

Introducción a la Programación en C#

Introducción.

Programa: Conjunto de instrucciones que entiende un ordenador para realizar una actividad.

Para la resolución de un problema hay que plantear un algoritmo.

Algoritmo: Son los pasos a seguir para resolver un problema.

Ayudas algoritmos:

Pseudocódigo: Escribimos los pasos del algoritmo en borrador en nuestro lenguaje general común.

Diagrama de flujo: es la representación gráfica de un ALGORITMO. Resulta mucho más fácil entender un gráfico.

Tipos y diferencias entre C, C++, C#

- **C** creado en 1972, lenguaje más popular para crear software de sistemas, aunque también se utiliza para crear aplicaciones. Se trata de un lenguaje de *medio nivel*, pero con muchas características de *bajo nivel*. Dispone de las estructuras típicas de los lenguajes de alto nivel, pero a su vez, dispone de construcciones del lenguaje que permiten un control a muy bajo nivel. Los compiladores suelen ofrecer extensiones al lenguaje que posibilitan mezclar código en ensamblador con código C.
- **C++** es un lenguaje de los años 1980. La intención de su creación fue el extender al exitoso lenguaje de programación C con mecanismos que permitan la manipulación de objetos. En ese sentido, desde el punto de vista de los lenguajes orientados a objetos, el C++ es un lenguaje híbrido: (programación estructurada y la programación orientada a objetos). Por esto se suele decir que el C++ es un lenguaje de programación multiparadigma.
- **C#** ("C Sharp") lenguaje de programación orientado a objetos desarrollado por *Microsoft* para su plataforma .NET. Su sintaxis básica deriva de C/C++. El nombre C Sharp fue inspirado por la notación musical, sugiriendo que C# es superior a C/C++.

Código: escritura de las instrucciones del programa en un lenguaje de programación.

- **Función:** El objeto de función es para poder dividir un programa grande en un subconjunto de programas o funciones más pequeñas.
- **Función Main:** Función principal que puede iniciarse con la siguiente estructura: *void main (void)* o *int main (int)*
- **Palabras clave:** Son palabras reservadas por el programa y no podemos emplear como nombres de identificadores.
- **Identificadores:** Nombre de una función, variable o constante. No puede contener espacios en blanco, acentos ni caracteres extraños. Distingue mayúsculas de minúsculas. No puede empezar por un número.
- **Comentarios:** /* varias líneas */ o // hasta final de línea
- **Operador de visibilidad ::** Permite acceder a una variable global cuando está oculta por otra local.

Plataforma .NET: Entorno de desarrollo para aplicaciones de Microsoft que permiten el desarrollo de aplicaciones de escritorio, móviles o web. Surgió como alternativa a al entorno virtual Java.

El **.NET Framework** proporciona un entorno de ejecución de aplicaciones Common Language Runtime o **CLR**. Este entorno permite ejecutar las aplicaciones .NET e interactuar con el sistema operativo, común a todos los lenguajes .NET.

Plantillas de Visual Studio

Proporcionan el código inicial para construir y crear rápidamente aplicaciones.

- *Console Application:* Para desarrollar una aplicación que se ejecute en una interfaz de línea de comandos.
- *Windows Forms Application:* código inicial para desarrollar una aplicación gráfica Windows Form.
- *WPF Application:* código inicial para desarrollar una aplicación Windows rica en interfaz de usuario.
- *Blank App (Universal Windows):* código inicial para desarrollar una aplicación de la Plataforma Universal de Windows.
- *Class Library:* código inicial para desarrollar una biblioteca de clases .dll. userer invocar desde otra aplicación.
- *ASP.NET Web: Application (.NET Framework):* desarrollar aplicaciones ASP.NET como Web Forms, MVC o Web API.

XAML:

Extensible Application Markup Language: utiliza elementos y atributos para definir controles en sintaxis XML compatibles con aplicaciones .NET, plataforma Universal de Windows (UWP) o para desarrollar aplicaciones para iOS y Android con Xamarin.

Instalación y descarga de Visual Studio C# Community para escritorio de Windows.

Descarga desde la web <https://visualstudio.microsoft.com/es/downloads/> El instalador Visual Studio.

Soluciones, Proyectos y Formas

- Una Solución contiene uno o más proyectos.
- Un Proyecto contiene una o más Formas, también llamadas formularios o ventanas.
- Una Forma contiene varios "controles".

Modo consola: Ordenes de entradas y salidas desde consola (i/o)

Mostrar mensajes en pantalla:

- En C#: utilizamos el objeto "Console": **Console.Write("Ingrese Horas trabajadas por el operario:");**
- En C: utilizamos **printf("entre comillas fijo");** sin comillas variable
- En C++: podemos utilizar la función cin de la librería iostream: **cout << "Hola " << endl;**

Entrada de datos por teclado:

- En C#: Debemos definir una variable de tipo string que la llamaremos linea: `string linea;`
Luego cada vez que necesitemos ingresar por teclado un conjunto de caracteres utilizaremos la función **ReadLine** del objeto Console con la siguiente sintaxis: `linea = Console.ReadLine();`
Luego poner el contenido de la variable linea en una variable de tipo int: `horasTrabajadas = int.Parse(linea);`
- En C: Usar: **scanf("%d",&horasTrabajadas);**
- En C++: podemos utilizar la función cin de la librería iostream: **cin>>opcion;**

Creación de un proyecto en C# (C sharp) desde consola

Pedir horas y coste/hora y mostrar el sueldo

Pasos para la creación de un proyecto en C# en Microsoft Visual Studio Express:

1. Entramos en "Microsoft Visual C# 2013 Express".
2. Para la creación del proyecto. Escogemos desde el menú la opción "Archivo" -> "**Nuevo proyecto...**"
Aparece un diálogo donde debemos indicar el nombre del proyecto y seleccionar el tipo de proyecto (elegiremos "**Aplicación de consola**") y pondremos como nombre al proyecto "CalculoSuelto".
Podemos ver que el entorno nos genera automáticamente el esqueleto del programa.
Para probar el funcionamiento del programa debemos presionar el ícono con un triángulo verde

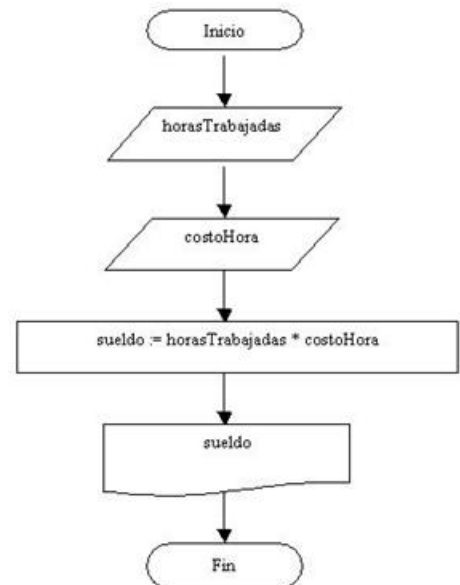
Primero vamos a definir tres variables: (horasTrabajadas, costoHora, sueldo). La cantidad de horas normalmente será un valor entero (integer), pero el costo de la hora es muy común que sea un valor decimal (coma flotante o float) y como el sueldo resulta de multiplicar las horas trabajadas por el costo por hora el mismo deberá ser decimal.

