**FUNDAMENTOS VISUAL STUDIO**

Software:

* **Visual Studio Community** (IDE)

[Visual Studio 2022 Community Edition: descargar la versión gratuita más reciente](https://visualstudio.microsoft.com/es/vs/community/)

Lo descargamos y, en la instalación, debemos seleccionar lo siguiente:



Mi entorno está en inglés. Podéis perfectamente descargar también el paquete de idioma

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

* Cuenta de **GitHub**. GitHub es un repositorio de código. Es de Microsoft, pero no tiene código solo de Microsoft, contiene de todo, desde código Front, React, Angular o Java que es Back.

Si no tenemos cuenta, nos hacemos una cuenta para poder trabajar con nuestros proyectos.

[GitHub · Build and ship software on a single, collaborative platform · GitHub](https://github.com/)

* **SQL Server Express**: Es la base de datos de Microsoft. Al instalar, seleccionamos Basic y ya está.

[Descargas de SQL Server | Microsoft](https://www.microsoft.com/es-es/sql-server/sql-server-downloads)



* Después de instalar SQL Server Express, nos pedirá, en la misma instalación, al finalizar, instalar **SQL Server Management Studio**.

Lo instalamos también con siguiente 🡪 siguiente.

Temario: **NET CORE LENGUAJE C#**

* **Windows Forms**. Será nuestro entorno de trabajo Visual. Iremos creando múltiples formularios para entender cada característica del lenguaje.
* **Fundamentos del lenguaje**: Iteraciones, sintaxis, Clases, POO, Colecciones.
* **Inyección de dependencias**:Nos permite tener todas las clases y elementos administrados en un solo punto de un proyecto.
* **Ado Net**:Es la tecnología de acceso a datos de Microsoft (antigua). Se sigue utilizando, pero le ha ganado la batalla otra tecnología.
* **LINQ y Entity Framework**: Es la tecnología de acceso a datos actual (2024) de Microsoft.

El temario que tenéis delante es un temario que se llama Net Framework. Es un temario dónde se hacían solamente aplicaciones para Windows y solo con Windows.

¿Qué es Net Core? Esta tecnología nos permite crear aplicaciones del fabricante Microsoft para distintas plataformas.

Con una aplicación, puede distribuirla en un Windows, Linux un MacOS.

Si montamos una aplicación Web, podemos ponerla en un servidor Apache, IIS, Nginx.

Todo lo que hagamos, estará listo para producción, es decir, cuando compilemos un programa, podremos utilizarlo en cualquier PC, sin necesidad del programa de Visual Studio.

Por ejemplo, si hacemos una calculadora, podría ejecutarla directamente sin necesidad de Visual, simplemente ejecutando su EXE.

Tenemos diferentes proyectos dentro de Visual Studio.

* Librerías: Son proyectos dónde solamente tenemos funcionalidades, no tenemos parte gráfica, solo lógica.
* Visuales (Forms): En nuestro caso, tenemos proyectos de Windows, pero, también tenemos proyectos Web visuales o proyectos para móviles.

**Metodología de trabajo**

Tendremos un solo proyecto y múltiples formularios.

Iremos generando más proyectos a medida que vayamos aprendiendo, pero un solo proyecto para cada característica.

GITHUB será el centro de trabajo para vosotros y para mí.

Vamos a crear un nuevo proyecto llamado **FundamentosOctubreNetCore** de tipo **Windows Forms**.

**Nota:** Aunque no tengamos todavía Visual Studio instalado, no pasa nada, lo vamos haciendo a lo largo de la clase y podéis coger mis recursos.

Podemos clonar un repositorio de GitHub (Paco). Si todavía no lo tenemos instalado, podemos recuperar lo que haya puesto el profe en la nube.

Interfaz de usuario gráfica, Texto, Aplicación, Word

Descripción generada automáticamente

Dentro de GitHub, simplemente copiamos el código que pone **.git**

Debemos estar seguros que hemos seleccionado el lenguaje **C#**.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

A continuación, nos pedirá un nombre y una ubicación.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Lo siguiente que nos pregunta son las librerías de Net Core a utilizar.

La última es la versión 8.0.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Voy a subir la documentación y el nuevo proyecto a GitHub para que podáis tenerlo en casa.

