FUNDAMENTOS VISUAL STUDIO

# Introducción:

Visual Studio es un **entorno de desarrollo integrado** (IDE) creado por Microsoft que proporciona herramientas potentes para desarrollar una amplia variedad de aplicaciones. Es una plataforma muy popular entre los desarrolladores debido a su versatilidad y funcionalidad.

Algunos de los fundamentos de Visual Studio incluyen:

1. **Interfaz de usuario**: Visual Studio cuenta con una interfaz intuitiva que facilita la navegación entre proyectos, archivos y herramientas. Permite a los desarrolladores ver el código, diseñar interfaces gráficas y gestionar recursos en un solo lugar.
2. **Soporte para múltiples lenguajes**: Aunque es más conocido por su uso con C# y .NET, Visual Studio es compatible con varios lenguajes de programación, incluidos Visual Basic, F#, C++, JavaScript, y más. Esto permite a los desarrolladores trabajar en diferentes tipos de proyectos con un solo IDE.
3. **Control de versiones**: Visual Studio integra herramientas para el control de versiones, como Git. Esto permite a los desarrolladores gestionar el historial de cambios en su código, colaborar con otros y revertir a versiones anteriores si es necesario.
4. **Depuración y pruebas**: La plataforma incluye potentes herramientas de depuración que ayudan a identificar y solucionar errores en el código. Los desarrolladores pueden establecer puntos de interrupción, inspeccionar variables y seguir el flujo de ejecución para garantizar que su aplicación funcione correctamente.
5. **Plantillas de proyectos**: Visual Studio ofrece una variedad de plantillas de proyectos que permiten a los desarrolladores comenzar rápidamente con su trabajo. Estas plantillas están diseñadas para diferentes tipos de aplicaciones, como aplicaciones web, de escritorio o móviles.
6. **Extensibilidad**: Los desarrolladores pueden personalizar y extender Visual Studio mediante complementos y extensiones. Esto les permite agregar nuevas funcionalidades y adaptar el IDE a sus necesidades específicas.

En resumen, Visual Studio es una herramienta fundamental para desarrolladores que buscan crear aplicaciones de alta calidad de manera eficiente. Su combinación de una interfaz amigable, soporte para múltiples lenguajes y potentes herramientas de desarrollo lo convierte en una elección popular en la comunidad de programación.

# Temario

1. NET CORE LENGUAJE C#
2. Windows Forms
3. Fundamentos del lenguaje
4. Inyección de dependencias
5. Base de datos
6. Diferencias entre .NET Framework y .NET Core

## Net Core Lenguaje C#

**.NET Core** con el lenguaje C# se refiere al uso de C# (un lenguaje de programación desarrollado por Microsoft) dentro del entorno .NET Core, que es una plataforma de desarrollo también de Microsoft.

* **.NET Core** es una *plataforma multiplataforma* (compatible con Windows, Linux y macOS) que permite desarrollar aplicaciones de distintos tipos, como aplicaciones web, aplicaciones de escritorio, servicios de backend, aplicaciones móviles, entre otras.
* **C#** es el lenguaje de programación principal para trabajar en la plataforma .NET Core. Este lenguaje es versátil y potente, y está orientado a objetos, lo cual permite organizar el código en "clases" y "objetos" para estructurarlo mejor.

En conjunto, .NET Core y C# se utilizan para crear aplicaciones modernas, eficientes y compatibles con distintos sistemas operativos y dispositivos. Gracias a su capacidad multiplataforma, un programa creado en .NET Core y C# puede ejecutarse en Windows, Linux y macOS, y si se trata de una aplicación web, también puede alojarse en servidores de varios tipos, como Apache o Nginx.

## Windows Forms

Este curso se basa en una metodología práctica para enseñar C# y .NET Core mediante el desarrollo de **interfaces visuales**.

Una **interfaz visual** es la parte gráfica de una aplicación, donde el usuario interactúa con botones, menús y formularios. Trabajar con interfaces visuales facilita el aprendizaje, ya que permite ver cómo los conceptos de programación se aplican en una interfaz tangible y funcional.

Para ello, utilizaremos **Windows Forms**, un entorno visual ideal para crear aplicaciones gráficas de escritorio.

## Fundamentos del lenguaje

A través de Windows Forms, aprenderemos los fundamentos de C# como sintaxis, clases, programación orientada a objetos (POO) y colecciones de datos, de forma práctica y visual.

