

# FUTURE DEVELOPER WEEK

**Javier San Juan Cervera**



# CONTENIDO

## Día 1: Introducción y entorno de desarrollo

- ¿Qué es la programación?
- El entorno Visual Studio
- ¡Hola, mundo!
- Sentencias y expresiones
- Strings. Variables

¿QUÉ ES LA  
PROGRAMACIÓN?

# ¿QUÉ ES LA PROGRAMACIÓN?

- Un dispositivo informático (ordenador personal, smartphone, tablet, etc) puede ayudarte a hacer cosas muy interesantes y divertidas:
  - Navegar por Internet
  - Jugar contigo
  - Ayudarte a preparar un trabajo
  - Resolver problemas matemáticos
  - ¡Y muchas cosas más!

# ¿QUÉ ES LA PROGRAMACIÓN?

- Esto es debido a que existen aplicaciones que hacen todas estas tareas, pero en realidad puedes enseñarle a hacer muchas más cosas si aprendes cómo comunicarte con él y darle instrucciones.
- En eso consiste la programación, en **decirle a un ordenador qué debe de hacer.**

¿CÓMO FUNCIONA UN ORDENADOR?

# EL PROCESADOR

- Un ordenador es un aparato electrónico formado por varios componentes.
- Independientemente de si es un ordenador personal, un móvil, una tablet, o cualquier otro aparato informático, el cerebro del ordenador será un componente llamado **procesador**, que es capaz de ejecutar a mucha velocidad una serie de instrucciones muy simples.

# EL PROCESADOR

- Probablemente hayas oído ya que los ordenadores trabajan únicamente con **unos y ceros**. Con tan sólo estos dos valores, podemos indicarle al ordenador que haga todas esas cosas que es capaz de hacer.
- Pero, si sólo entiende los valores 1 y 0... ¿cómo podemos decirle, por ejemplo, que imprima por pantalla el valor **5**?
- ¡Vamos a empezar la semana aprendiendo cómo trabaja un ordenador!



# EL PROCESADOR

- En primer lugar explicaremos, de una forma muy básica (¡y no especialmente precisa!), cómo opera el procesador.
- El procesador está básicamente compuesto de una serie de piezas llamadas **transistores**.
- Cuando el ordenador está funcionando, puede estar pasando por estos transistores corriente eléctrica o no.

# EL PROCESADOR

- Cada poco tiempo (muy, muy poco tiempo), el procesador comprueba el estado de esos transistores. Los que reciben corriente, toman el valor 1. Los otros, el valor 0.
- Con la secuencia de unos y ceros que se forma, el procesador decide la instrucción a ejecutar y los datos que necesita.
- Por ejemplo, una determinada secuencia podría indicar que se realice una suma, mientras otra podría significar que emita un pitido.

# CÓDIGO BINARIO

- Y aunque queda lejos de los objetivos de esta semana, vamos simplemente a comentar que con la secuencia de unos y ceros podemos representar también cualquier número.
- Los seres humanos normalmente manejamos los números utilizando la **base decimal**, compuesta por 10 dígitos, del 0 al 9.
- Cuando definimos una cifra, por ejemplo el **13**, decimos que el dígito de la derecha son las **unidades**, y según nos movemos hacia la izquierda tenemos **decenas**, **centenas**...

# CÓDIGO BINARIO

- Una forma de descomponer el número sería, según nos movemos de derecha a izquierda, multiplicar cada dígito por  $10$  (porque estamos en base decimal) elevado a una potencia cada vez mayor, empezando por  $0$ , y sumar todos:

$$1 \cdot 10^1 + 3 \cdot 10^0 = 10 + 3 = 13$$

- Es decir, sumando dígitos en base decimal multiplicados por **potencias de 10**, podemos obtener cualquier valor que queramos.

# CÓDIGO BINARIO

- Para obtener un valor, el ordenador hace exactamente lo mismo, pero como sólo tiene los valores 0 y 1, no utiliza la base decimal, sino la **base binaria**, en la que descomponemos cualquier valor en una suma de **potencias de 2**.
- Si queremos representar el valor 13 en binario, lo descomponemos en:

$$8 + 4 + 1 = 13$$

# CÓDIGO BINARIO

- Que podemos representar también así:

$$13 = 1 \cdot 8 + 1 \cdot 4 + 0 \cdot 2 + 1 \cdot 1 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$$

- Con lo que obtenemos una serie de potencias de dos consecutivas sumadas. Las potencias que deben sumarse llevan un 1 delante. Las que no, un 0. Esa secuencia de unos y ceros es el valor binario del número. Es decir, el valor decimal 13 se codifica en binario de la forma:

1101

LENGUAJES DE PROGRAMACIÓN

# LENGUAJES DE PROGRAMACIÓN

- Aunque es posible escribir programas definiendo secuencias de unos y ceros (de hecho, los primeros programas se escribían así), es una tarea bastante engorrosa, porque el **lenguaje de la máquina** es muy diferente del **lenguaje humano**.
- Por eso se crearon los **lenguajes de programación**, que permiten escribir programas en un lenguaje más parecido al humano.