Sobre la esquina inferior derecha, seleccionamos **Add to Source Control**

Interfaz de usuario gráfica, Aplicación, Teams

Descripción generada automáticamente

Nos abrirá una ventana para poder subir el proyecto a GitHub

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Y podremos visualizar nuestro proyecto subido a la nube de GitHub.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Mi proyecto de Github está en la siguiente dirección

<https://github.com/serraguti/FundamentosOctubreNetCore/>

Una vez que tenemos los elementos subidos en Github, debemos descargar los posibles cambios que vayamos haciendo en el programa y los cambios que irá haciendo Paco en el proyecto.

Tenemos dos opciones:

1. **Descargar todo el proyecto de GitHub desde CERO**.

Desde la página de GitHub copiamos el código GIT.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Clonamos el repositorio desde Visual Studio

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Le damos un nombre al proyecto para almacenar la carpeta.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

1. **Implementar los cambios** que se han realizado en el proyecto, una vez que lo hemos descargado.

En este caso, suponemos que ya tenemos el proyecto descargado. Dicho proyecto, desde el origen, ha tenido cambios y YO también he realizado cambios.

Por un lado, está mi proyecto (Alumno) y, por otro lado, está el proyecto de Profesor (Origen)

Debemos ir a una ventana llamada **Git Changes**

Ponemos nuestro mensaje (lo que sea) y pulsamos en **Commit All**



Una vez que hemos salvado todo, indicamos **Pull** sobre la flecha

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

1. **Cuando creemos un formulario, con el nombre que diga Paco, pues le agregamos nuestro nombre o nuestras iniciales y así tenemos ficheros distintos, uno para mis cosas y otro para la explicación y los Pull y así no entran en conflicto.**

Es lo mismo, debemos hacer un Commit para guardar los cambios en local y después, hacer un **Pull**

# IDE VISUAL STUDIO

**Importante:** Todas las ventanas que voy describiendo a continuación, estarán en el Menú **View/Ver**

* **App Principal:** Es la parte central de nuestra Aplicación, en nuestro ejemplo es un Formulario.

No podemos editar el formulario ni nada si estamos en ejecución.

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente con confianza media

* **Explorador de soluciones/Solution Explorer**: El explorador de soluciones nos permitirá acceder a nuestras clases y características que tengamos en nuestra App. Es decir, accedemos a los ficheros del proyecto.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

* **Barra de herramientas/ToolBox**: Solamente vamos a visualizar la barra de herramientas cuando estemos en MODO DISEÑO/DESIGN

En dicha barra, tendremos todos los controles para diseñar nuestro formulario.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

* **Propiedades/Properties:** Solamente se visualiza en modo Diseño/Design y nos permite modificar el aspecto Visual de los controles.

Los nombres de los controles (Button, Label, TextBox) debemos indicarlos si vamos a trabajar con ellos en código. Modificaremos su propiedad **name** para acceder a ella por código.

Dependiendo del control, asignaremos un nombre u otro

Ejemplo:

Button: **btnDescripcion** de la acción.

TextBox: txtDescripcion del contenido de la caja

Label: lblDescripcion de lo que tenga el Label.

En este lenguaje, se diferencia mayúsculas de minúsculas.

**Nota:** Ordenamos la ventana de propiedades de forma alfabética para visualizar mejor todo por su nombre.

Interfaz de usuario gráfica, Tabla

Descripción generada automáticamente

Por ejemplo, si vamos a poner un botón para Pulsar para algo

**Name: btnPulsar**

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Otro ejemplo con una Caja de texto

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Por otro lado, tenemos el código lógico, es decir, qué queremos hacer con los dibujos.

Este tipo de arquitectura de Windows Forms funciona y está orientada a eventos. Un evento es un **CUANDO** queremos hacer algo.

Dentro de dicho CUANDO es dónde escribiremos.

Para recuperar un EVENTO, necesitamos realizar **doble click** sobre el control que deseemos, por ejemplo, en este caso, en un botón.

Texto, Aplicación

Descripción generada automáticamente

**CONTROL DE ERRORES VISUAL STUDIO**

Existen multitud de errores dentro de un programa.

Vamos a comenzar por visualizar el más sencillo y cómo podemos solucionarlo con nuestro entorno gráfico.

**Error de compilación**. Dicho error sucede cuando tenemos algo mal escrito en nuestro código y no nos permite compilar. Compilar es ejecutar el proyecto.

