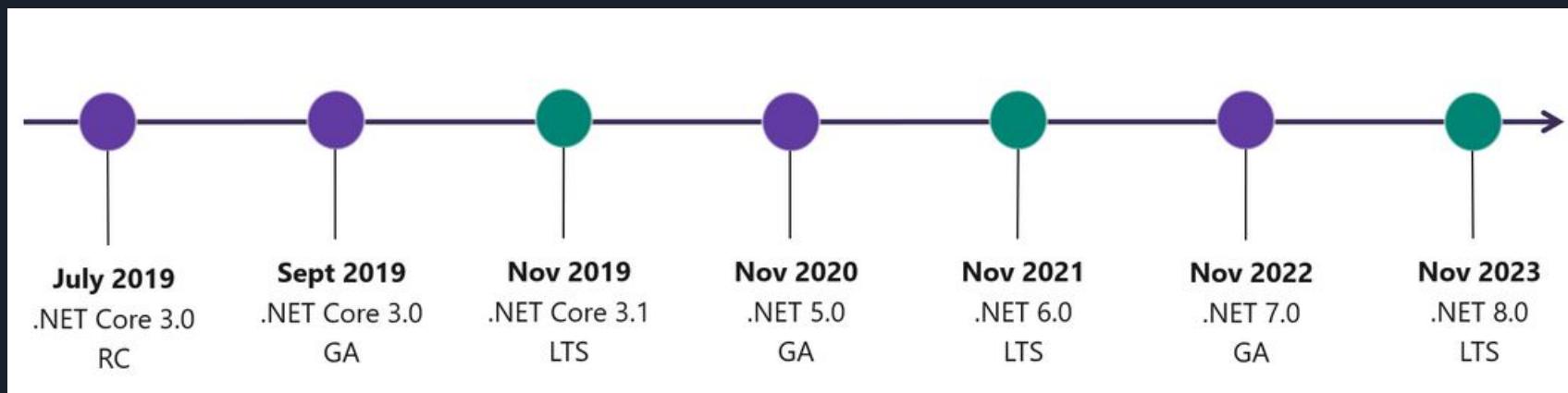
Aunque pudo llamarse .NET Core 5.0, se prefirió
cambiar el nombre a .NET 5.0 para dar idea de
unificación. En el futuro sólo se hablará de .NET

Generalidades de .NET

Evolución



Hoja de ruta de .NET



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

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

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

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**: (iniciado como **.Net Core**) es una implementación multiplataforma de .NET

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).

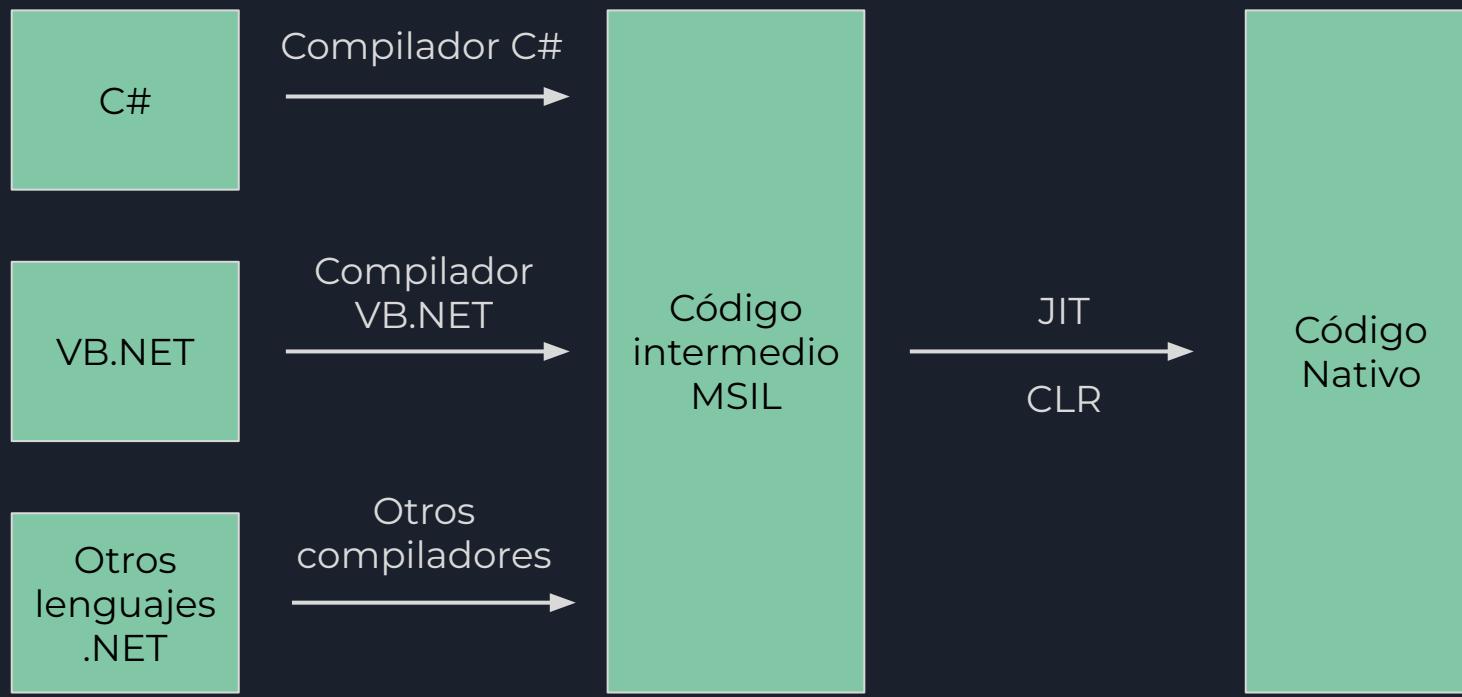
MSIL es un lenguaje similar a un código ensamblador pero de más alto nivel, creado para un hipotético procesador virtual que no está atado a una arquitectura determinada.

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 tipear ...

leo : bash — Konsole

Archivo Editar Ver Marcadores Preferencias Ayuda

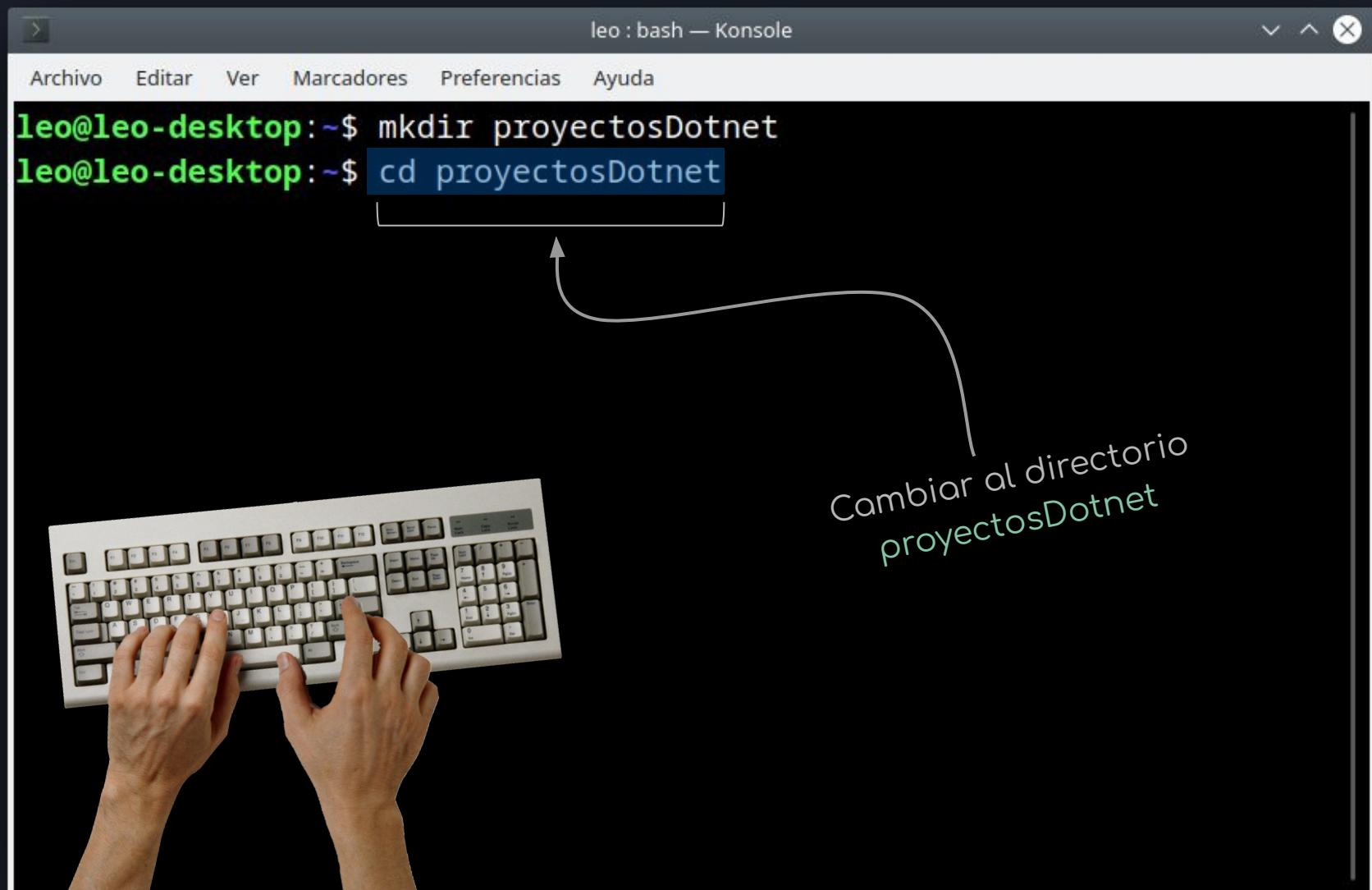
```
leo@leo-desktop:~$ mkdir proyectosDotnet
```

Crear el directorio
proyectosDotnet

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



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



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 with his right hand towards a chalkboard. The chalkboard has a wooden frame and contains text.