# LENGUAJES DE PROGRAMACIÓN

- Pero, si el ordenador sólo entiende unos y ceros, ¿cómo puede entender lo que escribimos con un lenguaje de programación?
- En realidad, no lo hace, pero utilizaremos un programa llamado **compilador** para convertir lo que escribimos en una secuencia de unos y ceros que el ordenador entienda.

# LENGUAJES DE PROGRAMACIÓN

- Cuanto más se acerque al lenguaje humano, diremos que es de más **alto nivel**. Cuanto más parecido sea al lenguaje máquina, será de más **bajo nivel**.
- También se clasifican en base a otros criterios: lenguajes funcionales, imperativos, estructurados, procedurales, orientados a objetos...
- En esta semana vamos a trabajar con C#, un lenguaje de Microsoft de **alto nivel, imperativo, estructurado, y orientado a objetos**.

# EL ENTORNO VISUAL STUDIO

# VISUAL STUDIO

- Visual Studio es un entorno de desarrollo integrado para la creación de aplicaciones para plataformas Windows.

Permite:

- Escribir programas tanto en modo texto (llamadas aplicaciones de consola) como gráfico (con controles visuales como ventanas, menús, botones, etc).
- Diseñar la interfaz de las aplicaciones en modo gráfico de forma intuitiva.
- Detectar y corregir errores en los programas.
- Etc.
- Soporta varios lenguajes de programación, como C++, C# y Visual Basic.

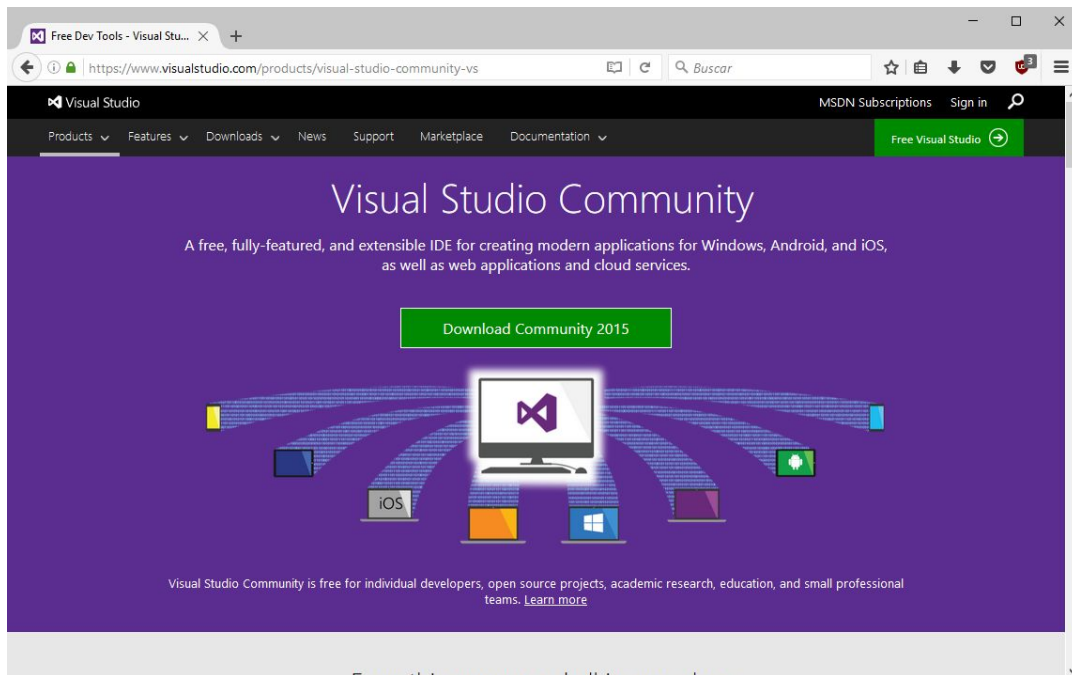
INSTALACIÓN Y ARRANQUE

# DESCARGA

- Podemos descargar Visual Studio de su página web:  
<http://www.visualstudio.com>.
- Existen múltiples versiones de Visual Studio disponibles. Algunas de ellas son gratuitas, como la versión Express o la Community.
- La versión [Express for Desktop](#) es una versión reducida, sin algunas características utilizadas en el ámbito profesional.
- La versión [Community](#) es una versión gratuita de la versión profesional, con restricciones en su uso para las grandes corporaciones.