Si tenemos un error, siempre diremos que NO en la ventana que nos aparecerá.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Estamos en un proyecto, para poder ejecutar el proyecto, todo debe estar bien escrito, no importa el lugar dónde esté escrito. Si tenemos errores, no podremos continuar.

Al pulsar en NO, vemos que nos abre una ventana de errores en la parte inferior.

Dicha ventana de errores nos irá llevando a todos los errores que tengamos en el proyecto para poder ejecutarlo

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Bastará con pulsar sobre CADA ERROR y solucionarlo.

# CODIGO DE OBJETOS EN EVENTOS

Cuando escribamos código dentro de un Evento (Click), debemos acceder a los objetos del formulario con la palabra clave **this**

Por ejemplo, tenemos una caja de texto ahora mismo y para acceder a ella en nuestro código, pondremos this.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Tipos de propiedades en objetos y clases. Todo es una clase, pero hablamos ahora mismo de clases gráficas.

Para saber si estamos accediendo a una propiedad, debemos mirar la llave inglesa.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Tenemos varios tipos de propiedad en código. Por ahora, nos da igual lo que hagan dichas propiedades.

1. **Propiedades de Tipos primitivos (números, letras):** Podemos representar directamente los números o las letras. Las letras se escriben entre comillas dobles y todo termina con punto y coma.

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

1. Propiedades de **New Object:** No sabemos el tipo que tiene la propiedad (ni idea). Lo que se escribe es **new** y lo que nos diga la ayuda de Visual Studio.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente



1. **Propiedades enumeradas:** Son “herramientas” para el lenguaje y el programador. Son una ayuda dónde me ofrecen todas las posibilidades que la propiedad necesite.

**Lo único que tenemos que visualizar es que sean de color AMARILLO.**

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente



### DECLARACION DE VARIABLES TIPOS PRIMITIVOS

Una variable es un espacio de memoria para almacenar algún dato de nuestro programa.

Por ejemplo, podemos pedir dos números al usuario y mostrarle la suma.

En cualquier programa, necesitamos almacenar elementos:

1. Necesitamos almacenar el numero 1.
2. Necesitamos almacenar el numero 2
3. Necesitamos almacenar la suma de los dos numeros

Declaración de las variables se realiza con la siguiente sintaxis:

TIPODATO nombreVariable;

Por ejemplo:

string texto = “Hola mundo”;

int numero = 17;

Vamos a comenzar viendo la teoría de los tipos primitivos. Un tipo primitivo o Wrapper se iguala directamente a un valor para poder ser utilizado.

El valor puede ser representado gráficamente.

PRIMITIVOS

* **char**: Representa un carácter. Se escribe entre comillas simples.
* **byte**: Almacena números de 0-255.
* **short**: Almacena números hasta 27.900
* **int**: Almacena más precisión
* **long**: El mayor número de precisión de un entero.
* **float, double, decimal**: Representan números decimales
* **bool**: true/false
* **DateTime**: Representa fechas.
* **object**: En realidad es cualquier elemento que tengamos dentro de Visual Studio. Podemos almacenar cualquier clase dentro de esta variable.

**CONVERSION DE TIPOS**

Dentro de cualquier lenguaje de programación, necesitamos almacenar valores según su tipo.

En ocasiones, dichos valores NO vienen en el formato que necesitamos para trabajar, por ejemplo, un número podría venir como texto, por lo que tendríamos que convertirlo para poder trabajar con operaciones matemáticas.

**Conversiones automáticas**

Estas conversiones se realizan cuando el tipo de dato a almacenar es **menor** que el tipo que estamos utilizando.

No hablamos del valor, estamos hablando del TIPO.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**Conversiones entre tipos compatibles**

Se realiza la conversión cuando los tipos son compatibles (número con número) y el tipo es **mayor** a lo que vamos a almacenar.

Se utiliza la sintaxis: (TIPO DATO A CONVERTIR)valor;

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

**Convertir de String a cualquier primitivo**

Esta conversión nos permitirá almacenar cualquier texto en un tipo primitivo deseado (números, booleans, fechas…)

El valor del texto debe ser compatible, esto no hace milagros, es decir, no podemos convertir el texto “Lunes” a un número.

Se utiliza un método que tienen TODOS los primitivos llamado **Parse**

Dicho método se llama desde la clase que deseamos convertir y aplicando el método **Parse**

**TipoDato.Parse(string)**

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

**Convertir cualquier clase a String**

Todas las clases contienen un método llamado **ToString()** para convertir a su representación de string, es decir, para almacenar su valor en un string.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Vamos a realizar un nuevo formulario para sumar dos números.