## Inyección de dependencias

Otro tema que veremos es la **inyección de dependencias**. Esta técnica **organiza el código** al permitir que cada clase (es decir, cada parte del programa que maneja una función o dato específico) reciba de manera automática las herramientas o datos que necesita para funcionar. Esto simplifica el mantenimiento de la aplicación y hace que el código sea más flexible. Por ejemplo, si una clase necesita conectarse a una base de datos, la inyección de dependencias proporciona esa **conexión directamente**, sin que tengamos que escribir el mismo código en varios lugares.

## Base de datos

Además, aprenderemos a **acceder a datos** desde nuestra aplicación, es decir, a **conectarnos a bases de datos** para consultar o almacenar información. Empezaremos con **ADO.NET**, una tecnología de Microsoft, y luego pasaremos a **LINQ y Entity Framework**, que son las herramientas recomendadas actualmente para trabajar con bases de datos en aplicaciones .NET.

## Diferencias entre .NET Framework y .NET Core

Este curso se centra en .NET Core, una tecnología que permite crear aplicaciones compatibles con Windows, Linux y macOS. A diferencia de .NET Framework, que solo funciona en Windows, .NET Core permite que nuestras aplicaciones se ejecuten en distintos sistemas operativos y servidores como Apache, IIS o Nginx.

## Tipos de proyectos en Visual Studio

Trabajaremos con dos tipos de proyectos en Visual Studio:

* **Librerías**: Proyectos que contienen funciones y lógica de programación reutilizables, sin interfaz gráfica.
* **Proyectos Visuales**: Aplicaciones con interfaz gráfica (ventanas, botones, formularios), principalmente para Windows, aunque también veremos ejemplos de aplicaciones web y móviles.

# Programas

Se utilizarán los siguientes programas (software):

1. **GitHub**.
2. **Visual Studio Community** (IDE)
3. **SQL Server Express**

## Instalación y preparación de programas:

### GitHub.

GitHub es un repositorio de código.

Un **repositorio de código** es un lugar donde se almacena y organiza el código de un proyecto de desarrollo. Es como una carpeta digital que contiene todos los archivos, el código fuente y la documentación necesarios para que un proyecto funcione. Los repositorios permiten llevar un **historial de cambios**, lo que significa que puedes ver y revertir a versiones anteriores del código si es necesario.

Además, los repositorios suelen estar gestionados en plataformas como GitHub, GitLab o Bitbucket, que usan herramientas de **control de versiones** (como Git). Esto permite que varios desarrolladores trabajen juntos en el mismo proyecto, ya que cada uno puede hacer cambios y subirlos al repositorio, mientras que Git ayuda a fusionar esos cambios y a resolver conflictos si varios modifican el mismo archivo.

En resumen, un repositorio de código es el espacio donde se guarda, organiza y versiona el código de un proyecto, permitiendo colaboración y un control ordenado de los cambios.

Github e s de Microsoft, pero no tiene código solo de Microsoft, contiene de todo, desde código Front, React, Angular o Java que es Back.

#### Cuenta en Github

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

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

#### 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.

Se puede crear un repositorio o clonarlo.

##### Crear

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.