La definición de las variables la hacemos en la Main:

```
int horasTrabajadas; float costoHora, sueldo;
```

las palabras clave en minúsculas y el nombre de la variable, por ejemplo: horasTrabajadas (se propone que el nombre de la variable comience con minúscula y en caso de estar constituida por dos palabras o más palabras poner en mayúsculas el primer carácter (un nombre de variable no puede tener espacios en blanco, empezar con un número, ni tampoco utilizar caracteres especiales)

Utilizar nombres de variables "amigables" que indiquen lo que representan.



Programación en C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;


namespace CalculoSuelto
{
    class Program
    {
        static void Main(string[] args)
        {
            int horasTrabajadas;
            float costoHora;
            float sueldo;
            string linea;
            Console.Write("Horas trabajadas:");
            linea = Console.ReadLine();
            horasTrabajadas = int.Parse(linea);
            Console.Write("Cote por hora:");
            linea = Console.ReadLine();
            costoHora = float.Parse(linea);
            sueldo = horasTrabajadas * costoHora;
            Console.Write("El sueldo total del operario:");
            Console.Write(sueldo);
            Console.ReadKey();
        }
    }
}
```

Programación en C o C++

```
#include <stdio.h> //ok edu
main ()
{
    int horasTrabajadas; //se declaran las variables
    float costoHora, sueldo; //2 variables en misma linea
    costoHora, sueldo=0; //se inicializan las variables
    printf("Horas trabajadas por el operario:");
    scanf("%d",&horasTrabajadas);
    printf("Coste por hora:");
    scanf("%f",&costoHora);
    sueldo = horasTrabajadas * costoHora;
    printf("El sueldo total del operario es: %f",
    sueldo);
    printf("\n");
    printf ("Pulsa RETURN para terminar.");
    scanf("%d");
}

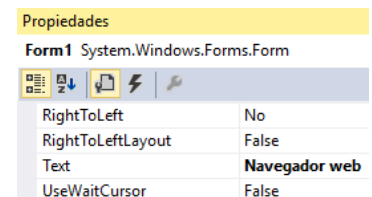
/*%c: formato caracter %d: formato entero %f:
formato decimal flotante. (introducir los decimales
con el punto decimal no la coma) */
```

Crear una aplicación de ventana/formulario Windows Forms en C#

- ▶ En el menú Archivo, haga clic en **Nuevo proyecto**.
- ▶ Aparecerá el cuadro de diálogo *Nuevo proyecto* con diferentes tipos de aplicaciones que puede crear.
- ▶ Seleccione **Aplicación de Windows Forms** como tipo de proyecto 
- ▶ Active la casilla: **Crear directorio para la solución** y cambie el nombre de la aplicación a *Navegador*. Aceptar.
- ▶ Se mostrará en la vista Diseñador un formulario o ventana de Windows vacía, titulada *Form1*.
- ▶ En la **vista Diseño**, puede arrastrar diversos controles desde el *Cuadro de herramientas* hasta el formulario. Estos controles no están realmente "activos", Visual C# en segundo plano, crea el código para que el control real ocupe la posición correcta cuando se ejecute el programa. Este código fuente de diseño se encuentra en el archivo *Form1.designer.cs*.

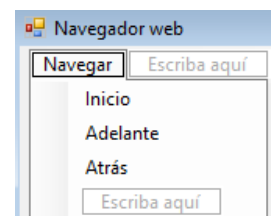
Cambiar el título del formulario Windows:

- ▶ Haga clic en el formulario para seleccionarlo.
- ▶ Active la ventana *Propiedades* desde el menú: *Ver – Ventana de propiedades*
- ▶ Cambie la propiedad *Text*, por: *Navegador web*.



Poner un menú:

- ▶ Active el *Cuadro de herramientas*. Desplácese hacia abajo por la lista de controles y expanda *Menús y barras de herramientas* hasta que vea **MenuStrip**. Arrastre este control a cualquier lugar del formulario Windows. Este control crea un menú predeterminado en la parte superior del formulario.
- ▶ En el cuadro que dice *Escriba aquí*, escriba **Navegar**. Cuando presione ENTRAR, aparecerán nuevos cuadros vacíos para crear más elementos de menú. En el cuadro inferior, escriba **Inicio**. Presione ENTRAR y aparecerán más cuadros. Escriba **Adelante**. Presione ENTRAR y escriba **Atrás**



Agregue un botón.

- ▶ En el *Cuadro de herramientas*, en la categoría *Controles comunes*, arrastre un control **Button** hasta aproximadamente la mitad del formulario, justo debajo de la barra de menús. En sus *Propiedades*, cambie la propiedad *Text* a *Ir* en lugar de *button1*, y cambie el nombre del diseño, que se muestra como (Nombre), de *button1* a **BotonIr**.

Agregue un control ComboBox.

- ▶ En el *Cuadro de herramientas*, en la categoría *Controles comunes*, arrastre un control **ComboBox** y colóquelo a la *izquierda* del botón. Arrastre los bordes y las esquinas para cambiar el tamaño hasta que quede alineado con el botón. El control **ComboBox** va a contener una lista de los sitios web favoritos. Para crear la lista de sitios, seleccione el control **ComboBox** y vea sus *propiedades*. Seleccione la propiedad **Items** Agregue tantas direcciones URL del sitio Web como desee, como por ejemplo:

http://www.ofimega.es

http://www.google.com

presionando RETORNO ↵ después de cada una.

Agregue el control WebBrowser.

- ▶ En el *Cuadro de herramientas*, en la categoría *Controles comunes*, desplácese hacia abajo hasta llegar al control **WebBrowser**. Arrastre el control hasta el formulario Windows Forms. Cambie el tamaño del control **WebBrowser** para ajustarlo al formulario Windows sin ocultar los controles **ComboBox** y **Button**. Al establecer la configuración de *Anchor* en *Superior, Inferior, Izquierda, Derecha*, el control **WebBrowser** cambiará su tamaño correctamente cuando se cambie el tamaño de la ventana de la aplicación. El control **WebBrowser** realiza la representación de páginas Web.

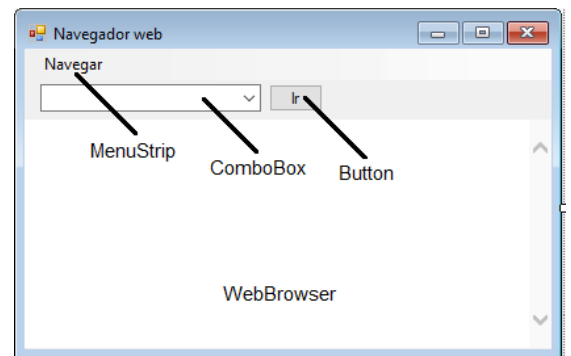
Agregue un controlador de eventos para el control Button.

- ▶ Un controlador de eventos es un método que se ejecuta cuando el usuario interactúa con el control.
- ▶ Haga doble clic en el botón y verá aparecer el *Editor de código* para el proyecto. También verá que se ha creado el controlador para el evento *Click*, Agregue código al método del controlador de eventos de modo similar al siguiente código:

```
private void goButton_Click(object sender, System.EventArgs e)
{
    webBrowser1.Navigate(new Uri(comboBox1.SelectedItem.ToString()));
}
```

Agregue controladores de eventos para las opciones de MenuStrip.

- ▶ Vuelva a la ventana Diseñador y haga doble clic en los subelementos del menú de uno en uno. Visual C# creará métodos de control de eventos para cada uno. Edite estos métodos, de modo que se asemejen al código siguiente.