En el momento de crear un formulario, iremos dando diferentes nombres con una numeración para saber el orden que hemos ido siguiendo.

**Nota:** Si estáis con mi proyecto y necesitáis “jugar”, llamar al formulario de otra forma, es decir, con vuestra inicial al final, por ejemplo.

En el **explorador de soluciones**, sobre el **proyecto** (negrita), botón derecho y **Agregar Nuevo Formulario**.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Llamaremos a nuestro formulario **Form01SumarNumeros**

Cuando agregamos un nuevo formulario al proyecto, no se incluye por defecto como formulario de inicio, tengo que indicarlo de forma explícita

Para indicar el formulario de inicio, lo haremos en la clase **Program**

**PROGRAM**

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

FORMULARIO

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

**CODIGO FORMULARIO**

**private** **void** **btnSumar\_Click**(**object** sender, EventArgs e)

{

    //CAPTURAMOS LOS DATOS DE CADA CAJA DE TEXTO EN UNA

    //VARIABLE

**int** numero1 **=** **int.Parse**(**this.**txtNumero1**.**Text);

**int** numero2 **=** **int.Parse**(**this.**txtNumero2**.**Text);

    //DECLARAMOS UNA VARIABLE PARA ALMACENAR LA SUMA

**int** suma **=** numero1 **+** numero2;

    //MOSTRAMOS EL RESULTADO EN EL LABEL DEL FORMULARIO

**this.**lblResultado**.**Text **=** suma**.ToString**();

}

**CONDICIONALES**

Un condicional es una instrucción que hacemos en un programa para poder evaluar unas condiciones en nuestro código.

Por ejemplo, ahora mismo, si no escribimos nada en las cajas, nuestro programa podría fallar.

Mediante un condicional podemos hacer que nuestro programa se comporte de forma distinta.

Operadores de comparación

== IGUAL

!= DISTINTO

> MAYOR

< MENOR

>= MAYOR O IGUAL

<= MENOR O IGUAL

Sintaxis:

if (condicion == true)

{

//CONDICION TRUE

}

Podemos realizar más preguntas como, un IF-ELSE:

if (condicion == true){

//CONDICION TRUE

}else{

//CONDICION FALSE

}

Tenemos otro código para hacer más preguntas dentro de un mismo IF:

if (condicionA == true) {

//TRUE CONDICION A

}else if (condicionB == true) {

//TRUE CONDICION B

}else {

//SI NO SE CUMPLEN EL RESTO DE PREGUNTAS

}

**Operadores relacionales**: Estos operadores nos permiten agrupar más preguntas dentro de una sola condición

&& AND: Cada pregunta debe cumplirse

|| OR: Cualquier pregunta entra dentro de la condición IF

! NOT: Negar una condición

Por ejemplo, podemos modificar el código del ejemplo anterior y que muestre un mensaje en el caso de que no tenga contenido ninguna de las cajas.

**MODIFICACION DEL EJEMPLO ANTERIOR**

Texto

Descripción generada automáticamente

También podemos preguntar por las dos cajas a la vez. Si no ha escrito nada en ninguna de las cajas, que muestre un mensaje.

Texto

Descripción generada automáticamente con confianza media

Creamos un nuevo formulario llamado **Form02PositivoNegativoCero**

* El formulario tendrá una caja de texto para pedir un número.
* Tendremos un botón para evaluar dicho número.
* Cuando pulsemos el botón, mostraremos en un LABEL si el número es positivo, negativo o es cero.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

**CODIGO FORMULARIO**

**private** **void** **btnEvaluarNumero\_Click**(**object** sender, EventArgs e)

{

    //CAPTURAMOS EL NUMERO DE LA CAJA DE TEXTO

**int** numero **=** **int.Parse**(**this.**txtNumero**.**Text);

    //DEBEMOS PREGUNTAS POR LAS DIFERENTES OPCIONES QUE

    //TENEMOS EN EL PROGRAMA: POSITIVO, NEGATIVO O CERO

**if** (numero **>** 0)

    {

**this.**lblResultado**.**Text **=** "POSITIVO";

    }

**else** **if** (numero **<** 0) {

**this.**lblResultado**.**Text **=** "NEGATIVO";

    }

**else**

    {

**this.**lblResultado**.**Text **=** "CERO";

    }

}