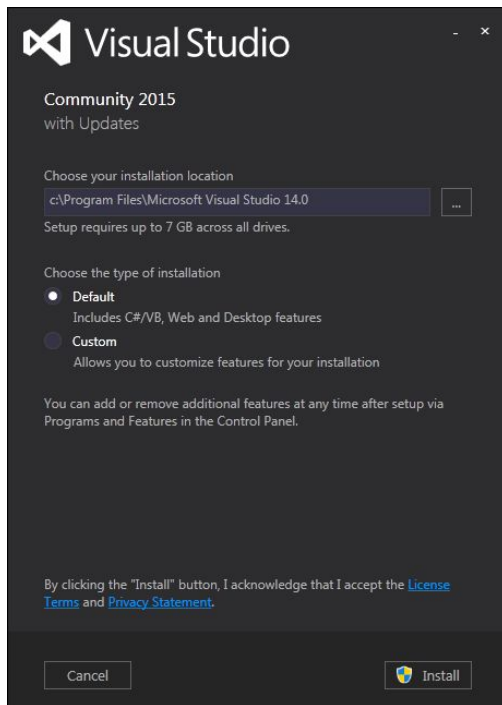
# DESCARGA

Se recomienda obtener **Visual Studio Community 2015**:



# INSTALACIÓN

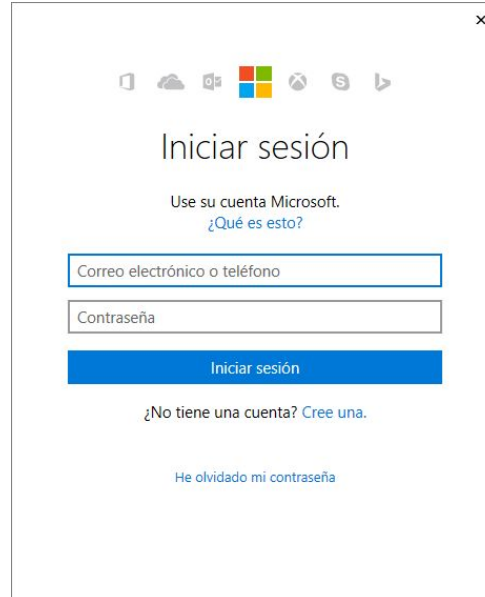
Una vez descargado, podemos arrancar el programa de instalación, que nos mostrará la siguiente pantalla:





# ARRANQUE

Tras la instalación, arrancaremos el programa y nos dará la opción de **iniciar sesión** con nuestra cuenta Microsoft (Hotmail, Outlook...). Esto nos permite guardar la configuración del entorno en la nube y utilizarla en todos nuestros equipos:

A screenshot of the Microsoft login window. At the top, there is a row of icons: a folder, a cloud, a document, the Windows logo, a game controller, a speech bubble, and a play button. Below the icons, the text "Iniciar sesión" is centered. Underneath, it says "Use su cuenta Microsoft." followed by a link "¿Qué es esto?". There are two input fields: the first is labeled "Correo electrónico o teléfono" and the second is labeled "Contraseña". Below these fields is a blue button labeled "Iniciar sesión". At the bottom, there is a link "¿No tiene una cuenta? Cree una." and another link "He olvidado mi contraseña". The window has a close button (X) in the top right corner.

Iniciar sesión

Use su cuenta Microsoft.  
[¿Qué es esto?](#)