```
private void homeToolStripMenuItem_Click(object sender, System.EventArgs e)
{
    webBrowser1.GoHome();
}
private void goForwardToolStripMenuItem_Click(object sender, System.EventArgs e)
{
    webBrowser1.GoForward();
}
private void goBackToolStripMenuItem_Click(object sender, System.EventArgs e)
{
    webBrowser1.GoBack();
}
```

Agregue código de inicialización a Form1 en el método Form1_Load.

- Haga clic en la ficha Form1.cs [Diseño] en la parte superior del editor de código para regresar al formulario Windows. Seleccione el formulario y, en la ventana *Propiedades*, haga clic en el botón *Eventos* (el que tiene un icono de rayo) y, a continuación, haga doble clic en Cargar. Esto agregará un método de control de eventos y colocará el cursor en el método en la vista Código.
- En la vista Código, agregue:

```
private void Form1_Load(object sender, EventArgs e)
{
    comboBox1.SelectedIndex = 0;
    webBrowser1.GoHome();
}
```

Genere y ejecute el programa.

- Presione F5 para generar y ejecutar el explorador web. Se mostrará en pantalla el formulario *Windows Forms* creado y, a continuación, aparecerá la página principal predeterminada del equipo. Puede utilizar el control ComboBox para seleccionar un sitio web, y hacer clic en Ir para navegar al mismo.

Para finalizar pulse en *Archivo – Guardar todo*.

Ampliación:

Si también quieres que acceda a una dirección local puedes añadir el código:

```
System.Diagnostics.Process.Start(comboBox1.SelectedItem.ToString());
```

Un poco más de teoría:

Objetos y clases en lenguajes POO (Programación Orientada a Objetos)

Un objeto es una “cosa” en nuestro programa, representada por un nombre y que pertenece a un tipo o clase de objetos. Nuestro televisor es un objeto particular que pertenece a la clase general del tipo “televisores”.

Los lenguajes POO permiten abstracción, herencia, encapsulación y polimorfismo.

Tipo de datos en lenguajes POO o de clases.

Los datos concretos o **primitivos** son los que viene “de serie” con el lenguaje.

En los datos **abstractos** tenemos que declarar el **tipo** previamente como una *clase* o *estructura* de conjunto de datos.

Tipos de datos primitivos y operadores.

Tipos de Datos en C#		Operadores	
Tipo	Descripción		
int / long	Números enteros.	Aritméticos	+, -, *, /, %
float / double	Números de coma flotante	Incremento, decremento	++, --
decimal	Valores de moneda.	Concatenación	+
char	Un simple carácter Unicode.	Operaciones lógicas	&, , ^, !, ~, &&,
bool	Valor booleano.	Indizado	[]
dateTime	Momentos en el tiempo	Asignación	=, +=, -=, *=, /=, %=, ^=, <<=
string	Texto o cadena de caracteres	Tipo de datos	sizeof, typeof
		Apuntadores	*, -, [, &

TIPOS DE ARCHIVOS EN C# CON WINDOWS FORMS:

Un archivo de solución (*.sln) puede contener a uno o varios proyectos. Un proyecto puede contener varias clases o tipos. Un proyecto de Windows Forms contiene la clase o tipo Form1. Al guardar una solución, se guardan, a su vez, los siguientes archivos, por ejemplo:

Form1.cs: Código fuente en C#; Form1.resx: archivo de recursos en XML;

Ejemplo.csproj: Archivo del proyecto; Ejemplo.projdata: archivo oculto interno de VStudio

ClassExample.sln: Archivo que contiene los elementos de la solución.

ClassExample.csproj.user: guarda las opciones personalizadas del proyecto.

EJERCICIO 1

- Realizar una interfaz gráfica que permita al usuario introducir dos números. El programa calculará y desplegará la suma de ambos.
- Se requieren 3 etiquetas, 3 cuadros de texto y dos botones. Cambiar sus propiedades como indica la 2ª figura



► **Añadir el código:**

En el botón sumar button1_Click():

```
int n1, n2, suma;
n1 = int.Parse(txtNumero1.Text);
n2 = int.Parse(txtNumero2.Text);
suma = n1 + n2;
txtResultado.Text = suma.ToString();
```

En el botón salir: Application.Exit();

Ejercicio propuesto: Agregar un botón más al formulario para “Limpiar” el contenido de los cuadros de texto

EJERCICIO 2:

Realizar una pantalla que pida “Login” y “Password” a un usuario. Mostrar un mensaje de “Bienvenida” si los datos son correctos, o un mensaje de “Rechazo” si no lo son.

Datos correctos: – Login: “ofimega” – Password: “danone”

Formulario:

Código:



```
private void button1_Click(object sender, EventArgs e)
{
    string Login, Password; //variables de texto
    Login = txtLogin.Text.Trim(); //quita espacios
    Password = txtPassword.Text.TrimEnd(); //quita espacios
    if (Login=="Ofimega" && Password=="danone")
    {MessageBox.Show("Bienvenido al sistema");}
    else
    { MessageBox.Show("Acceso denegado"); }
}
```

- Ocultar los caracteres tecleados en la *password* en la propiedad *PasswordChar*
- El método *TrimEnd()* elimina los espacios en blanco hasta el final
- MessageBox.Show* (“Mensaje”) muestra una ventana con un mensaje para el usuario

Comparación de cadenas:

- mediante el comparador ==
- mediante *Equals*
- mediante *CompareTo*

EJERCICIO. AREA DEL TRIANGULO:

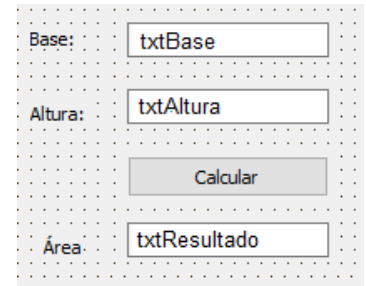
- El programa calculará el área del triángulo a partir de la base y la altura con la fórmula: $\text{Area} = \text{Base} * \text{Altura} / 2$
- Se requieren 3 etiquetas, 3 cuadros de texto y un botón.
- Distribuir y cambiar sus propiedades como indica la figura

Código en C++ (Builder) :