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

##### Clonar

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**

Mi proyecto de Github está en la siguiente dirección

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

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

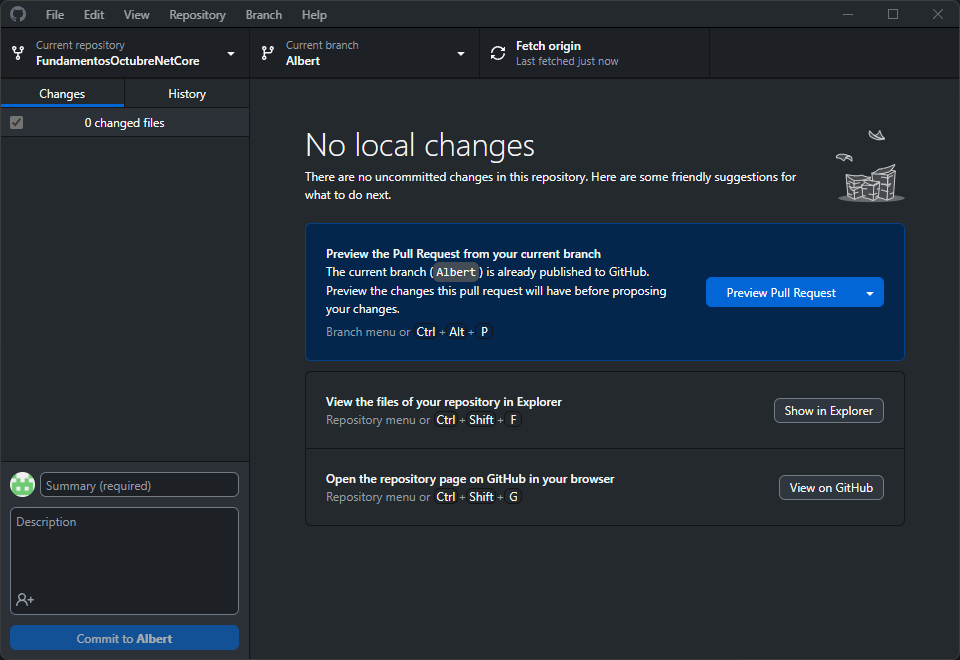
Descripción generada automáticamente

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

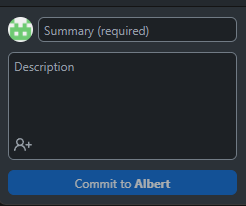
En este caso, suponemos que ya tenemos el proyecto descargado. Dicho proyecto, desde el origen, ha tenido cambios y YO también he realizado cambios.

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

Debemos ir a una ventana llamada **Changes**



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



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



**Nota:** 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.

**Recuerda**

Debemos hacer un Commit para guardar los cambios en local y después, hacer un **Pull/Flech**

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.

### 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

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

1. **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

1. **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

1. **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

1. **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.

### 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)



1. 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.

# 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

1. **char**: Representa un carácter. Se escribe entre comillas simples.
2. **byte**: Almacena números de 0-255.
3. **short**: Almacena números hasta 27.900
4. **int**: Almacena más precisión
5. **long**: El mayor número de precisión de un entero.
6. **float, double, decimal**: Representan números decimales
7. **bool**: true/false
8. **DateTime**: Representa fechas.
9. **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**

1. El formulario tendrá una caja de texto para pedir un número.
2. Tendremos un botón para evaluar dicho número.
3. 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";

    }

}

Realizar una práctica en la que evaluaremos dos números.

Creamos un nuevo formulario llamado **Form03NumerosMayorMenor**

1. Tendremos dos cajas de texto para introducir dos números.
2. Un botón para evaluar la comparación de los números.
3. Al pulsar el botón, debemos mostrar en un LABEL qué número es mayor al otro

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

**CODIGO FORMULARIO**

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

{

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

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

    //DECLARAMOS UNA VARIABLE PARA SABER EL NUMERO MAYOR

**int** mayor **=** 0;

**if** (numero1 **>** numero2)

    {

        mayor **=** numero1;

    }

**else**

    {

        mayor **=** numero2;

    }

**this.**lblResultado**.**Text **=** "El número mayor es " **+** mayor;

}

Vamos a crear otro formulario llamado **Form04MayorTresNumeros**

1. Necesitamos tres cajas de texto para pedir tres números.
2. Un botón para mostrar una serie de datos.
3. Al pulsar el botón, debemos mostrar los siguientes resultados:
   * Mayor
   * Menor
   * Intermedio

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

**CODIGO FORMULARIO**

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

{

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

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

**int** numero3 **=** **int.Parse**(**this.**txtNumero3**.**Text);

**int** mayor **=** 0;

**int** menor **=** 0;

**int** intermedio **=** 0;

    //DEBEMOS COMPARAR EL NUMERO 1 CON EL RESTO DE NUMEROS QUE TENGAMOS

    //VOY A UTILIZAR EN LA COMPARACION TAMBIEN SI SON IGUALES >=

**if** (numero1 **>=** numero2 **&&** numero1 **>=** numero3)

    {

        mayor **=** numero1;

    }**else** **if** (numero2 **>=** numero1 **&&** numero2 **>=** numero3)

    {

        mayor **=** numero2;

    }

**else**

    {

        mayor **=** numero3;

    }

    //COMPARAMOS LOS NUMEROS ENTRE SI CON EL OPERANDO DE <=

**if** (numero1 **<=** numero2 **&&** numero1 **<=** numero3)

    {

        menor **=** numero1;

    }**else** **if** (numero2 **<=** numero1 **&&** numero2 **<=** numero3)

    {

        menor **=** numero2;

    }

**else**

    {

        menor **=** numero3;