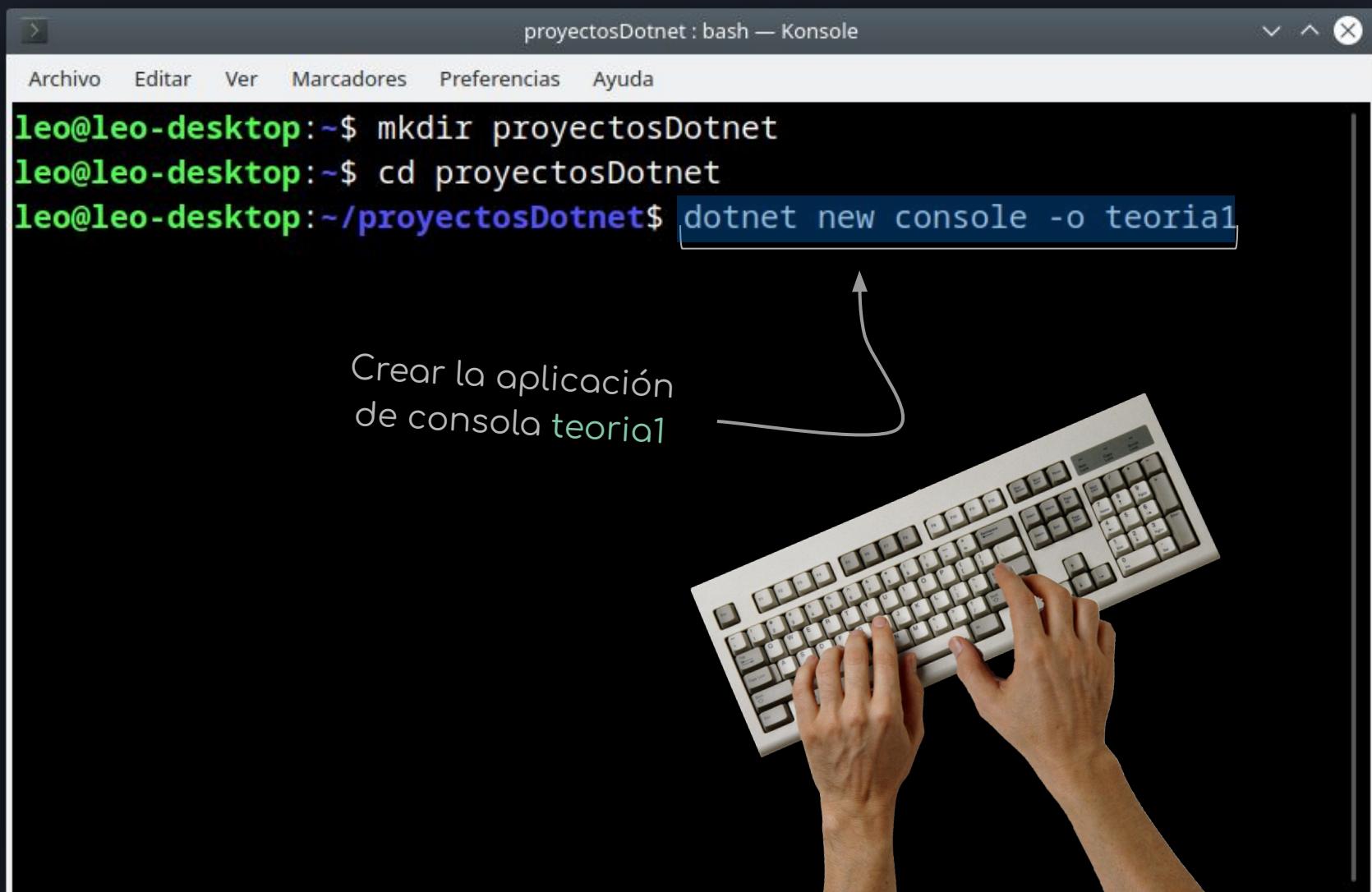
Para crear nuestra aplicación vamos a utilizar .NET CLI

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 tipear ...



```
proyectosDotnet : bash — Konsole
Archivo Editar Ver Marcadores Preferencias Ayuda
leo@leo-desktop:~$ mkdir proyectosDotnet
leo@leo-desktop:~$ cd proyectosDotnet
leo@leo-desktop:~/proyectosDotnet$ dotnet new console -o teoria1
```

Crear la aplicación de consola teoria1



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

```
proyectosDotnet : bash — Konsole
Archivo Editar Ver Marcadores Preferencias Ayuda
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.
```

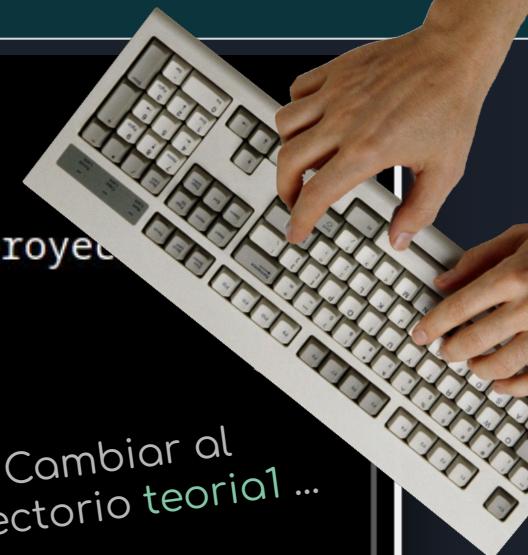
```
Restore succeeded.
```

```
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



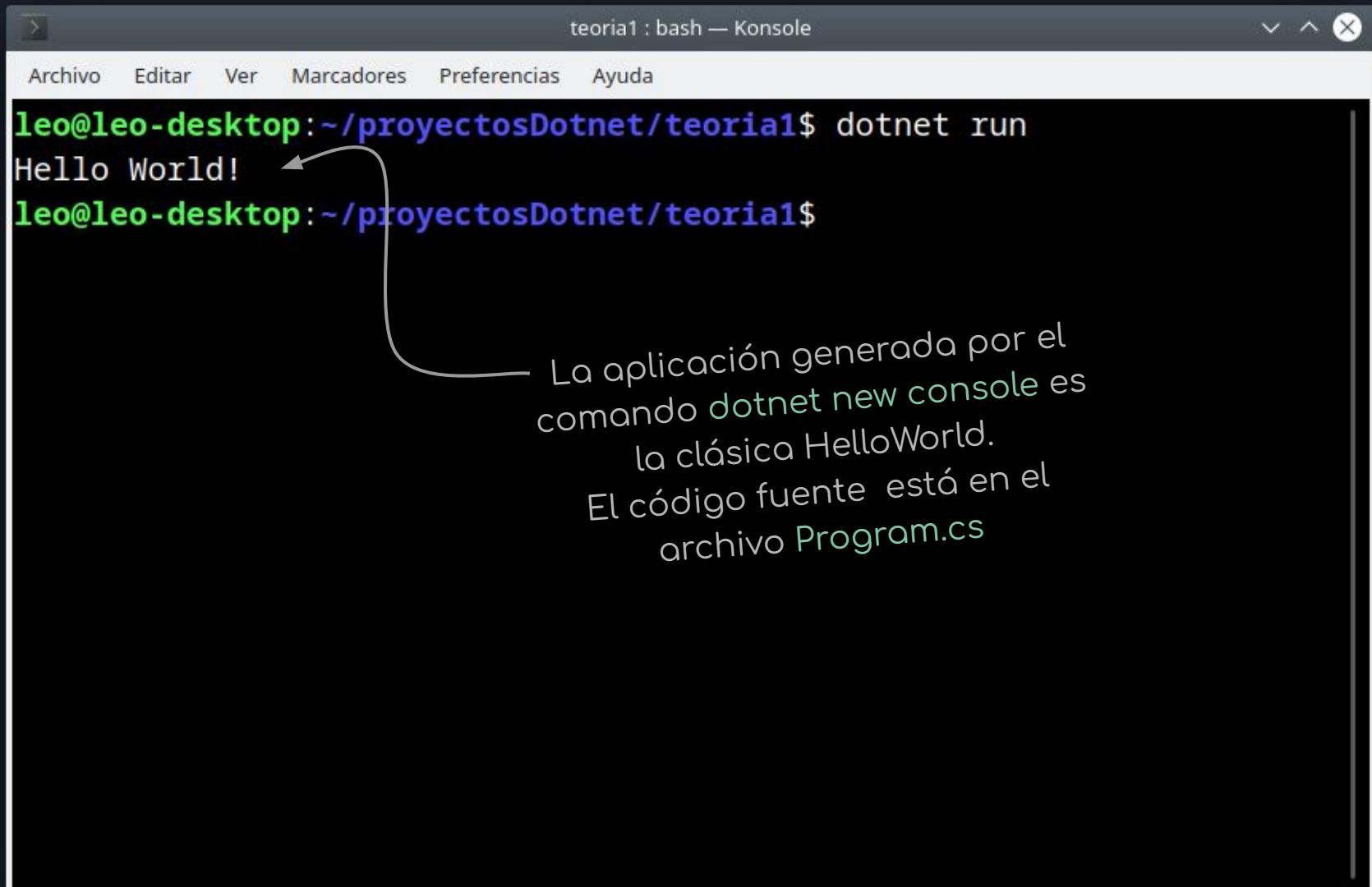
En la terminal del sistema operativo hacer:



```
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 main content area shows an instance of VS Code running in a browser window. The terminal tab in VS Code displays the command 'node' and the output 'You can now view create-react-app in the browser.' The status bar at the bottom of the VS Code window shows 'Local: http://localhost:3000/ On Your Network: http://10.211.55.3:3000/'. The status bar also includes information about the current file ('serviceWorker.js'), the editor ('create-react-app'), and the terminal ('node').