Correo electrónico o teléfono

Contraseña

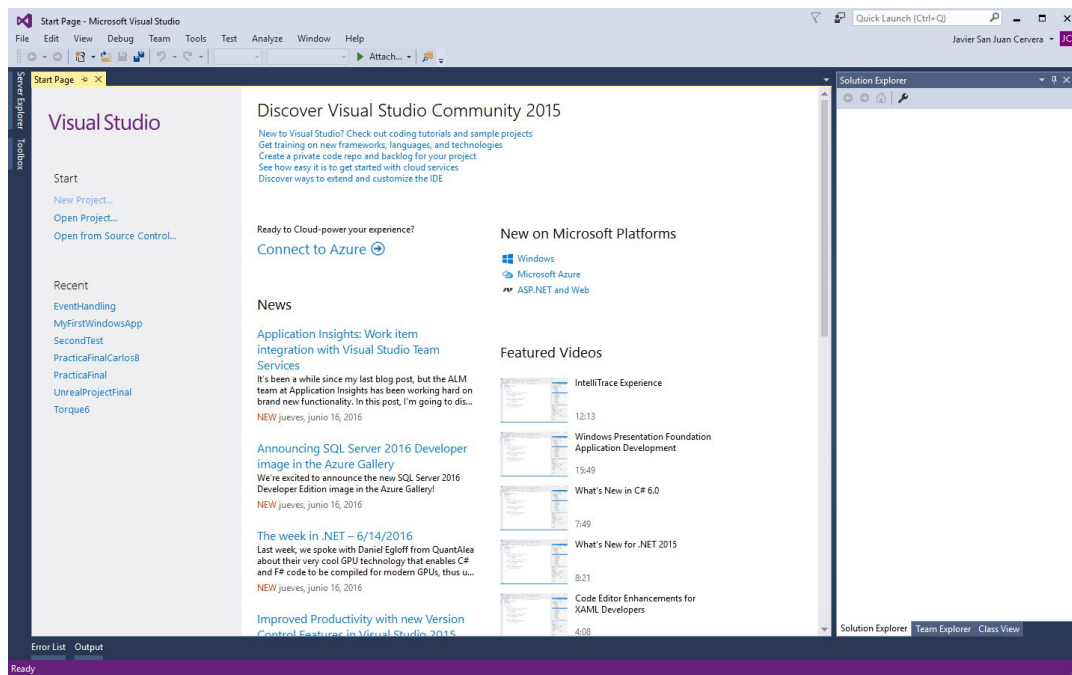
Iniciar sesión

[¿No tiene una cuenta? Cree una.](#)

[He olvidado mi contraseña](#)

# ARRANQUE

Finalmente, se nos mostrará la pantalla de inicio:



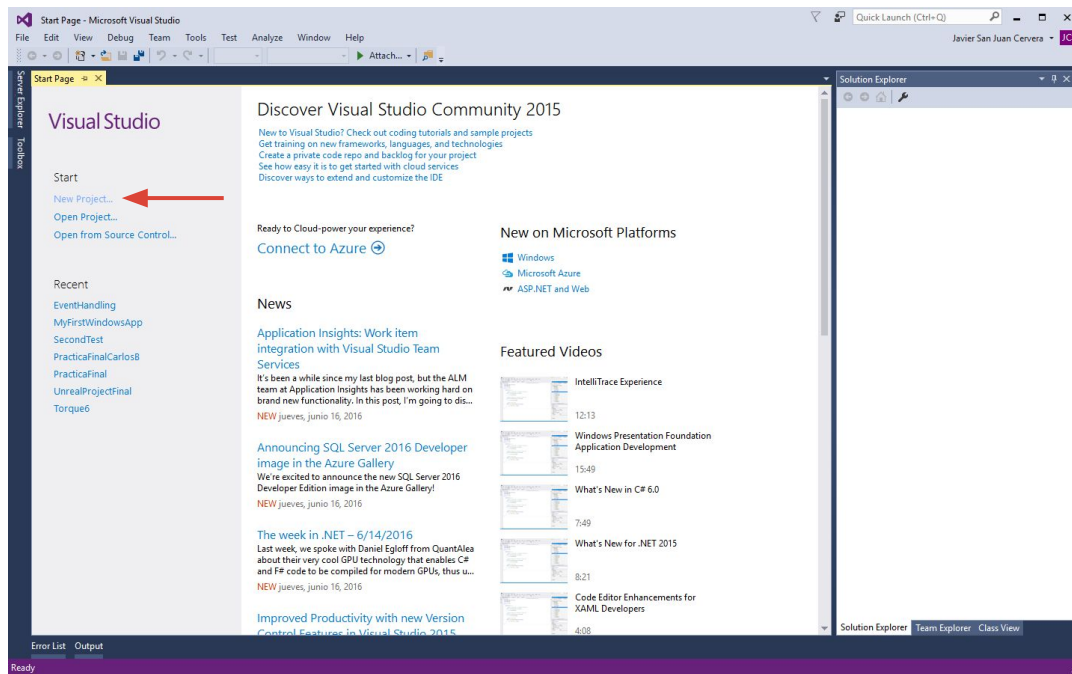
SOLUCIONES Y PROYECTOS

# SOLUCIONES Y PROYECTOS

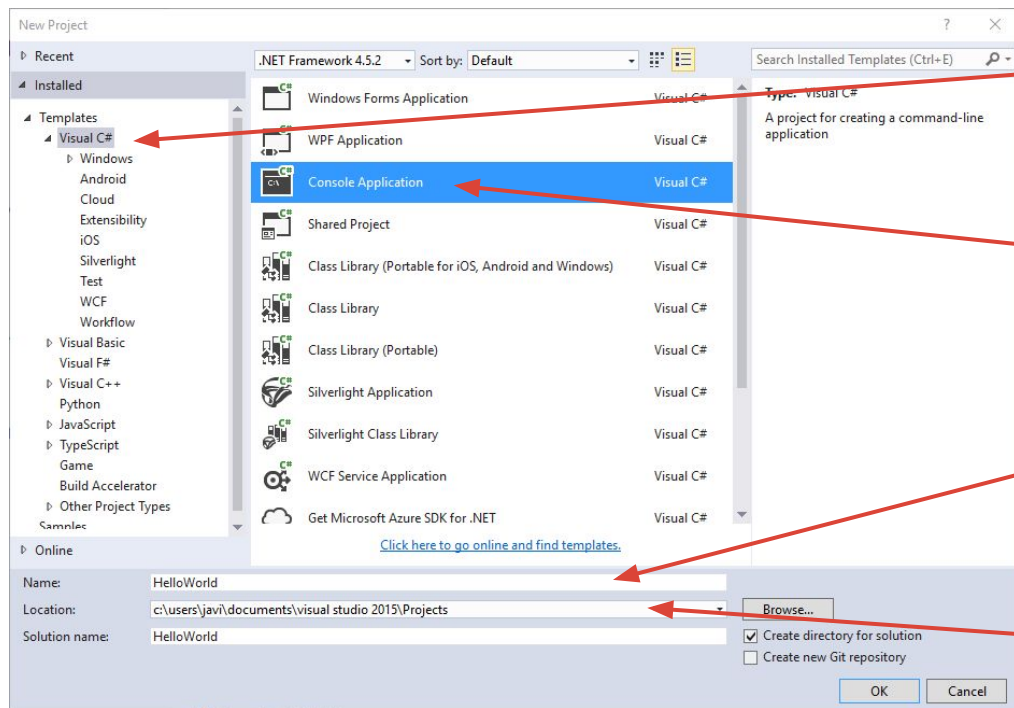
- Cada aplicación que queramos desarrollar requiere de una serie de archivos de configuración, código, iconos y otros recursos.
- Todos estos archivos se agrupan bajo un **proyecto**.
- Podemos tener varios proyectos relacionados, por ejemplo:
  - Aplicación principal
  - Aplicación de pruebas
  - Versión demo
- Varios proyectos se pueden agrupar en una **solución**.

# CREAR UN PROYECTO

Desde la pantalla de inicio de Visual Studio, podemos crear un nuevo proyecto:



# CREAR UN PROYECTO



Lenguaje: Elegimos **Visual C#**

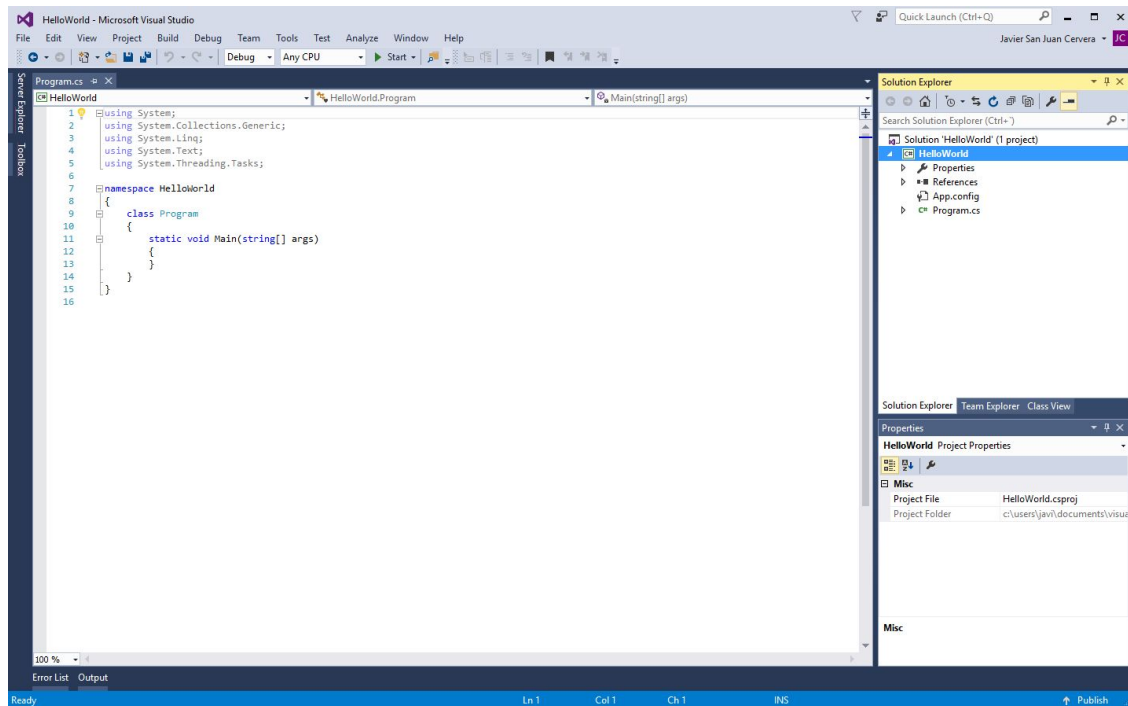
Tipo de aplicación: Hoy seleccionaremos **Console Application**