```
void __fastcall TForm5::Button1Click(TObject *Sender) {  
    txtResultado->Text=FormatFloat("###.##",txtBase -> Text.ToDouble()*  
    txtAltura ->Text.ToDouble()/2);  
}
```

Código en C# :

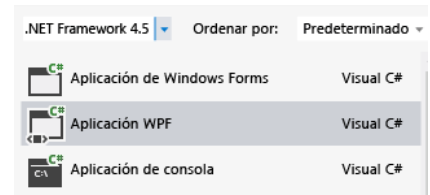
```
private void button1_Click(object sender, EventArgs e) {  
    float b, a;  
    b=float.Parse(txtBase.Text);  
    a=float.Parse(txtAltura.Text);  
    txtResultado.Text = (a * b / 2).ToString();  
}
```



Crear una aplicación en modo WPF:

Una aplicación en modo **WPF** (Windows Presentation Foundation) permite utilizar el diseño del formulario en formato hipertexto extendido XAML, sus controles están basados en formato vectorial, código de programación separado del diseño gráfico, permite la posibilidad de trabajo en conjunto para diseñadores y programadores

- Elije del menú Crear un Nuevo proyecto del tipo: Aplicación WPF →
- Dale el nombre: **HolaMundoWPF** y la ubicación que quieras para tu aplicación.
- Aconsejable tener marcada la casilla de verificación: *Crear directorio para la solución.*

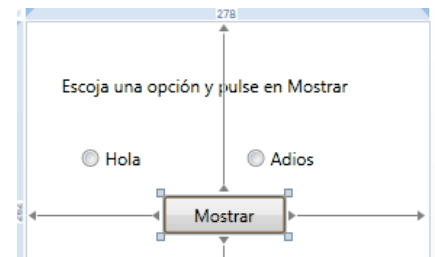


Cambiar el nombre de MainWindow.xaml:

- En el Explorador de soluciones, selecciona MainWindow.xaml.
- En la ventana Propiedades, (Si no se ve: Ver -> Ventana de propiedades). Cambia la propiedad Nombre de archivo (File Name) : a Hola.xaml. Este archivo de código está anidado bajo el nodo del archivo .xaml para mostrar su relación.

Agregar controles:

- En el Cuadro de herramientas (Si no se ve: Ver -> Ventana de propiedades), busca el control: TextBlock. (Bloque de texto) y arrástralo a la ventana.
- Para cambiar el texto del recuadro en las propiedades: Con el textBlock seleccionado, busca la propiedad Text y añade el texto: "Escoja una opción y pulse en Mostrar"
- Para cambiar el texto del recuadro en modo Xaml, busca la línea Xaml:
<TextBlock Margin="30,58,21,0" Name="textBlock1" Text="Escoja una opción y pulse en Mostrar" Height="42" VerticalAlignment="Top" />
- Elijiendo el elemento RadioButton y arrástralo a la dos veces para tener dos controles RadioButton.
- En la superficie de diseño, selecciona RadioButton1 y en sus propiedades añade a la propiedad Content el texto: Hola.
- En la superficie de diseño, selecciona RadioButton2 y en sus propiedades añade a la propiedad Content el texto: Adiós.
- En el Cuadro de herramientas, busca el control Botón (Button) y, después, agrégalo a la superficie de diseño.
- Cambia la propiedad del botón Content por: Mostrar.



Agregar código al botón Mostrar

En la versión 2018 pulsa doble clic sobre el botón para abrir el evento: button1_Click

o en la versión 2013 pulsa en el rayo junto a las propiedades.

Depurar y probar:

Para buscar y corregir errores, inicia el depurador seleccionando *Depurar -> Iniciar depuración.*

Aparece un mensaje de error: No se encuentra el recurso 'mainwindow.xaml'.

Falta especificar Hola.xaml como el URI de inicio: abre el archivo App.xaml y cambia StartupUri="MainWindow.xaml" a StartupUri="Hola.xaml" y después guarda los cambios con Ctrl-s.

Para más información véase el sitio de Microsoft: <https://msdn.microsoft.com/es-es/library/jj153219.aspx>

```
private void button1_Click(object sender,  
    RoutedEventArgs e)  
{  
    if (radioButton1.IsChecked==true)  
    {  
        MessageBox.Show("Hola");  
    }  
    else  
    {  
        radioButton2.IsChecked = true;  
        MessageBox.Show("Adiós");  
    }  
}
```

Mostrar mensajes y ventanas de diálogo

La clase: **MessageBox** muestra una ventana modal de mensajes que puede contener texto, botones y símbolos que informan y dan instrucciones al usuario. (*En Visual Basic: MsgBox ()* , *en C++ Builder: ShowMessage()*)

Ejercicio 1:

1. Crea un nuevo proyecto. (Escoge del menú: *Archivo ▶ Nuevo proyecto* de Windows forms)
2. Añade un botón de comando. Ve cambiando el código de evento del botón1 para los mensajes.
3. Comprueba su funcionamiento. Al finalizar, cierra y guarda el proyecto con el nombre: Mensaje1

➤ *Mensaje de información:*

```
{
  MessageBox.Show("Mensaje informativo", "Atención", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
}
```

➤ *Mensaje de error:*

```
{
  MessageBox.Show("Ha habido un error", "Mensaje de error", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

➤ *Mensajes de decisión*

```
{
  MessageBox.Show("¿Desea continuar?", "Pregunta", MessageBoxButtons.OKCancel);
}
```

Ejercicio con evaluación de la respuesta:

```
{
  var result = MessageBox.Show("¿Mensaje?", "Pregunta", MessageBoxButtons.YesNo, MessageBoxIcon.Question);
  if (result == DialogResult.Yes) MessageBox.Show("Has contestado si");
  else MessageBox.Show("Has contestado no");
}
```

Ejercicio 2:


Añade al formulario anterior un cuadro de texto como en la imagen.