En la terminal del sistema operativo hacer:

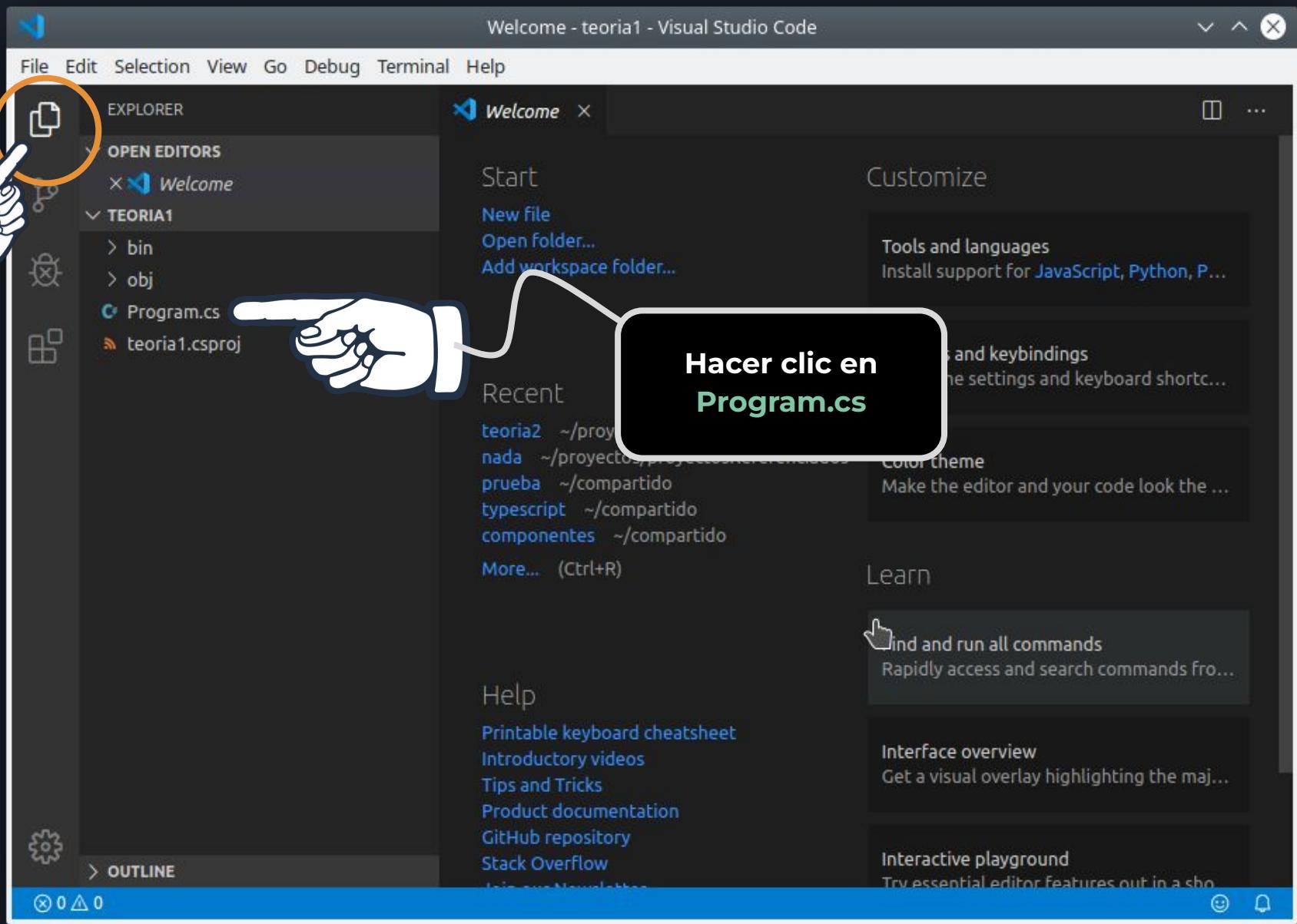
```
teoria1 : bash — Konsole
Archivo Editar Ver Marcadores Preferencias Ayuda
leo@leo-desktop:~/proyectosDotnet/teoria1$ dotnet run
Hello World!
leo@leo-desktop:~/proyectosDotnet/teoria1$ code .
```

El punto (.) indica la carpeta actual

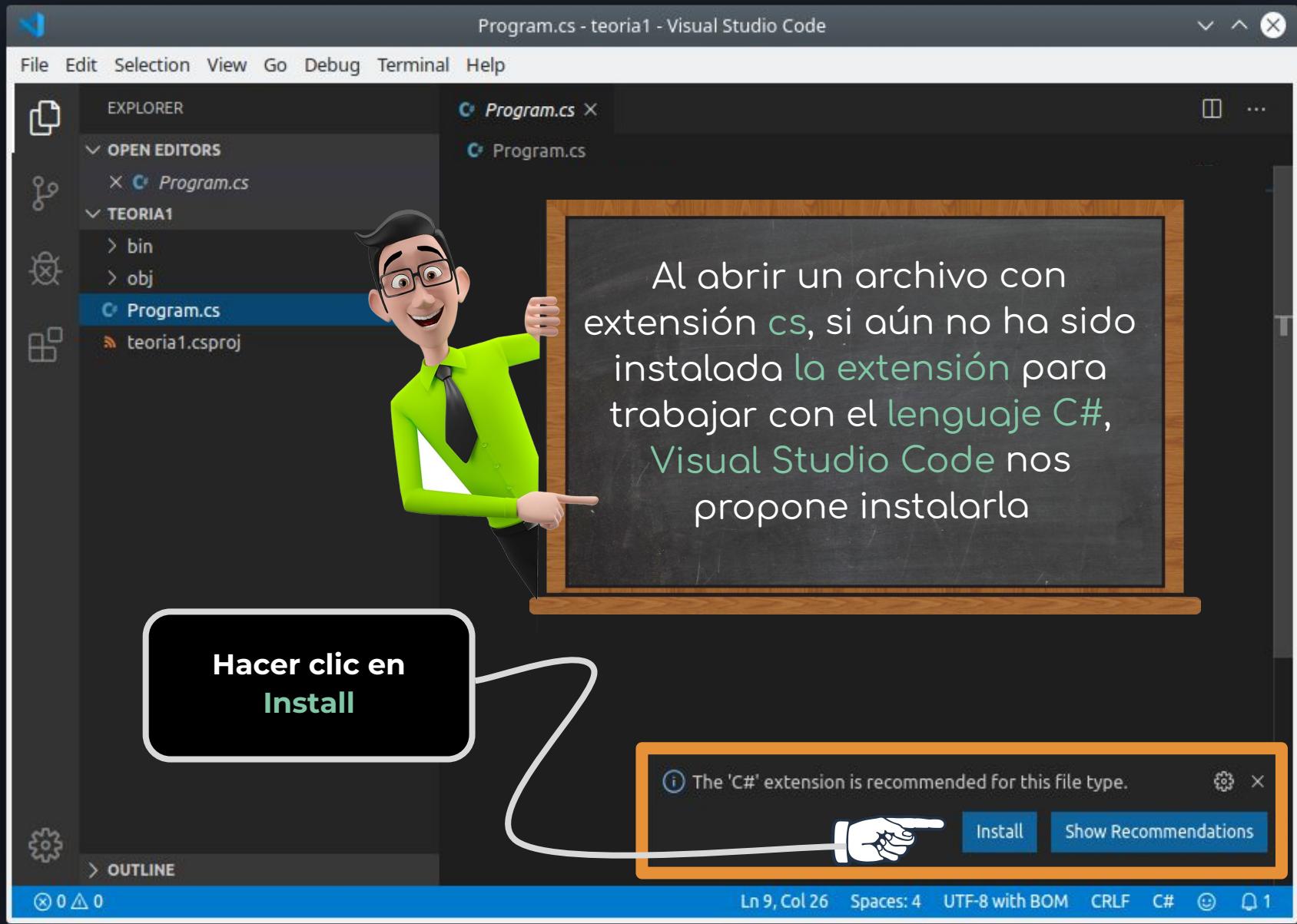
Abrir Visual Studio Code en la carpeta actual

35

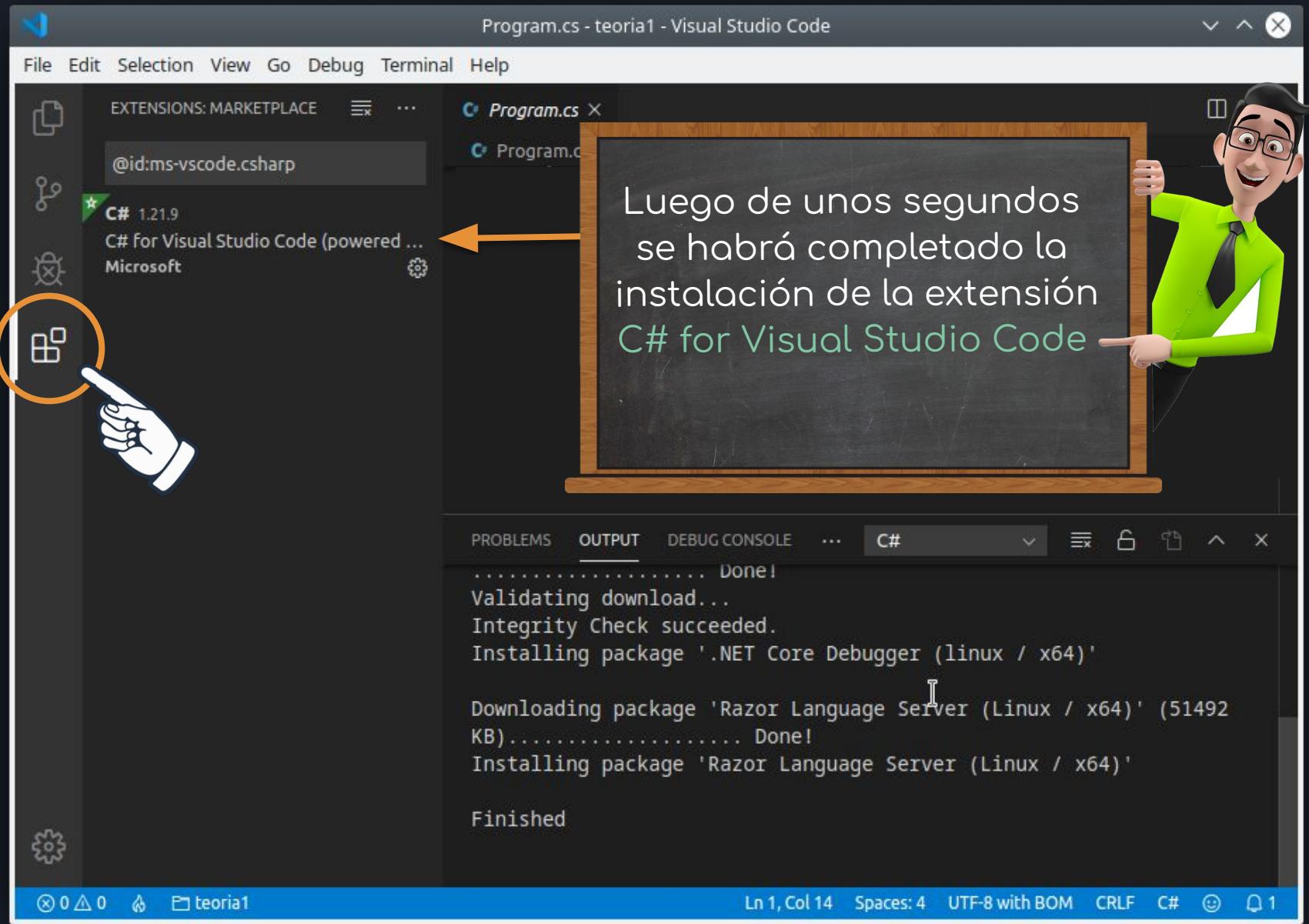
Introducción a C# - Primera aplicación en C#