    }

    //PARA EL INTERMEDIO, VAMOS A SUMAR TODOS LOS NUMEROS QUE TENEMOS

**int** suma **=** numero1 **+** numero2 **+** numero3;

    //EL INTERMEDIO SERA LA SUMA MENOS EL MAYOR Y EL MENOR

    intermedio **=** suma **-** mayor **-** menor;

**this.**lblMayor**.**Text **=** "Mayor: " **+** mayor;

**this.**lblMenor**.**Text **=** "Menor: " **+** menor;

**this.**lblIntermedio**.**Text **=** "Intermedio " **+** intermedio;

}

**CALCULAR FECHA NACIMIENTO DIA SEMANA**

Creamos un nuevo formulario llamado **Form05DiaNacimiento**

* Pedir una fecha al usuario para calcular el día de la semana de su nacimiento.
* Realizaremos el diseño con tres cajas para que el usuario pueda incluir día, mes y año.
* Mediante un botón, mostraremos el día de la semana en un Label, por ejemplo.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Tenemos que tener la tabla de días de la semana para la correspondencia comenzando en sábado:



Debemos pedir el **día**, el número de **mes** y el **año** que el usuario haya nacido.

A partir de esto datos hay que calcular lo siguiente para averiguar el día de la semana de nacimiento:

Ejemplo: **15/06/1997**

Hay que tener en cuenta el mes para realizar el cálculo, si el mes es Enero, el Mes será 13 y restaremos uno al año. Si el Mes es Febrero, el Mes será 14 y restaremos uno al año. **Solamente para estos dos meses**

Para poder calcular el número final de la semana debemos seguir los siguientes pasos:

1. Multiplicar el Mes más 1 por 3 y dividirlo entre 5

((6 + 1) \* 3) / 5 🡪 4

1. Dividir el año entre 4

1997 / 4 🡪 499

1. Dividir el año entre 100

1997 / 100 🡪 19

1. Dividir el año entre 400

1997 / 400 🡪 4

1. Sumar el día, el doble del mes, el año, el resultado de la operación 1, el resultado de la operación 2, menos el resultado de la operación 3 más la operación 4 más 2.

15 + (6 \* 2) + 1997 + 4 + 499 - 19 + 4 + 2 🡪 2514

1. Dividir el resultado anterior entre 7.

2514 / 7 🡪 359

1. Restar el número del paso 5 con el número del paso 6 por 7.

2514 – (359 \* 7) 🡪 1

1. Miramos la tabla y vemos que el número 1 corresponde a **DOMINGO**

**CODIGO FORMULARIO**

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

{

**int** dia **=** **int.Parse**(**this.**txtDia**.**Text);

**int** mes **=** **int.Parse**(**this.**txtMes**.**Text);

**int** anyo **=** **int.Parse**(**this.**txtAnyo**.**Text);

    //15/06/1997

**if** (mes **==** 1)

    {

        mes **=** 13;

        anyo **=** anyo **-** 1;

    }**else** **if** (mes **==** 2)

    {

        mes **=** 14;

        anyo **=** anyo **-** 1;

    }

    //COMENZAMOS A REALIZAR LAS OPERACIONES MATEMATICAS

    //Multiplicar el Mes más 1 por 3 y dividirlo entre 5

**int** op1 **=** ((mes **+** 1) **\*** 3) **/** 5;

    //Dividir el año entre 4

**int** op2 **=** anyo **/** 4;

    //Dividir el año entre 100

**int** op3 **=** anyo **/** 100;

    //Dividir el año entre 400

**int** op4 **=** anyo **/** 400;

    //Sumar el día, el doble del mes, el año, el resultado de la operación 1

    //, el resultado de la operación 2, menos el resultado de la operación 3

    //más la operación 4 más 2.

**int** op5 **=** dia **+** (mes **\*** 2) **+** anyo **+** op1 **+** op2 **-** op3 **+** op4 **+** 2;

    //Dividir el resultado anterior entre 7.

**int** op6 **=** op5 **/** 7;

    //Restar el número del paso 5 con el número del paso 6 por 7.