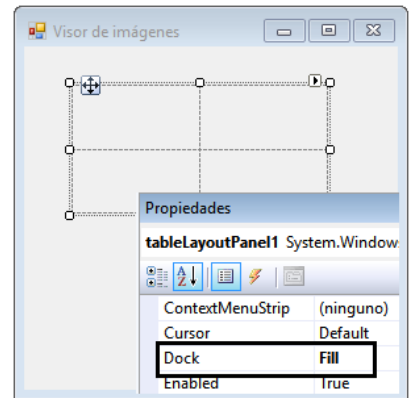
Si al pulsar en el botón, no has escrito nada en el cuadro de texto, se mostrará un mensaje de decisión.

```
private void button1_Click(object sender, EventArgs e)
{
    {
        if (textBox1.Text.Length == 0) // Comprueba si la longitud del texto es 0
        {
            // Variables del tipo primitivas (de texto):
            string mensaje = "No has escrito el nombre. ¿Cancelar operación?";
            string titulo = "Error de entrada";
            // Variables del tipo abstractas:
            MessageBoxButtons botones = MessageBoxButtons.YesNo;
            DialogResult resultado;
            // Muestra el mensaje -----
            resultado = MessageBox.Show(mensaje, titulo, botones);
            if (resultado == System.Windows.Forms.DialogResult.Yes)
            {
                this.Close(); // Cierra esta ventana
            }
        }
    }
}
```

Comprueba su funcionamiento. Al finalizar, cierra y guarda el proyecto con el nombre: Mensaje2

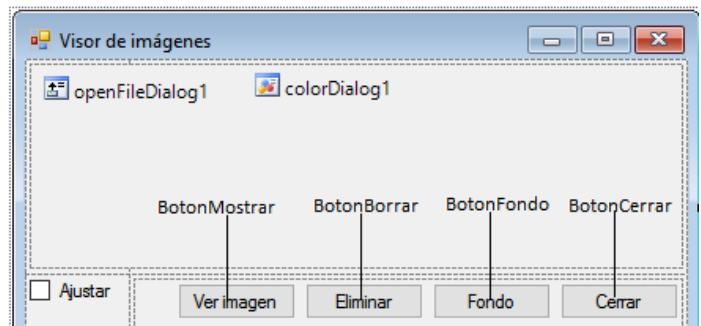
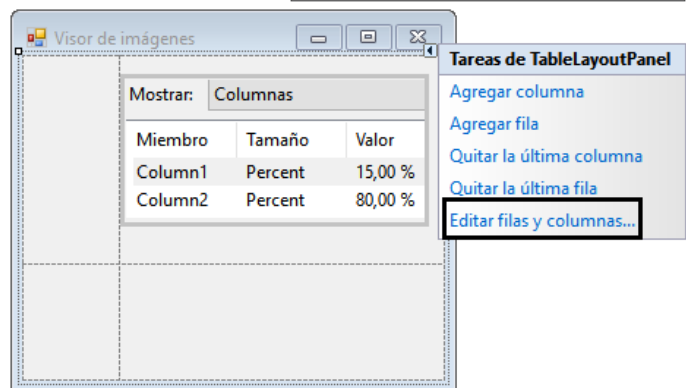
Ejercicio C# Windows Forms: Visor de imágenes.

- Crea un *Nuevo Proyecto*– Aplicación de Windows Forms.
- Pon el nombre del nuevo formulario: **Visor**, y pulsa Aceptar.
- Arrastra la esquina de la ventana-formulario para ampliar su tamaño.
- En el panel *Propiedades*, escribe en **Texto: Visor de imágenes**. Si no ves la ventana de propiedades pulsa en .
- Arrastra de la **Caja de herramientas** un control: **TableLayoutPanel** sobre el formulario.
- Selecciónalo y cambia la *propiedad: Dock* a **Fill** para ajustarlo a todo el formulario.
- TableLayoutPanel tiene dos filas y dos columnas de igual tamaño. Para cambiar el tamaño de la fila superior y la columna derecha, pulsa en un pequeño triángulo negro de la esquina superior derecha. Selecciona **Editar filas y columnas**. Pon **15** en el porcentaje de la columna 1.



Seleccione Filas (rows) y pon a **90 %** la fila 1 y a **10 %** la fila 2.

- Añade, del cuadro de herramientas, un **PictureBox** (cuadro de imagen) al formulario.
- Cambia su propiedad **Dock** por **Fill** (llenar).
- Establecer su **ColumnSpan** propiedad a **2**. Además, cuando el PictureBox está vacío, quiere mostrar un marco vacío. Establece su propiedad **BorderStyle** a **Fixed3D**.
- Añade un control **CheckBox** al formulario.
- Agregar un control **FlowLayoutPanel** a la última celda (abajo a la derecha) y cambia su propiedad **Dock** por **Fill** (llenar).
- De la caja de herramientas, añade **tres** botones **Button** al FlowLayoutPanel.
- En el primer botón y establece su propiedad texto a: **Ver imagen**. A continuación, establece las propiedades de texto de los tres botones: **Eliminar**, **Color de fondo** y **Cerrar**.
- Cambiar en el FlowLayoutPanel su propiedad **FlowDirection** en **RightToLeft**. Los botones deben alinearse a la derecha de la celda, e invertir su orden.
- Selecciona todos los botones a la vez, (tecla CTRL) y cambia la propiedad **AutoSize** = True.
- Cambiar el nombre de los botones: **BotonBorrar**, **BotonCerrar**, **BotonFondo** y **BotonMostrar**.
- Añade un control **ColorDialog** al formulario desde el cuadro de herramientas (Tool box)
- Añade también un control **OpenFileDialog**. Cambia su propiedad **Filter**: **JPEG (*.jpg)|*.jpg|PNG (*.png)|*.png|BMP (*.bmp)|*.bmp|Todos (*.*)|*.*** y su propiedad **Title** : “Escoja una imagen”.



Código en C#:

```
private void BotonMostrar_Click(object sender,
EventArgs e)
{
    if (openFileDialog1.ShowDialog() ==
DialogResult.OK)
    {
        pictureBox1.Load(openFileDialog1.FileName);
    }
}

private void BotonBorrar_Click(object sender,
EventArgs e)
{
    pictureBox1.Image = null; // Borra la imagen.
}

private void BotonFondo_Click(object sender,
EventArgs e)
{
    if (colorDialog1.ShowDialog() ==
DialogResult.OK) // abre la caja de color
    pictureBox1.BackColor = colorDialog1.Color;
}
}
```

```
private void BotonCerrar_Click(object sender,
EventArgs e)
{
    this.Close(); // cierra la ventana
}

private void checkBox1_CheckedChanged(object
sender, EventArgs e)
{
    if (checkBox1.Checked) // cambia el ajuste
    pictureBox1.SizeMode =
PictureBoxSizeMode.StretchImage;
    else
    pictureBox1.SizeMode =
PictureBoxSizeMode.Normal;
}
}
```


Ejercicio C# Windows Forms: Listas y login

En este ejercicio utilizaremos variables de cadena de texto (strings) para traspasar texto de unos objetos a otros.

- Crea un nuevo proyecto:
 - Archivo – Nuevo proyecto (File – New Project)
 - Aplicación de Windows Forms: **Listas**
- Añade al formulario los objetos de la figura. →

Ingredientes:

- 7 Labels
- 2 Buttons
- 2 TextBox
- 1 ComboBox
- Items: Administrador / Usuario
- Text: Usuario
- 2 ListBox

- Añade el código a los eventos ⚡:
- Añadir el código al evento Click del botonCerrar:

```
private void botonCerrar_Click(object sender, EventArgs e)
    {this.Close();} //-> añadir este código
(en realidad este código es delegado al método:
this.botonCerrar.Click += new System.EventHandler(this.botonCerrar_Click);)
```

Teoría: El identificador **this**: En C#, se puede usar la palabra clave this, para nombrar al objeto que se está ejecutando ese código.