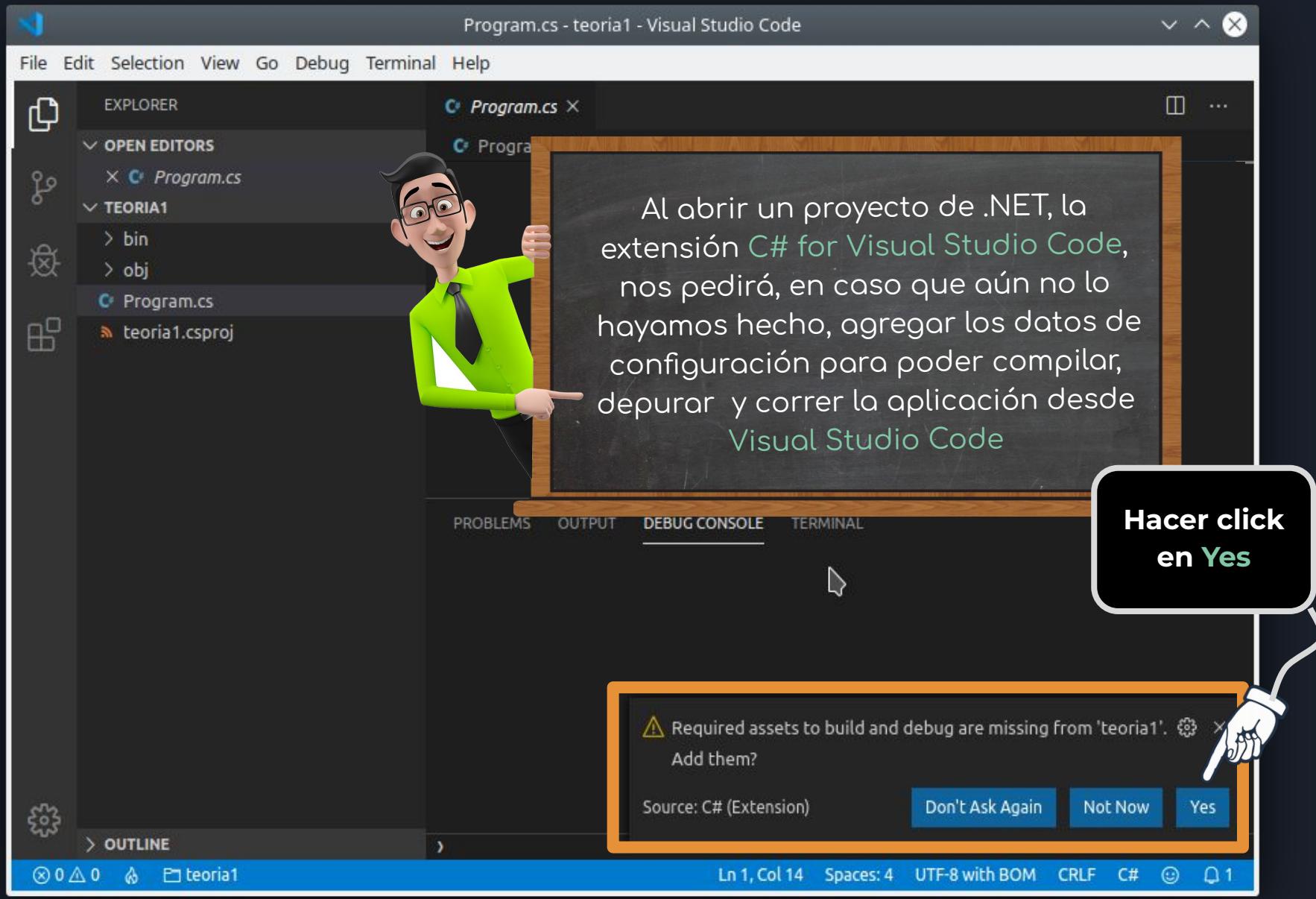
Introducción a C# - Primera aplicación en C#



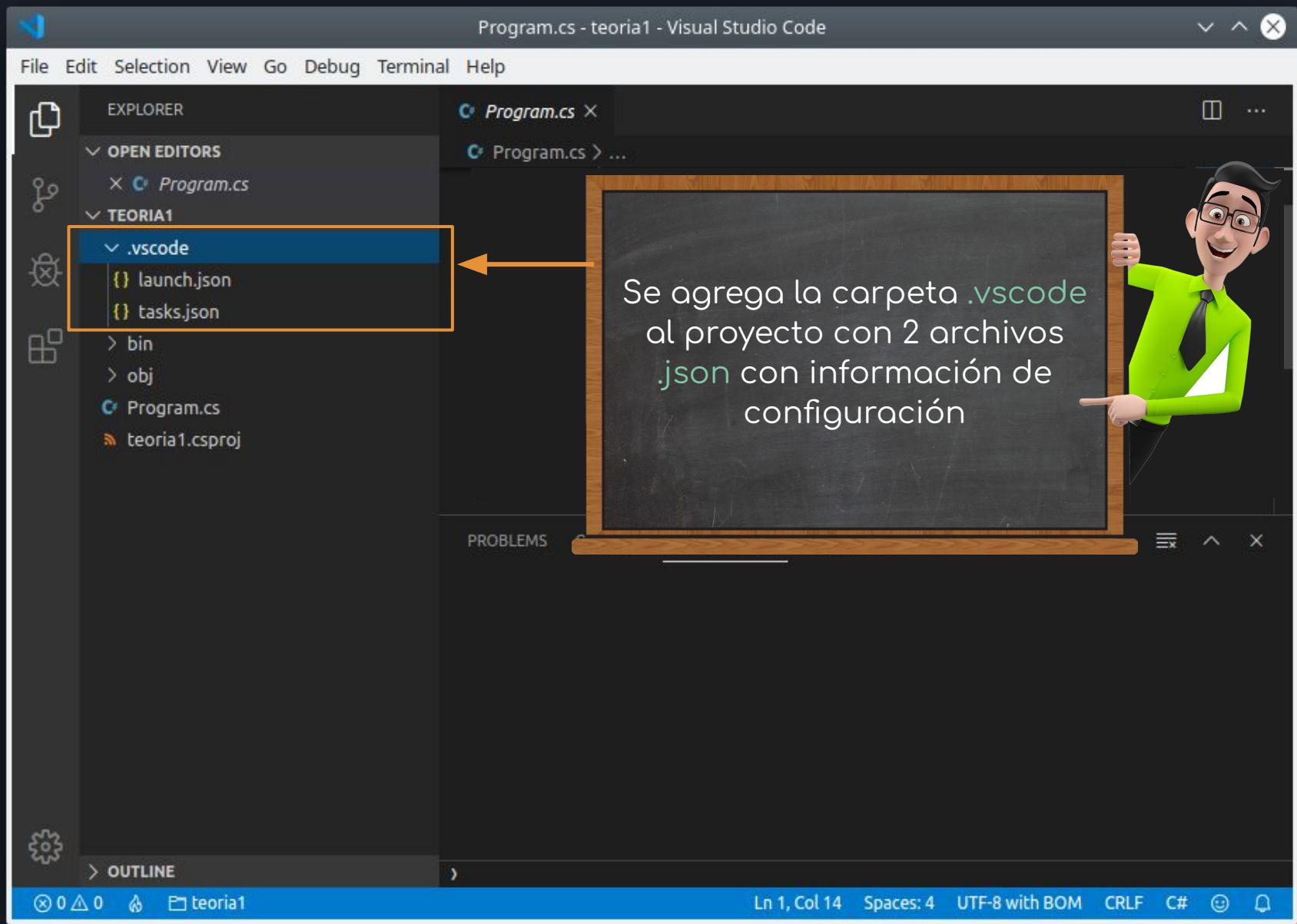
Introducción a C# - Primera aplicación en C#



Introducción a C# - Primera aplicación en C#



Introducción a C# - Primera aplicación en C#



The screenshot shows the Visual Studio Code interface with the title "Program.cs - teoria1 - Visual Studio Code". The menu bar includes File, Edit, Selection, View, Go, Debug, Terminal, and Help. The Explorer sidebar on the left shows the project structure: "OPEN EDITORS" with "Program.cs", "TEORIA1" with ".vscode" (containing "launch.json" and "tasks.json"), "bin", "obj", "Program.cs", and "teoria1.csproj". The main editor area displays two files: "Program.cs" and "Program.cs > ...". A large orange arrow points from the explanatory text on the right towards the ".vscode" folder in the Explorer sidebar. The explanatory text is contained within a chalkboard graphic and reads: "Se agrega la carpeta .vscode al proyecto con 2 archivos json con información de configuración". A cartoon character of a man with glasses and a green shirt is pointing towards the chalkboard.

Program.cs - teoria1 - Visual Studio Code

File Edit Selection View Go Debug Terminal Help

EXPLORER

OPEN EDITORS

Program.cs

Program.cs > ...