Nombre del proyecto

Ruta donde guardar el proyecto

# CREAR UN PROYECTO

Una vez creado, el proyecto se abrirá en Visual Studio:



¡HOLA, MUNDO!



# ¡HOLA, MUNDO!

- Existe una tradición entre programadores: cuando se aprende un nuevo lenguaje de programación, la primera tarea que el programador aprende a realizar en ese lenguaje es imprimir el mensaje “**¡Hola, mundo!**” en la pantalla.
- Así que nuestra tarea de hoy será hacer un programa en C# que imprima dicho mensaje.

# ¡HOLA, MUNDO!

Al generarse el proyecto **HelloWorld**, se nos ha abierto el fichero de código principal donde escribiremos nuestro programa. El fichero tiene varias líneas ya escritas. Vamos a fijarnos en las siguientes:

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

# ¡HOLA, MUNDO!

No es el momento de explicar el significado de la mayoría de líneas que aparecen en el programa, pero simplemente diremos que entre el símbolo de apertura de llave ({) y el de cierre (}) que aparecen debajo de la línea `static void Main(string[] args)`, escribiremos las instrucciones que debe realizar nuestro programa.

# ¡HOLA, MUNDO!

Para que escriba el mensaje que queremos, pondremos la siguiente línea entre las llaves:

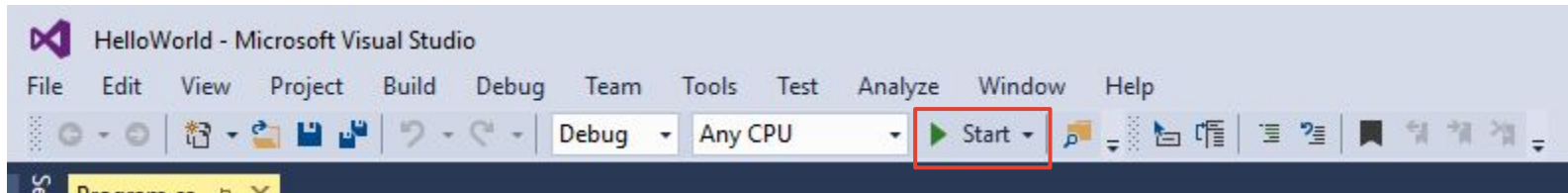
```
Console.WriteLine("¡Hola, mundo!");
```

Es importante escribir la línea tal y como aparece aquí representada, con todos sus símbolos. Esta instrucción le dice al ordenador que escriba la línea **"¡Hola, mundo!"** (excluyendo las comillas) en la consola (esa ventana con texto sobre fondo negro que veremos en un momento).

EJECUTAR EL PROGRAMA

# EJECUTAR EL PROGRAMA

- Cuando estamos escribiendo un programa, es posible ponerlo en marcha para ver que hace lo que esperamos. A esto se le llama **ejecutar** el programa.
- Podemos ejecutar el programa con la opción de menú **Debug / Start Debugging**, o bien con el botón **Start** de la barra de herramientas:



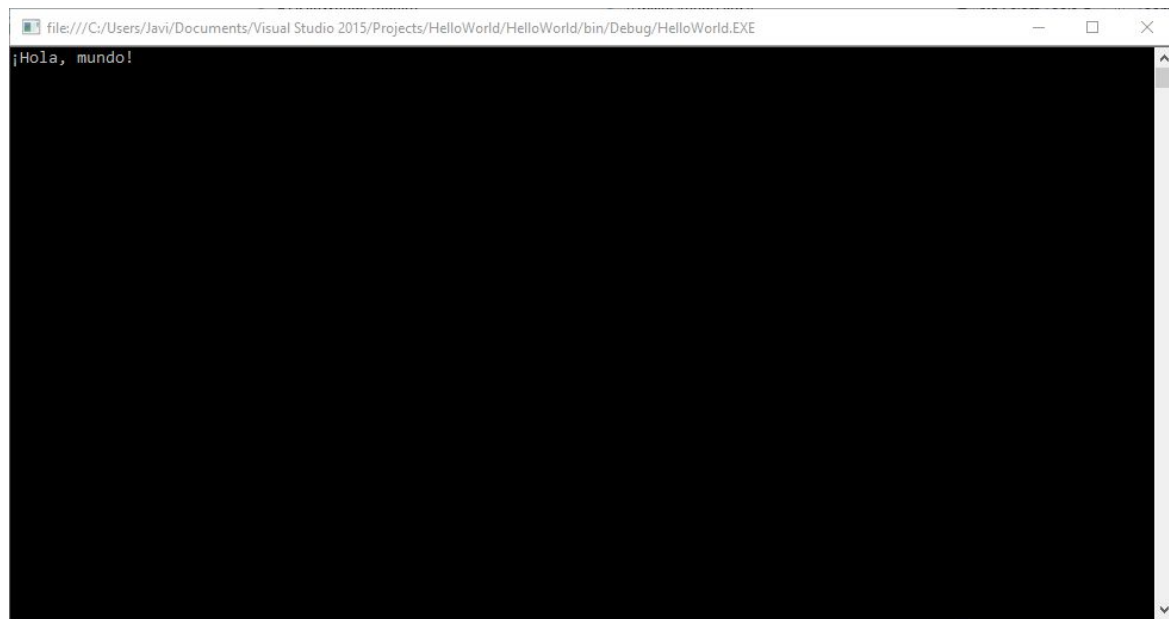
# EJECUTAR EL PROGRAMA

- Si ejecutamos el programa, veremos que una ventana se abre y **se cierra inmediatamente**.
- Esto es porque, justo después de escribir "¡Hola, mundo!", las instrucciones a ejecutar terminan y el programa se cierra.
- Para solucionarlo, podemos colocar después una instrucción que **ponga el programa en espera** hasta que se pulse una tecla:

```
Console.ReadKey();
```

# EJECUTAR EL PROGRAMA

Si ejecutamos el programa de nuevo, veremos que la ventana de la consola permanece abierta hasta que pulsemos una tecla:





# SENTENCIAS Y EXPRESIONES

# SENTENCIA

- Volvamos de nuevo al programa que acabamos de escribir. Hemos añadido las siguientes instrucciones:

```
Console.WriteLine("¡Hola, mundo!");  
Console.ReadKey();
```

- En los lenguajes de alto nivel, en lugar de instrucciones, hablaremos de **sentencias**. En C#, las sentencias terminan con el símbolo **punto y coma (;)**.

- ¿CUÁNTAS SENTENCIAS  
TIENE NUESTRO  
PROGRAMA?
- ¿QUÉ HACE CADA UNA?

# EXPRESIÓN

- Una expresión es una operación que produce un valor como resultado.
- Por ejemplo, la expresión `5 + 2` produce como resultado el valor 7.
- Igualmente, `5` y `2` son expresiones que producen, respectivamente, los valores 5 y 2.
- Se puede usar el valor de una expresión para encadenarlo a otra expresión y producir un nuevo resultado.

# EXPRESIÓN

- Esto es lo que se ha hecho en el caso anterior:
  - El **operador** suma (+) obtiene el valor de la expresión de su izquierda y lo suma al valor de la expresión de la derecha, produciendo un nuevo resultado.
  - Como valor de la expresión de la izquierda, se obtiene el valor **5**.
  - Como valor de la expresión de la derecha, se obtiene el valor **2**.
  - Como resultado de la suma, se obtiene el valor **7**.
- Este resultado se podría encadenar a una nueva expresión.

¿CUÁNTAS EXPRESIONES HAY  
AQUÍ, Y QUÉ RESULTADO SE  
OBTIENE?:

$$5 + 2 + 1$$

# TIPOS

- Las expresiones producen valores de un tipo concreto.
- Por ejemplo, las expresiones anteriores producen un resultado de **tipo entero**, es decir, un valor numérico sin decimales.
- La expresión `5.5 + 2.5` produce como resultado `8.0`, que es de **tipo decimal**, es decir, un valor numérico con parte decimal.
- Es posible imprimir el valor de una expresión con `Console.WriteLine`.