- Añadir el código al evento Click del botonAñadir:

```
private void botonAñadir_Click(object sender, EventArgs e) {
    string tipo = comboBox1.Text;
    string nombre = textBox1.Text;
    string apellidos = textBox2.Text;
    string nombreCompleto = nombre + " " + apellidos;
    int total1, total2;
    if (nombreCompleto != " ") { // probar con: if (comboBox1.SelectedIndex==0)
        if (tipo == "Administrador")
            listBox1.Items.Add(nombreCompleto);
        else listBox2.Items.Add(nombreCompleto);
        total1 = comboBox1.Items.Count;
        labelTotal1.Text = "Total: " + total1.ToString();
    }
    else MessageBox.Show("Datos incorrectos");
} }
```

Ampliación:

- Añadir un botón para eliminar de la lista y otro para vaciar la lista:

```
private void buttonVaciar1_Click(object sender, EventArgs e)
    {listBox1.Items.Clear(); } // borra todos los elementos
private void buttonEliminar1_Click(object sender, EventArgs e)
    {listBox1.Items.Remove(listBox1.SelectedItem); } //borra el seleccionado de la lista
```

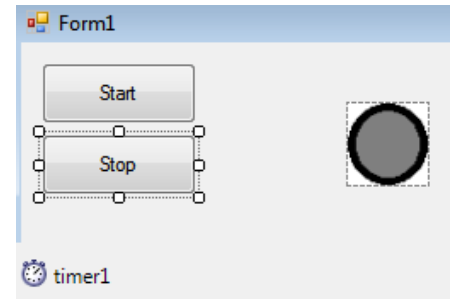
- Añadir un botón para llamar a iniciar sesión: (el formulario del ejercicio 2):

```
this.Hide(); //oculta este objeto (la venta actual)
Form2 frm = new Form2();
frm.ShowDialog(); //abre la otra ventana en modo modal (diálogo)
if (frm.DialogResult == DialogResult.OK) MessageBox.Show("Correcto");//si se cerró con OK...
this.Show(); //vuelve a mostrar la ventana
```

Ejercicio C# Windows Forms: Pong

En este ejercicio utilizaremos las propiedades *Left* y *Top* y el objeto *Timer* para desplazar una imagen por la ventana.

- Crea un nuevo proyecto:
Archivo – Nuevo proyecto (File – New Project)
Aplicación de Windows Forms: **Pong**
- Añade al formulario los objetos de la figura. →



Ingredientes:

- 2 Buttons: Text: Start y Stop
- 1 Timer: Interval: 20.
Image: Importar pelota.gif
- 1 Picture box. Name: bola

▶ Añade el código:

Primero crearemos dos variables públicas antes de public Form():

```
int vel=5; //variable numérica entera para la velocidad
int direc = 1; // variable numérica entera para la dirección
public Form1()
```

- ▶ Añadir el código al evento Click del botón Start: `timer1.Enabled = true;`
- ▶ Añadir el código al evento Click del botón Stop: `timer1.Enabled = false;`
- ▶ Añadir el código al evento Tic del Timer:

private void timer1_Tick(object sender, EventArgs e)

```
{
Try //bloque a evaluar en caso de error...
{
if (direc == 1) //abajo der
{
bola.Left = bola.Left + vel;
bola.Top = bola.Top + vel;
if ((bola.Top+bola.Height) >= this.Height) direc = 2;
if ((bola.Left+ bola.Width) >= this.Width) direc = 3;
}
if (direc == 2) //arriba der
{
bola.Left = bola.Left + vel;
bola.Top = bola.Top - vel;
if ((bola.Top) <=0) direc = 1;
if ((bola.Left + bola.Width) >= this.Width)
direc = 4;
}
}
if (direc == 3) //abajo iz
{
bola.Left = bola.Left - vel;
bola.Top = bola.Top + vel;
if ((bola.Top) >= this.Height) direc = 4;
if ((bola.Left)<=0 ) direc = 1;
}
if (direc == 4) //arriba iz
{
bola.Left = bola.Left - vel;
bola.Top = bola.Top - vel;
if ((bola.Top) <=0) direc = 3;
if (bola.Left <= 0) direc = 2;
}
}
Catch // bloque en caso de error
{
MessageBox.Show("error de dirección");
Close();
}
}
```

Ejercicio propuesto:

Al hacer clic sobre la pelota, aumentará la velocidad y la puntuación.

- Incrementando la variable `vel`: `vel++;`
- Reduciendo el intervalo del timer: `timer1.interval--;`

Teoría: Control de errores o excepciones **TRAY - CATCH - FINALLY**

Agrupar instrucciones en un bloque **try** nos permite saltar el bloque en caso de error y detectar el tipo de excepción: `outOfMemory`, `stackOverflow`, `indexOutOfRangeException`, `divideByZero`, etc...

try

```
{ instrucciones }
```

catch

```
{ instrucciones de excepción }
```

finally

```
{ instrucciones de liberación }
```

Ejercicio C# Windows Forms: Juego de parejas. (Extracto tutorial MSDN)

Se trata de un Juego de buscar parejas entre iconos ocultos.

- ▶ Crear el proyecto: Archivo – Nuevo proyecto (File – New Project) - Aplicación de Windows Forms: **Parejas**
- ▶ Cambia la propiedad Tamaño del Form (Size): en 550; 550

- ▶ Agrega un control *TableLayoutPanel*

Propiedades: BackColor: tipo web: CornflowerBlue - Dock: Fill - CellBorderStyle: Inset (Insertado)

Pulsa en el triángulo del menú contextual: Editar Filas y Columnas: 4 filas x 4 columnas. 25%

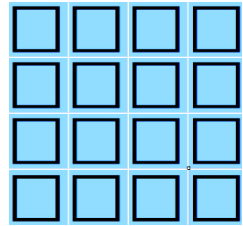
- ▶ Con el control *TableLayoutPanel* seleccionado, agrega un control *Label* a la celda superior izquierda del *TableLayoutPanel*.

Propiedades para el *label*: BackColor: tipo web: CornflowerBlue - AutoSize: False – Dock: Fill – TextAlign: MiddleCenter – Font: Webdings, Estilo de fuente en Negrita y Tamaño en 70 – Text: c.


- ▶ Copia el *Label* y pega en cada cuadro.