TEORIA1

.vscode

launch.json

tasks.json

bin

obj

Program.cs

teoria1.csproj

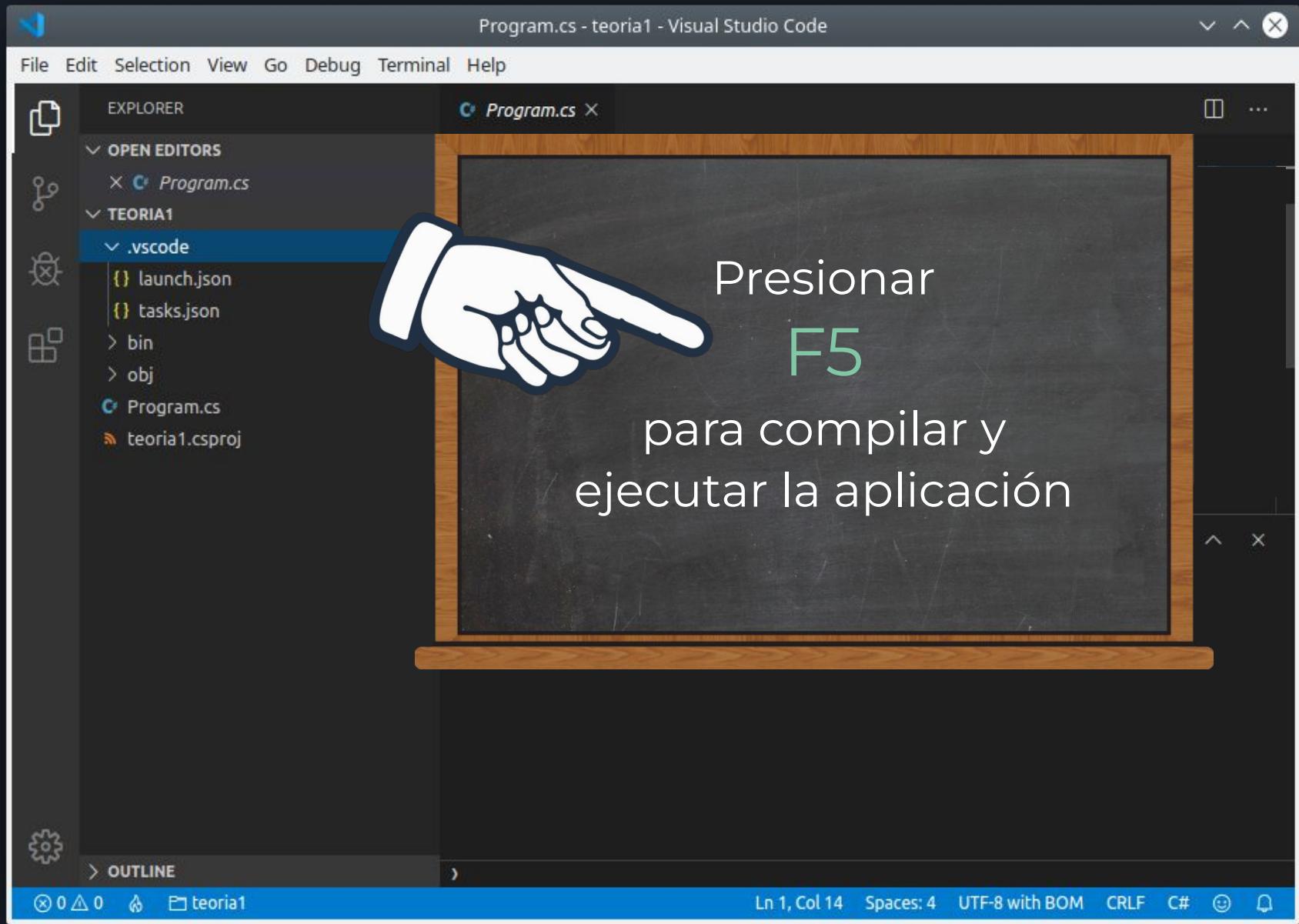
PROBLEMS

Se agrega la carpeta .vscode al proyecto con 2 archivos json con información de configuración

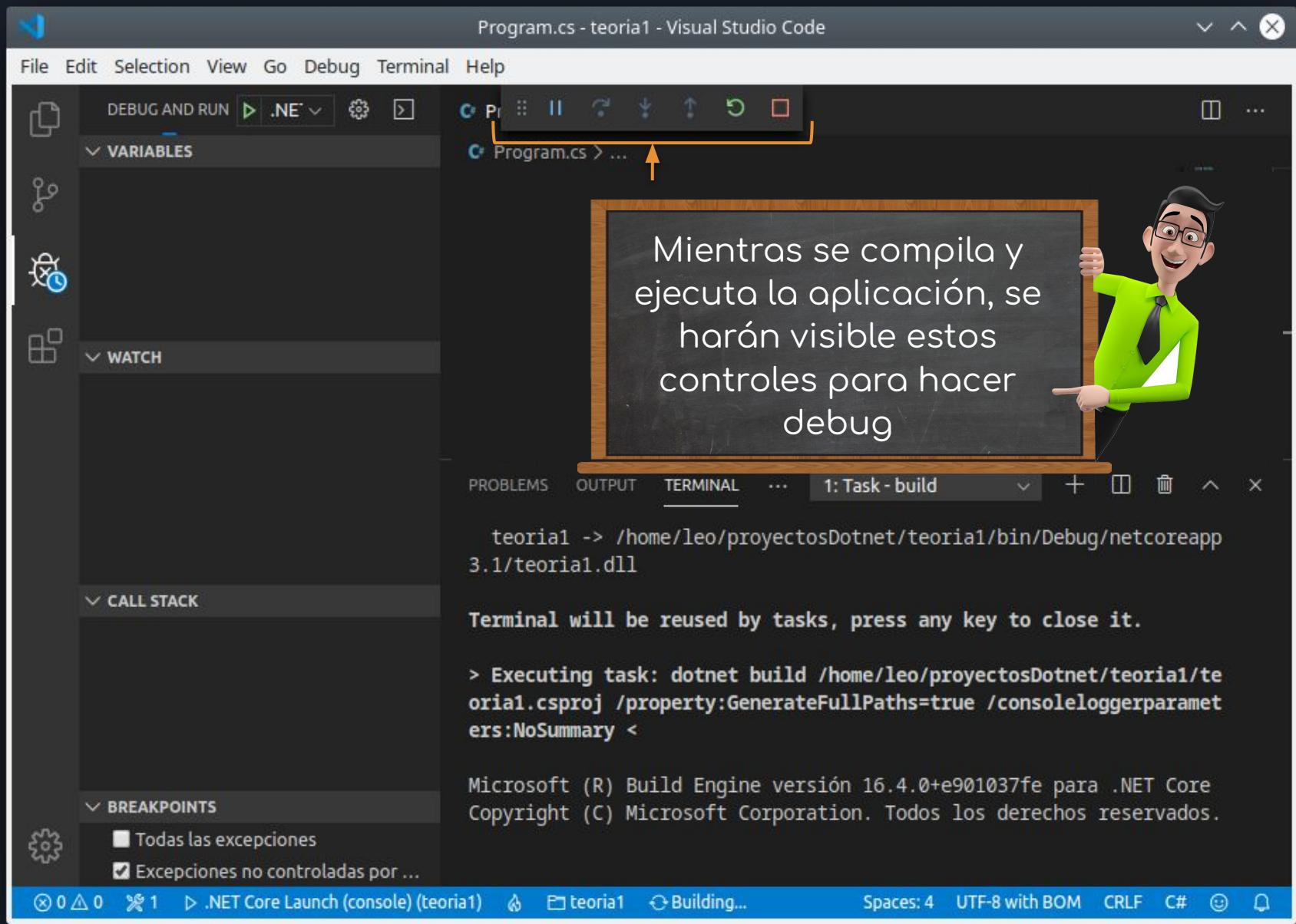
OUTLINE

Ln 1, Col 14 Spaces: 4 UTF-8 with BOM CRLF C# 😊

Introducción a C# - Primera aplicación en C#



Introducción a C# - Primera aplicación en C#



Mientras se compila y ejecuta la aplicación, se harán visible estos controles para hacer debug

PROBLEMS OUTPUT TERMINAL ... 1: Task - build + X

```
teoria1 -> /home/leo/proyectosDotnet/teoria1/bin/Debug/netcoreapp3.1/teoria1.dll
Terminal will be reused by tasks, press any key to close it.

> Executing task: dotnet build /home/leo/proyectosDotnet/teoria1/teoria1.csproj /property:GenerateFullPaths=true /consoleloggerparameters:NoSummary <
```

Microsoft (R) Build Engine versión 16.4.0+e901037fe para .NET Core
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

File Edit Selection View Go Debug Terminal Help

DEBUG AND RUN .NET

VARIABLES

WATCH

CALL STACK

BREAKPOINTS

Todas las excepciones

Excepciones no controladas por ...

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

Introducción a C# - Primera aplicación en C#

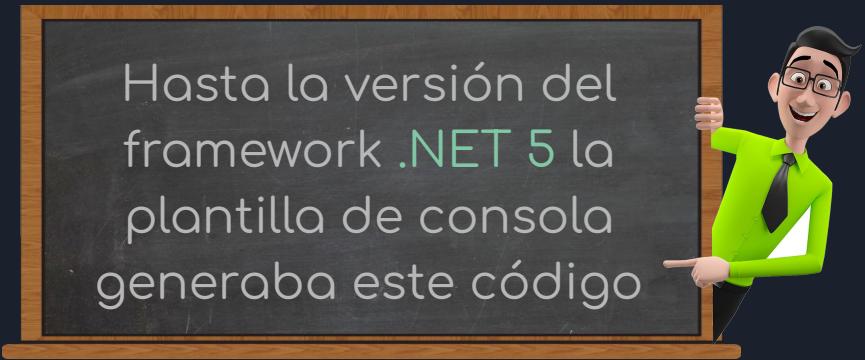
The screenshot shows the Visual Studio Code interface with the title bar "Program.cs - teoria1 - Visual Studio Code". The menu bar includes File, Edit, Selection, View, Go, Debug, Terminal, and Help. On the left, there are several toolbars: DEBUG AND RUN, VARIABLES, WATCH, CALL STACK, and BREAKPOINTS. A cartoon character is pointing at a chalkboard that contains the text: "En DEBUG CONSOLE, entre mensajes del debugger se observa la salida de la aplicación". An orange arrow points from the BREAKPOINTS toolbar down to the DEBUG CONSOLE tab in the center. The DEBUG CONSOLE tab is active, showing the following text:

```
Program.cs(1,1): warning CS0619: 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.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).
```

The status bar at the bottom shows: 0 0 .NET Core Launch (console) (teoria1) teoria1 Ln 13, Col 1 Spaces: 4 UTF-8 with BOM CRLF C# 43

Archivo Program.cs en la versión .NET 5

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



Archivo Program.cs en la versión .NET 5

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

Espacio de nombre **teorial** 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

Archivo Program.cs a partir de .NET 6

```
// See https://aka.ms/new-console-template for more information  
Console.WriteLine("Hello, World!");
```

La doble barra (`//`)
indica que la línea es un
comentario

Con .NET 6, el nuevo estilo para la clase
`Program` usa características recientes
de C# que nos permiten escribir sólo el
cuerpo del método `Main`
Esta característica se conoce como
“Instrucciones de nivel superior”