STRINGS. VARIABLES



# STRINGS

- Las expresiones no tienen por qué ser de tipo numérico.
- Como dijimos en el apartado anterior, es posible imprimir una expresión con `Console.WriteLine`.
- Cuando escribíamos el mensaje “¡Hola, mundo!”, lo que estábamos haciendo era, precisamente, escribir una expresión de **tipo texto**.
- A estas expresiones se les llama **strings** (“cadenas” en inglés) en la mayoría de lenguajes de programación, por estar formado por cadenas de caracteres.

# STRINGS

- Los strings van siempre **entre comillas**. Así, `5` es una expresión de tipo entero, mientras que `"5"` es una expresión de tipo string.
- Al igual que con los números, podemos usar la **operación suma** con strings. El resultado de la suma será un nuevo string con **ambos strings concatenados**.
- Por ejemplo, la expresión `"Hola," + " mundo"` produce el string `"Hola, mundo"`.

- ¿QUÉ SE ESCRIBIRÁ EN PANTALLA SI IMPRIMIMOS LA EXPRESIÓN  $5 + 2$ ?
- ¿Y SI IMPRIMIMOS LA EXPRESIÓN “5” + “2”?
- ¿Y LA EXPRESIÓN “5” + 2?

# CONVERSIÓN DE TIPOS

- Es probable que no hayamos acertado al responder a la última pregunta.
- Es posible **mezclar distintos tipos** de datos en una operación.
- Por ejemplo, si un valor entero se suma a un decimal, **el entero se convierte en decimal** y se hace la suma entre dos decimales.
- Y si un tipo numérico se suma a un string, **el número se convierte en un string** y se concatenan ambos.

VARIABLES

# VARIABLES

- Antes de entrar en la explicación de lo que son las variables, vamos a ir desarrollando un caso en el que eventualmente se va a hacer necesario su uso.
- En el arranque del programa, vamos a pedir al usuario su nombre, y saludará utilizando el nombre introducido.
- Si recordamos, anteriormente habíamos pausado el programa hasta que se pulsase una tecla utilizando `Console.ReadKey()`.

# VARIABLES

- De la misma forma, podemos permitir al usuario introducir un texto (no un sólo carácter) utilizando `Console.ReadLine()`.
- Esto permitirá introducir caracteres con el teclado hasta que se pulse la tecla **Enter**, y entonces producirá una expresión con el string introducido.
- Vamos a intentar modificar la aplicación para que salude utilizando el nombre introducido por el usuario.

# VARIABLES

- Probablemente hayamos hecho algo como esto:

```
Console.WriteLine("¡Hola, " + Console.ReadLine() + "!");
```

- Muy bien, ahora vamos a hacer que escriba otra línea, en la que se despida del usuario antes de terminar la aplicación. Podríamos hacer algo así:

```
Console.WriteLine("Adiós, " + Console.ReadLine());
```

- Pero, ya habíamos introducido el nombre antes, ¿no queremos que el usuario tenga que escribirlo de nuevo!



# VARIABLES

- Aquí es donde entran en juego las **variables**. Una variable permite **almacenar el valor de una expresión**. Una variable no es más que un espacio en la memoria del ordenador, y se llama variable porque podemos **cambiar el valor** que contiene, aunque una variable sólo puede guardar valores de un determinado tipo.
- Para utilizar una variable, debemos crearla previamente en una sentencia, indicando el tipo de datos que puede contener, y su nombre, que utilizaremos para referirnos a ella.
- El nombre debe empezar por una letra o el símbolo de guión bajo (\_), y puede estar seguido de estos mismos caracteres o un número.
- No se admiten espacios en el nombre.
- Se pueden utilizar mayúsculas y minúsculas, y se distingue entre una y otra (es decir, `miVariable`, `MiVariable` y `MiVaRiaBle` son variables diferentes).

# VARIABLES

- Por ejemplo, podemos crear una variable donde guardar un valor entero así:

```
int unEntero;
```

- Otra para un valor decimal:

```
double unDecimal;
```

- Otra para un string:

```
string unTexto;
```

# VARIABLES

- Para asignar un valor a la variable, utilizamos el operador `=`.
- Por ejemplo, para asignar valores a las variables anteriores:

```
unEntero = 5 + 2;  
unDecimal = 2.5;  
unTexto = Console.ReadLine();
```

- También se puede asignar el valor al crear la variable:

```
string unTexto = Console.ReadLine();
```

¿CÓMO MODIFICAMOS EL PROGRAMA  
PARA QUE SALUDE AL USUARIO, SE  
DESPIDA DE ÉL, PERO LE PIDA UNA  
SOLA VEZ SU NOMBRE?