- ▶ En: Ver- Código, agregar el código:

```
public partial class Form1 : Form
{
    Random random = new Random(); // aleatorio
    List<string> icons = new List<string>() //nuevo objeto del tipo lista de strings para los iconos
    {
        "l", "l", "N", "N", ",", ",", "k", "k",
        "b", "b", "v", "v", "w", "w", "z", "z"
    };
    Label firstClicked = null;
    Label secondClicked = null;
    private void AssignIconsToSquares() //asignaremos a cada label un valor aleatorio
    {
        foreach (Control control in tableLayoutPanel1.Controls) //repite para cada control
        {
            Label iconLabel = control as Label;
            if (iconLabel != null)
            {
                int randomNumber = random.Next(icons.Count);
                iconLabel.Text = icons[randomNumber];
                icons.RemoveAt(randomNumber);
            }
        }
    }
    public Form1()
    {
        InitializeComponent();
        AssignIconsToSquares(); //--> Llamamos al método
    }
}
```



Una vez comprobado, añade el código: `iconLabel.ForeColor = iconLabel.BackColor;` para ocultar iconos

- ▶ Añade el código al evento  Click del Label:

```
Label clickedLabel = sender as Label;
if (clickedLabel != null) {
    if (clickedLabel.ForeColor == Color.Black)
        return;
    if (firstClicked == null) {
        firstClicked = clickedLabel;
        firstClicked.ForeColor = Color.Black;
        return;
    }
}
```

- ▶ Agrega un control Timer. Interval: 750

```
private void timer1_Tick(object sender, EventArgs e)
{
    timer1.Stop();
    firstClicked.ForeColor = firstClicked.BackColor;
    secondClicked.ForeColor = secondClicked.BackColor;
    firstClicked = null;
    secondClicked = null;
}
```

Namespace: Conjunto de objetos y clases con un nombre, para poder ser compartidos con otros programadores. .NET ya viene con varios nombres de espacios predefinidos como *System* que contienen clases que implementan funciones básicas y de conversión.

Código completo:

```
namespace WindowsFormsApplication{
    public partial class Form1 : Form{ // crea la clase form que representa la ventana de la aplicación
        Random random = new Random(); // creamos la variable random del tipo random (aleatorio)
        List<string> icons = new List <string>() //array del tipo lista de texto que contendrá letras para los iconos
        {"!", "!", "N", "N", ",", ",", "k", "k", "b", "b", "v", "v", "w", "w", "z", "z"};
        Label firstClicked = null;
        Label secondClicked = null;

    private void AssignIconsToSquares() {
        foreach (Control control in tableLayoutPanel1.Controls) //repite para cada control
        {
            Label iconLabel = control as Label;
            if (iconLabel != null) {
                int randomNumber = random.Next(icons.Count);
                iconLabel.Text = icons[randomNumber];
                icons.RemoveAt(randomNumber);
                iconLabel.ForeColor = iconLabel.BackColor;
            }
        }
        public Form1()
        {
            InitializeComponent();
            AssignIconsToSquares();
        }

    private void label1_Click(object sender, EventArgs e)
    {
        if (timer1.Enabled == true)
            return;
        Label clickedLabel = sender as Label;
        if (clickedLabel != null) {
            if (clickedLabel.ForeColor == Color.Black)
                return;
            CheckForWinner();
            if (firstClicked == null) {
                firstClicked = clickedLabel;
                firstClicked.ForeColor = Color.Black;
                return;
            }
            secondClicked = clickedLabel;
            secondClicked.ForeColor = Color.Black;
            secondClicked = clickedLabel;
            secondClicked.ForeColor = Color.Black;
            if (firstClicked.Text == secondClicked.Text) {
                firstClicked = null;
                secondClicked = null;
                return;
            }
            timer1.Start();
        }
    }

    private void timer1_Tick(object sender, EventArgs e) {
        timer1.Stop();// Stop the timer
        firstClicked.ForeColor = firstClicked.BackColor;// Hide both icons
        secondClicked.ForeColor = secondClicked.BackColor;
        firstClicked = null;
        secondClicked = null;
    }

    private void CheckForWinner(){
        foreach (Control control in tableLayoutPanel1.Controls)
        {
            Label iconLabel = control as Label;
            if (iconLabel != null) {
                if (iconLabel.ForeColor == iconLabel.BackColor)
                    return;
            }
        }
        MessageBox.Show("Terminado. Felicidades!");
        Close(); } }
```

Ejercicio C# Windows Forms: Calculadora humana

Se trata de un Juego/test para resolver una serie de operaciones en un tiempo mínimo.

- Crea un nuevo proyecto: Archivo – Nuevo proyecto (File – New Project)

Aplicación de Windows Forms: **Calculman**

- Añade al formulario un *timer* y los objetos de la figura. →

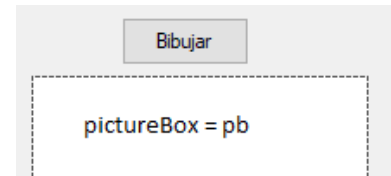
<https://docs.microsoft.com/en-us/visualstudio/ide/tutorial-2-create-a-timed-math-quiz?view=vs-2017>

```
namespace Math_Quiz
{
    public partial class Form1 : Form
    {
        // Create a Random object called randomizer
        Random randomizer = new Random();
        int addend1; // variables for addition
        int addend2;
        int minuend; // variables for subtraction
        int subtrahend;
        int multiplicand; // variables multiplication
        int multiplier;
        int dividend; // variables for division
        int divisor;
        int timeLeft; // variable for remaining time
    public Form1()
        InitializeComponent();
    private void Form1_Load(object sender, EventArgs e)
    {
    }
    //enable the Start button.
    private void startButton_Click(object sender, ...
        StartTheQuiz();
        startButton.Enabled = false;
    // Start the quiz by filling in all of the problem
    public void StartTheQuiz()
        // Fill in the addition problem.
        // Generate two random numbers to add.
        addend1 = randomizer.Next(51);
        addend2 = randomizer.Next(51);
        // into strings so that they can be displayed
        plusLeftLabel.Text = addend1.ToString();
        plusRightLabel.Text = addend2.ToString();
        //NumericUpDown control value is zero before
        sum.Value = 0;
        // Fill in the subtraction problem.
        minuend = randomizer.Next(1, 101);
        subtrahend = randomizer.Next(1, minuend);
        minusLeftLabel.Text = minuend.ToString();
        minusRightLabel.Text = subtrahend.ToString();
        difference.Value = 0;
        // Fill in the multiplication problem.
        multiplicand = randomizer.Next(2, 11);
        multiplier = randomizer.Next(2, 11);
        timesLeftLabel.Text = multiplicand.ToString();
        timesRightLabel.Text = multiplier.ToString();
        product.Value = 0;
        // Fill in the division problem.
        divisor = randomizer.Next(2, 11);
        int temporaryQuotient = randomizer.Next(2, 11);
        dividend = divisor * temporaryQuotient;
        dividedLeftLabel.Text = dividend.ToString();
        dividedRightLabel.Text = divisor.ToString();
        quotient.Value = 0;
        timeLeft = 30; // Start the timer.
        timeLabel.Text = "30 segundos";
        timer1.Start();
    }
}
```

```
Private void timer1_Tick(object sender, ...
if (CheckTheAnswer()) // Time the quiz.
{
    // If CheckTheAnswer() returns true, then the user
    // got the answer right. Stop the timer
    timer1.Stop();
    MessageBox.Show("Felicidades");
    startButton.Enabled = true;
}
else if (timeLeft > 0)
{
    // If CheckTheAnswer() return false, keep counting
    // down. Decrease the time left by one second
    timeLeft--;
    timeLabel.Text = timeLeft + " segundos";
}
else
{
    // If the user ran out of time, stop the timer,
    show
    // a MessageBox, and fill in the answers.
    timer1.Stop();
    timeLabel.Text = "Time's up!";
    MessageBox.Show("Se acabó el tiempo");
    sum.Value = addend1 + addend2;
    difference.Value = minuend - subtrahend;
    product.Value = multiplicand * multiplier;
    quotient.Value = dividend / divisor;
    startButton.Enabled = true;
}
}
// Check the answers to see if the user got
everything right.
private bool CheckTheAnswer()
{
    if ((addend1 + addend2 == sum.Value)
        && (minuend - subtrahend == difference.Value)
        && (multiplicand * multiplier == product.Value)
        && (dividend / divisor == quotient.Value))
        return true;
    else
        return false;
}
// Modify the behavior of the NumericUpDown control
// to make it easier to enter numeric values for
// the quiz.
private void answer_Enter(object sender, EventArgs
e)
{
    // Select the whole answer in the NumericUpDown
    control.
    NumericUpDown answerBox = sender as
    NumericUpDown;
    if (answerBox != null)
    {
        int lengthOfAnswer =
        answerBox.Value.ToString().Length;
        answerBox.Select(0, lengthOfAnswer);
    } } } }
```