Aplicación HelloWorld generada

```
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.





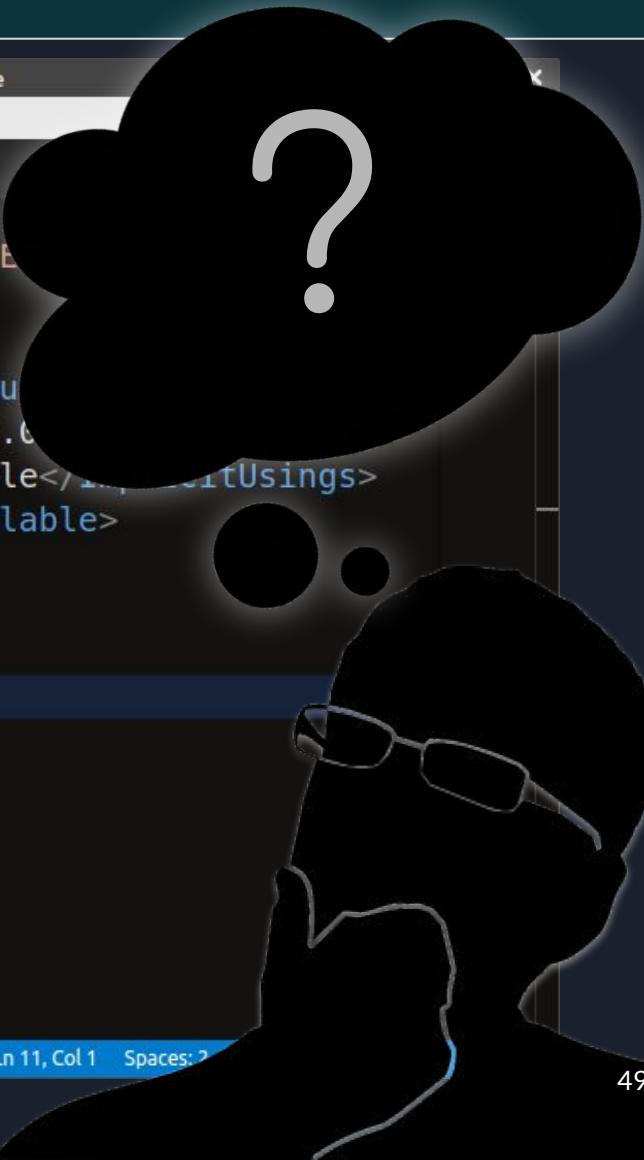
Editar teoria1.csproj. Cambiar enable por disable en ImplicitUsings

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net6.0</TargetFramework>
    <ImplicitUsings>enable</ImplicitUsings>
    <Nullable>enable</Nullable>
  </PropertyGroup>
</Project>
```





Presionar F5 para compilar y ejecutar

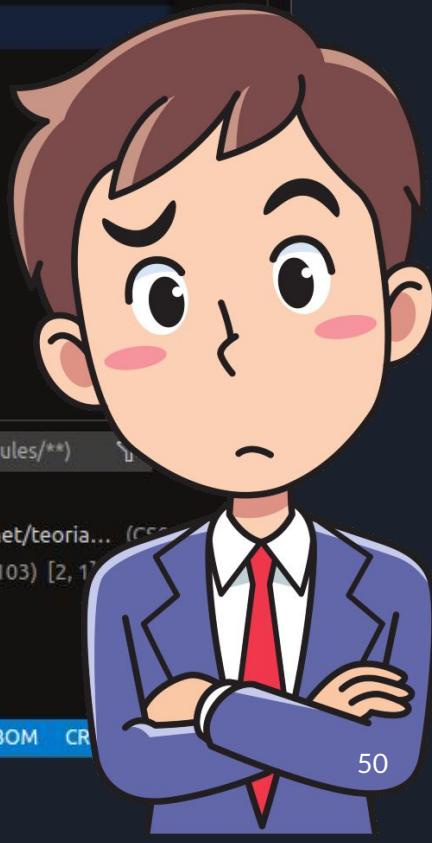


A screenshot of the Visual Studio Code interface showing a C# project named "teoria1". The "Program.cs" file is open, displaying the following code:

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net6.0</TargetFramework>
    <ImplicitUsings>enable</ImplicitUsings>
    <Nullable>enable</Nullable>
  </PropertyGroup>
</Project>
```

The "EXPLORER" sidebar shows the project structure with files ".vscode", "bin", "obj", "Program.cs", and "teoria1.csproj". The status bar at the bottom indicates ".NET Core Launch (console) (teoria1)" and "Ln 11, Col 1 Spaces: 2".

Error de compilación!



A screenshot of Visual Studio Code showing a compilation error. The code editor displays the following C# code:

```
// See https://aka.ms/new-console-template for more information
Console.WriteLine("Hello, World!");
```

The word "Console" is underlined with a red squiggly line, indicating a syntax error. A callout bubble points to this underlined text with the message: "El nombre 'Console' no existe en el contexto actual".

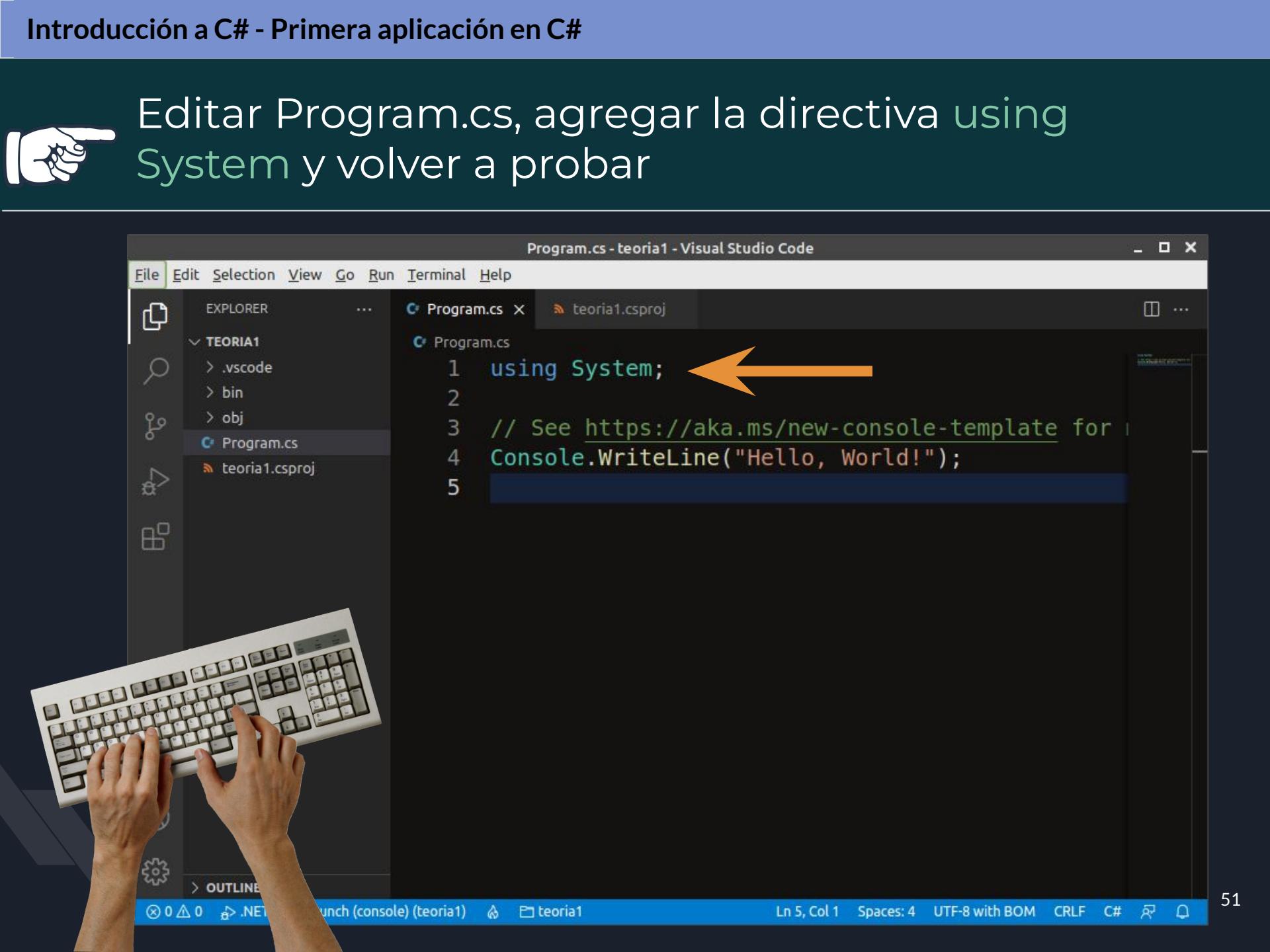
In the bottom left corner of the code editor, there are two error icons with the number "2". The Problems panel shows two errors:

- El nombre 'Console' no existe en el contexto actual [/home/leo/proyectosDotnet/teoria... (CS0103)]
- The name 'Console' does not exist in the current context [teoria1] csharp(CS0103) [2, 1]

The status bar at the bottom of the screen shows: "Ln 3, Col 1 Spaces: 4 UTF-8 with BOM CR".



Editar Program.cs, agregar la directiva using System y volver a probar



The image shows a person's hands typing on a white keyboard in front of a dark-themed Visual Studio Code interface. The code editor displays the 'Program.cs' file from a project named 'TEORIA1'. The 'using System;' directive is highlighted with an orange arrow. The code itself is as follows:

```
1 using System;
2
3 // See https://aka.ms/new-console-template for
4 Console.WriteLine("Hello, World!");
5
```

The status bar at the bottom of the VS Code window indicates: 'Ln 5, Col 1 Spaces: 4 UTF-8 with BOM CRLF C#'. The title bar of the window reads 'Program.cs - teoria1 - Visual Studio Code'.

Introducción a C# - Primera aplicación en C#

```
using System;  
  
// See https://aka.ms/new-console-template for more information  
Console.WriteLine("Hello, World!");
```





Modificar nuevamente Program.cs de la siguiente manera y probar nuevamente

The image shows a screenshot of Visual Studio Code with a dark theme. The file `Program.cs` is open, displaying the following code:

```
// See https://aka.ms/new-console-template for more information
System.Console.WriteLine("Hello, World!");
```

A callout arrow points from the text "Eliminar la directiva using System pero anteponer System. a la clase Console" to the word `System.` in the code. In the bottom-left corner of the screen, there is a graphic of a person's hands typing on a white keyboard.

Elminster la directiva
using System pero
anteponer System. a la
clase Console

Ln 3, Col 1 Spaces: 4 UTF-8 with BOM CRLF C# ⚙️ 🔍

Introducción a C# - Primera aplicación en C#

```
// See https://aka.ms/new-console-template for more information
System.Console.WriteLine("Hello, World!");
```



Directiva using

- El nombre **extenso** de una clase (o de cualquier otro tipo) lleva como prefijo el espacio de nombres en la que se ha definido.
- El nombre **extenso** de la clase **Console** es **System.Console** porque está definida en el espacio de nombres **System**.
- Si usamos la directiva **using System** al inicio del código podemos referirnos a cualquier tipo definido en el espacio de nombres **System** sin anteponer el prefijo **System**.

Directiva using

- El alcance de la directiva `using` es el archivo fuente donde se especifica.
- Con `C#10` (liberado con `.NET 6`) es posible utilizar la directiva `global using` para extender el alcance a todos los archivos fuente del proyecto.
- Si en el archivo de configuración del proyecto (extensión `.csproj`) colocamos:
`<ImplicitUsings>enable</ImplicitUsings>`
estamos habilitando las directivas de `using` implícitas

Directivas `using` implícitas (incorporadas en .NET 6)

El compilador agrega automáticamente un conjunto de directivas `using` en función del tipo de proyecto. En el caso de las aplicaciones de consola se incluye:,

- `using System;`
- `using System.IO;`
- `using System.Collections.Generic;`
- `using System.Linq;`
- `using System.Net.Http;`
- `using System.Threading;`
- `using System.Threading.Tasks;`

Otros tipos de aplicación incluyen más espacios de nombres que son comunes para esos tipos de aplicación.

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
sbyte	De -128 a 127	8 bits con signo	System.SByte
byte	De 0 a 255	8 bits sin signo	System.Byte
short	De -32.768 a 32.767	16 bits con signo	System.Int16
ushort	De 0 a 65.535	16 bits sin signo	System.UInt16
int	De -2.147.483.648 a 2.147.483.647	32 bits con signo	System.Int32
uint	De 0 a 4.294.967.295	32 bits sin signo	System.UInt32
long	De -9.223.372.036.854.775.808 a 9.223.372.036.854.775.807	64 bits con signo	System.Int64
ulong	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
float	De $\pm 1,5 \times 10^{-45}$ a $\pm 3,4 \times 10^{38}$	32 bits	System.Single	6 a 9 dígitos aprox.
double	De $\pm 5,0 \times 10^{-324}$ a $\pm 1,7 \times 10^{308}$	64 bits	System.Double	15 a 17 dígitos aprox.
decimal	De $\pm 1,0 \times 10^{-28}$ a $\pm 7,9228 \times 10^{28}$	128 bits	System.Decimal	6 a 9 dígitos aprox.

Demás tipos integrados en C#

Alias C#	Tipo de .NET
char	System.Char
bool	System.Boolean
string	System.String
object	System.Object
dynamic	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
- 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

```
// 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

```
// 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 las siguientes líneas

```
double d = 15.1;  
float f = 21.2;
```



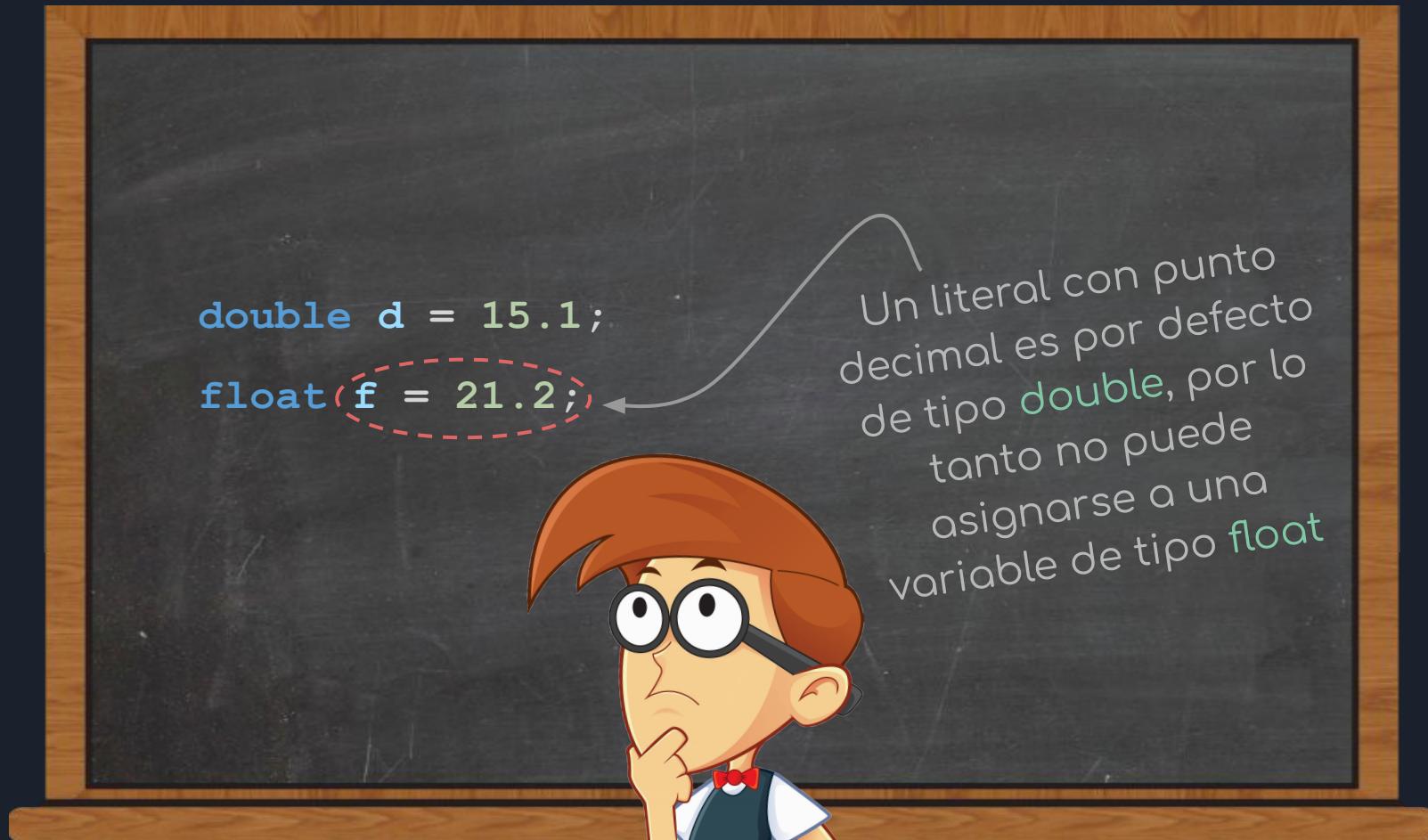


Codificar las siguientes líneas

```
double d = 15.1;  
float f = 21.2;
```

¿ Cuál es el
problema ?

Literales



Literales

```
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 ...



```
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?

```
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;`?



```
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?



```
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?



```
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?

```
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

```
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 integrada del Visual Studio Code o una terminal del sistema operativo

Introducción a C# - Ejercitación

File Explorer

OPEN EDITORS

Program.cs launch.json

.vscode > {} launch.json > ...

```
1    {
2     // Use IntelliSense to learn about this C# file!
3     // Hover to view descriptions of methods, fields, #region blocks, etc.
4     // For more information, visit: https://go.microsoft.com/fwlink/?linkid=808683
5     "version": "0.2.0",
6     "configurations": [
7       {
8         "name": ".NET Core Launch (code-interactive)",
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       "name": ".NET Core Attach",
20       "type": "coreclr",
21       "request": "attach",
22       "processId": "${command:pickProcess}"
23     }
24 }
```

Vamos a usar una terminal del sistema operativo.
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

.vscode > launch.json > ...

```
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 }
```



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

BREAKPOINTS

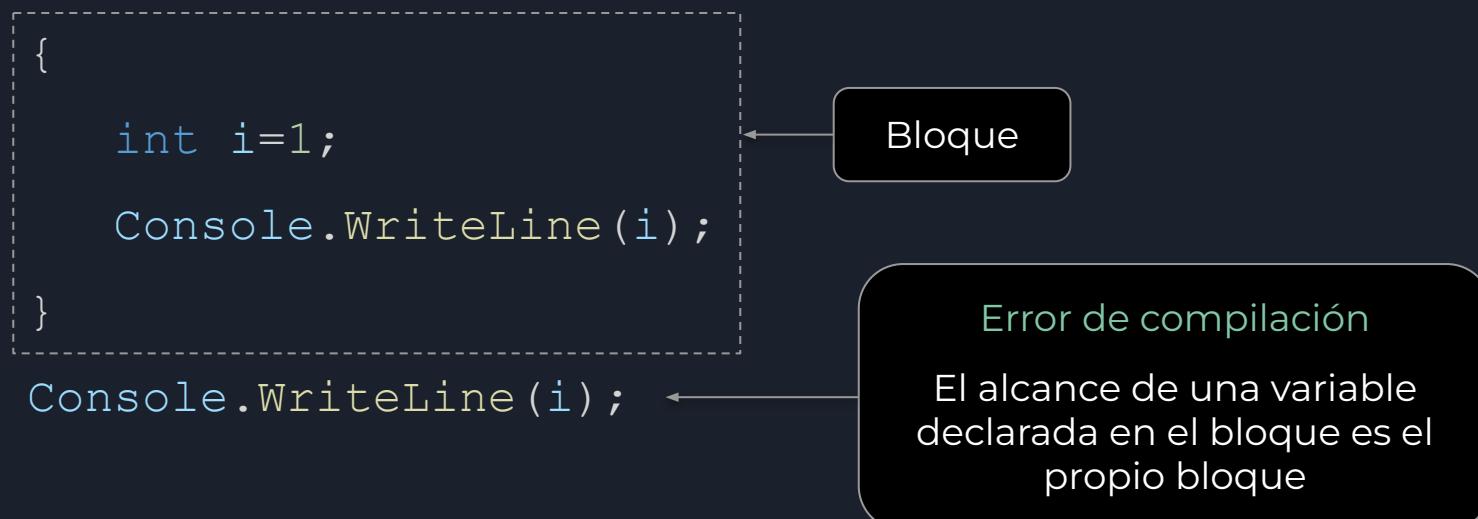
- Todas las excepciones
- Excepciones no controladas por ...

Add Configuration...

Estructuras de control

Estructura de control: Bloque

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



Estructura de control: Bloque

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