**int** resultado **=** op5 **-** (op6 **\*** 7);

**if** (resultado **==** 0)

    {

**this.**lblDiaSemana**.**Text **=** "SABADO";

    }**else** **if** (resultado **==** 1)

    {

**this.**lblDiaSemana**.**Text **=** "DOMINGO";

    }**else** **if** (resultado **==** 2)

    {

**this.**lblDiaSemana**.**Text **=** "LUNES";

    }**else** **if** (resultado **==** 3)

    {

**this.**lblDiaSemana**.**Text **=** "MARTES";

    }**else** **if** (resultado **==** 4)

    {

**this.**lblDiaSemana**.**Text **=** "MIERCOLES";

    }**else** **if** (resultado **==** 5)

    {

**this.**lblDiaSemana**.**Text **=** "JUEVES";

    }**else** **if** (resultado **==** 6)

    {

**this.**lblDiaSemana**.**Text **=** "VIERNES";

    }

**else**

    {

**this.**lblDiaSemana**.**Text **=** "Tenemos un error, Houston";

    }

}

# **SUPER IMPORTANTE EN DESARROLLO**

Control de código o depuración. En cualquier programa, existen múltiples tipos de error.

Hemos visto que podemos controlar dos errores:

* **Errores de compilación**: Este tipo de errores, suceden cuando tenemos algún problema con la sintaxis del programa. Simplemente, al ejecutar, indicamos que NO y solucionamos el problema de sintaxis.
* **Errores en ejecución**: Suceden cuando el usuario realiza acciones que no hemos controlado por código, por ejemplo, si estamos pidiendo números en una caja de texto y el usuario escribe letras o deja la caja vacía. Estos errores se controlan mediante preguntas IF.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

* **Errores lógicos**: Este tipo de errores son los más importantes y los que más difícil solución tienen. El programa funciona, todo está estupendo y no tenemos ningún error, pero no hace lo que deseamos, se comporta de forma distinta.

La única forma que tenemos de controlar estos errores es “visualizar” qué está sucediendo en nuestro código y con nuestras variables.

La forma de solucionar este problema es ir paso a paso dentro de la ejecución de nuestro programa y VER los resultados de las variables o lo que estamos haciendo.

En la línea gris que tenemos a la izquierda del código, podemos poner **Puntos de interrupción** para que el programa se detenga en ese momento exacto.

Si pulsamos, nos pondrá una línea roja indicando la parada. También podemos utilizar la tecla F9.

Texto

Descripción generada automáticamente

Cuando ejecutemos el programa, en algún momento se detendrá en esa línea o en otras que hayamos marcado.

Podemos visualizar nuestras variables en ejecución o ir paso a paso con la tecla F11.

Si ponemos el ratón encima de las variables, veremos su valor una vez que hayan pasado el punto de interrupción.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente



Mientras estamos en ejecución, podemos visualizar el valor de las variables locales del programa en una ventana inferior llamada **Locals**

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Una vez que hemos terminado la ejecución y hemos encontrado el problema, podemos quitar el punto de interrupción y ejecutar el programa sin depurar.

Todo en Visual Studio son clases. Tenemos clases gráficas como, por ejemplo, Button, TextBox o clases no gráficas que nos permiten realizar acciones dependiendo de su tipo. (int, float, string)

Cada clase tendrá una serie de Propiedades o métodos con los que podremos trabajar, todo depende de la clase. Dichos métodos son herramientas para facilitar el trabajo en el momento de realizar tareas.

### Clase DateTime

Es una clase en la que podemos utilizar y almacenar Fechas y horas.

Es una clase wrapper, lo que quiere decir que no es necesario que pongamos new para crear un valor, podemos perfectamente igualar su valor a una fecha.

DateTime fecha = “29/10/2024”;

Utilizando **new**

DateTime fecha = **new** DateTime(“29/10/2024”);

Métodos y propiedades de la clase DateTime.

* AddDays(número): Añade un número de días a una fecha
* AddMonths(número): Añade un número de meses a una fecha
* AddYears(número): Añade un número de años a una fecha
* ToLongDateString(): Devuelve la fecha en formato string y nos la devuelve en su formato largo: **Martes 29 de octubre de 2024**
* ToShortDateString(): Devuelve la fecha en formato string y nos la devuelve en su formato corto: **29/10/2024**
* Year: Devuelve el año de una fecha
* Minutes: Devuelve los minutos de una fecha
* Month: Devuelve los meses de una fecha
* Seconds: Devuelve los segundos de una fecha
* DayOfWeek: Devuelve una enumeración con el día de la semana en inglés.
* DayOfYear: Indica el día del año actual.
* Parse(string): Convierte un string al tipo de dato DateTime

Métodos de tipo **static.** Este concepto no tiene que ver con la clase DateTime.

Un método static es un método que pertenece a la clase y es una herramienta.

Dicha herramienta NO utiliza el valor del objeto de la clase.