```
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



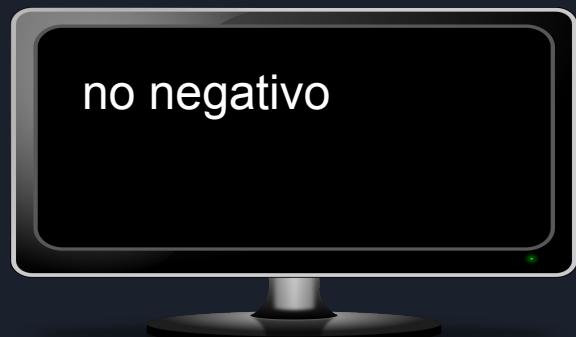
Operador condicional ?:

También conocido como operador condicional ternario.

```
condición ? consiguiente : alternativa
```

Ejemplo de uso:

```
int n = 10;  
  
string st = (n < 0) ? "negativo" : "no negativo";  
Console.WriteLine(st);
```

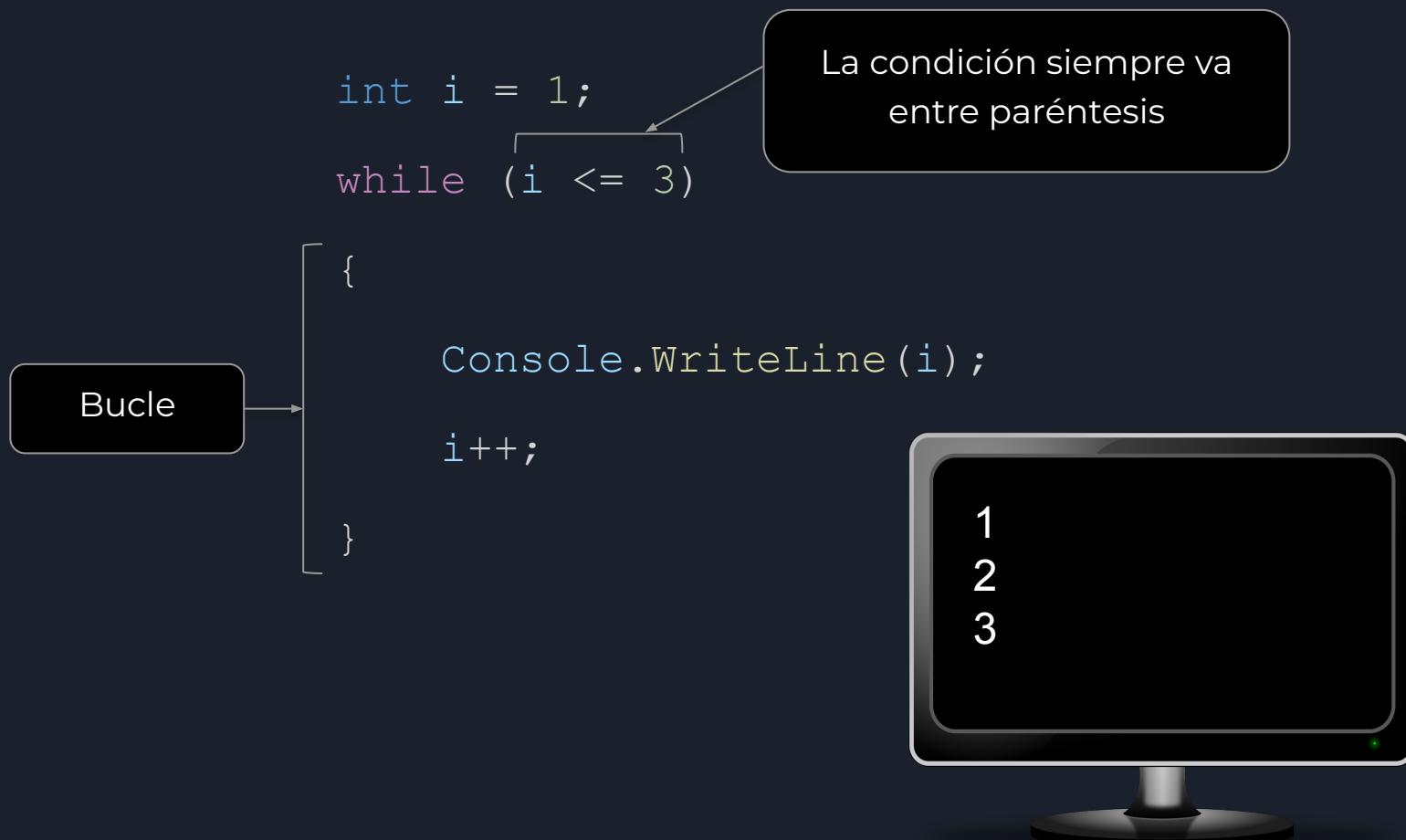


Estructura de control: Switch

```
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

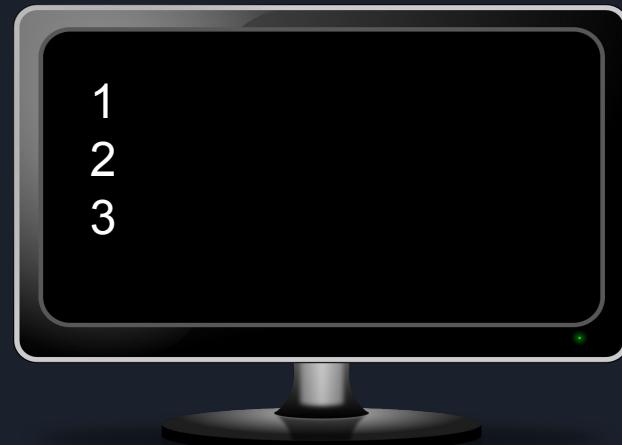


Estructura de control: do-while

```
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 >
}
```

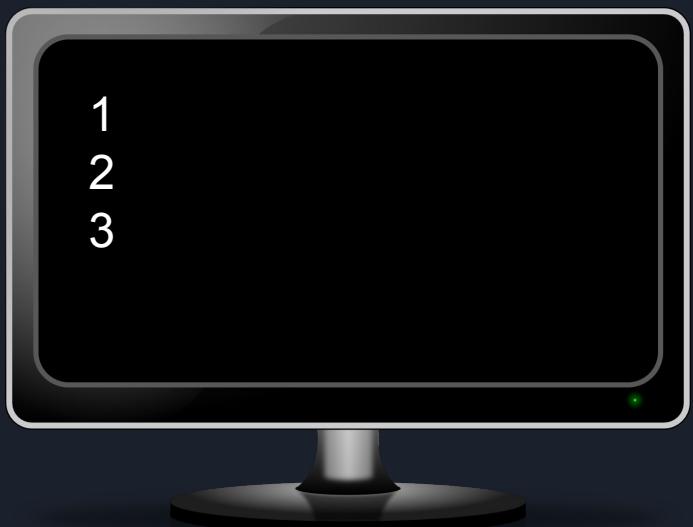
Es equivalente a:

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

Estructura de control: `for`

Ejemplo:

```
int i;  
for (i = 1; i <= 3; i++)  
{  
    Console.WriteLine(i);  
}
```



Estructura de control: for

Distintas maneras de hacer lo mismo

```
int i = 1;  
for (; i <= 3; i++)  
{  
    Console.WriteLine(i);  
}
```

```
int i = 1;  
for (; i <= 3; )  
{  
    Console.WriteLine(i++);  
}
```

```
int i = 1;  
for (; ; )  
{  
    Console.WriteLine(i++);  
    if (i > 3) break;  
}
```

Las secciones
del for pueden
omitirse

finaliza el for

Estructura de control: for

Alternativa

```
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

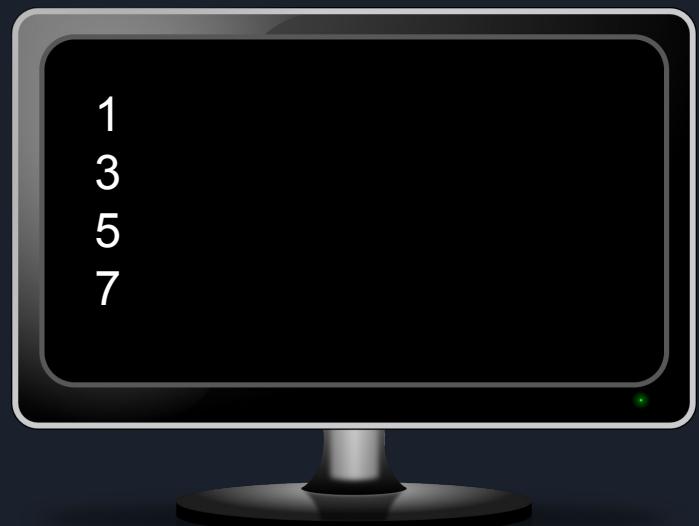
Interrupción de bucles

- **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:

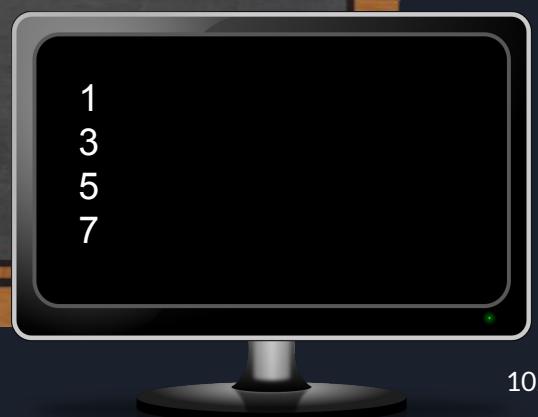
```
for (int i = 1; i <= 20; i++)
{
    if (i == 8)
        break;
    if (i % 2 == 0)
        continue;
    Console.WriteLine(i);
}
```



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



```
for (int i = 1; i <= 7; i += 2)
{
    Console.WriteLine(i);
}
```



```
1
3
5
7
```



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

```
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
de la primera
teoría