Por ejemplo, dentro de DateTime tenemos un método llamado **IsLeapYear(AÑO)** que nos devuelve un true/false si el año es bisiesto o no. Este método es **static**

DateTime fecha = “29/10/2024”;

Escribimos el siguiente código:

fecha.IsLeapYear(2030);

El método static utiliza el valor del parámetro que le enviamos, no utiliza el valor del objeto (2024).

De hecho, dicho método NO se llama desde el objeto (fecha), se llama desde el nombre de la clase (DateTime).

DateTime.IsLeapYear(2030);

Tenemos una propiedad muy útil llamada **Now** que nos indica la fecha y la hora actual para un tipo DateTime.

DateTime fecha = DateTime.**Now**;

Para probar esta teoría, vamos a crear un nuevo formulario llamado **Form06DateTime**

Cuando estamos con clases y queremos hacer algo al inicio de dicha clase, se utilizan los constructores.

Un constructor es el lugar dónde se inicializan las variables o los elementos que deseemos en una clase, por ejemplo, en una clase Formulario.

Un constructor se llama exactamente igual que la clase en la que estemos trabajando.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

En nuestro ejemplo, necesitamos dibujar la fecha actual dentro de la caja de texto Fecha Actual y lo haremos en el constructor (Al inicio de la clase)

**Nota:** Tenemos un método llamado InitializeComponent() que no debemos tocar. Dicho método lo utiliza nuestro proyecto Net para los dibujos que hacemos.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

**CODIGO FORMULARIO**

**public** **partial** **class** Form06DateTime : Form

{

**public** Form06DateTime()

    {

**InitializeComponent**();

        //SIEMPRE ESCRIBIREMOS A PARTIR DE ESTA INSTRUCCION

        //NECESITAMOS RECUPERAR LA FECHA ACTUAL

        DateTime fechaActual **=** DateTime**.**Now;

        //DIBUJAMOS EN LA CAJA LA FECHA ACTUAL

**this.**txtFechaActual**.**Text **=** fechaActual**.ToString**();

    }

**private** **void** **chkFormato\_CheckedChanged**(**object** sender, EventArgs e)

    {

        //NECESITAMOS CAPTURAR LA FECHA QUE TENEMOS EN LA CAJA

        DateTime fecha **=** DateTime**.Parse**(**this.**txtFechaActual**.**Text);

        //EL CONTROL CHECKBOX TIENE UNA PROPIEDAD LLAMADA Checked

        //QUE INDICA SI ESTA CHEQUEADO O NO

**if** (**this.**chkFormato**.**Checked **==** **true**)

        {

            //DIBUJAMOS EN LA CAJA EL FORMATO DE FECHA LARGO

**this.**txtFechaActual**.**Text **=** fecha**.ToLongDateString**();

        }

**else**

        {

            //DIBUJAMOS EN LA CAJA EL FORMATO DE FECHA CORTO

**this.**txtFechaActual**.**Text **=** fecha**.ToShortDateString**();

        }

    }

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

    {

        //CAPTURAMOS LA FECHA DE LA CAJA

        DateTime fecha **=** DateTime**.Parse**(**this.**txtFechaActual**.**Text);

        //AÑADIMOS 5 DIAS A LA FECHA

        //AUNQUE APLIQUEMOS UN METODO A UN OBJETO, DICHO OBJETO NO CAMBIA

        //DE VALOR.  SOLAMENTE SI IGUALAMOS EL OBJETO, CAMBIARA DE VALOR

        //fecha = fecha.AddDays(5);

        //DIBUJAMOS EN LA CAJA INFERIOR EL NUEVO VALOR

        //CAPTURAMOS EL INCREMENTO DE LA CAJA DE TEXTO

**int** incremento **=** **int.Parse**(**this.**txtIncremento**.**Text);

        //PREGUNTAMOS POR LOS RADIO BUTTONS

**if** (**this.**rdbDias**.**Checked **==** **true**)

        {

            fecha **=** fecha**.AddDays**(incremento);

        }**else** **if** (**this.**rdbMeses**.**Checked **==** **true**)

        {

            fecha **=** fecha**.AddMonths**(incremento);

        }

**else**

        {

            fecha **=** fecha**.AddYears**(incremento);

        }

**this.**txtNuevaFecha**.**Text **=** fecha**.ToString**();

**this.**lblDayOfWeek**.**Text **=** fecha**.**DayOfWeek**.ToString**();

**this.**lblDayOfYear**.**Text **=** fecha**.**DayOfYear**.ToString**();

    }

}