Dibujar una línea en un lienzo:

```
private void button1_Click(object sender, EventArgs e)
{
    Graphics lienzo;
    lienzo = pb.CreateGraphics();
    Pen lapiz = new Pen(Color.Black, 3);
    // Create points that define line.
    PointF point1 = new PointF(100.0F, 100.0F);
    PointF point2 = new PointF(500.0F, 100.0F);
    // Draw line to screen.
    lienzo.DrawLine(lapiz, point1, point2);
}
```



Ejercicio for: Muestra los números pares del 1 al 100:

```
using System;
namespace basicos3
{
    class MainClass
    {
        public static void Main(string[] args)
        {
            for (int i = 2; i<=100 ; i+=2)
            {
                Console.WriteLine(i);
            }
            Console.ReadLine();
        }
    }
}
```

Ejercicio 6: creación de objetos en el formulario por código

```
public partial class Form1 : Form //el Form1 es un objeto del tipo Form
{
    public Button button1; //declaro button1 del tipo Button
    public Form1()
    {
        button1 = new Button(); //creo el objeto button1 nuevo en el formulario
        button1.Size = new Size(80, 40); //le pongo su tamaño
        button1.Location = new Point(30, 30); //le pongo su posición
        button1.Text = "Pulsame"; //le pongo un texto
        this.Controls.Add(button1); //lo pongo en el formulario
        button1.Click += new EventHandler(button1_Click); //asigno el evento button1_Click al pulsar
    }
    private void button1_Click(object sender, EventArgs e)
    {
        MessageBox.Show("Hello World"); //muestra el mensaje
    }
}
```

Ejercicio 7: vínculos o llamadas a archivos:

Para ejecutar un proceso del sistema operativo, en versión antiguas utilizábamos la función ShellExecute. En la versión .NET se utiliza Process.Start incluido en la librería System.Diagnostics;

Ejemplo: añadir la librería: `using System.Diagnostics;`

```
private void linkLabel1_LinkClicked(...)
{
    Process.Start(linkLabel1.Text, ""); // o System.Diagnostics.Process.Start(linkLabel1.Text);
}
```


Ejercicio Constructor:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication3
{
class Carta
{
// MIEMBROS
int numero;
int palo;
string[] palos = { "Oros", "Copas", "Espadas",
"Bastos" };
/*
palos[0] es Oros palos[1] es Copas
palos[2] es Espadas palos[3] es Bastos*/
//CONSTRUCTOR
/* Le pasamos por parámetro el número en la variable n
y el palo (valor de 0 a 4) en la variable p */
public Carta(int n, int p)
{
/* Se asignan los valores pasados a los
miembros propios de la clase (objeto) */
numero = n;
palo = p;
}
//MÉTODOS O FUNCIONES
/* Este método escribe el valor de la carta actual */
public void escribeCarta()
{
Console.WriteLine(numero + " de " + palos[palo]);
}
}
class Baraja
{
//MIEMBROS
//Lista de cartas
List<Carta> baraja = new List<Carta>();
//Carta
Carta card;
//CONSTRUCTOR
public Baraja()
{
//Variables usadas en los for
int i, j;
/* 2 bucles for anidados, fuera el de los palos que
realiza el ciclo 4 veces, por 12 el de dentro = 48
cartas de la baraja */
for (j = 0; j < 4; j++)
{
for (i = 0; i < 12; i++)
{
/* Se crea una carta cada vez ya que el valor (i + 1)
va de 1 a 12. La primera vez que este bucle se repite
las 12 veces, crea todas las cartas del palo 0 que es
"Oros". Luego las 12 de "Copas" y así hasta crear
todas de todos los palos. */
card = new Carta(i + 1, j);
//Cada carta creada reescribe la anterior, pero como
se añaden a la lista no perdemos esos datos.
//Añadimos el objeto recién creado a la lista baraja.
baraja.Add(card);
}
}
}
//MÉTODOS
//Escribe el número de cartas que hay en la baraja
public void numeroCartas()
{
Console.WriteLine("En la baraja hay " +
baraja.Count + " cartas.");
}
//Coge la primera carta de la baraja y la elimina de
la lista
public void robaCarta()
{
Console.WriteLine("Has robado una carta: ");
//Se llama al método escribeCarta de la clase carta ya
que baraja[0] es un objeto carta
baraja[0].escribeCarta();
}
}
}
//Se elimina la carta que hemos escrito por si se roba
de nuevo, nos salga la siguiente
baraja.Remove(baraja[0]);
}
//Coge una carta de la posición indicada por parámetro
public void cogeCarta(int n)
{
/* Es como el método anterior, pero en lugar de operar
con la carta baraja[0] se opera con baraja[n] donde
n es el valor dado a la función */
Console.WriteLine("Has cogido la carta de la
posición: " + n);
baraja[n].escribeCarta();
baraja.Remove(baraja[n]);
}
//Coge una carta al azar
public void cogeCartaAlAzar()
{
/* Basado en el método anterior pero ahora el valor n
es obtenido aleatoriamente, debe ser un valor entre 0
y el número de elementos que haya en la lista */
Random r = new Random();
int n = r.Next(0, baraja.Count);
Console.WriteLine("Has cogido una carta al
azar: ");
baraja[n].escribeCarta();
baraja.Remove(baraja[n]);
}
//Escribe todas las cartas que hay en la baraja
public void escribeBaraja()
{
int i;
//Bucle for para recorrer la lista
for (i = 0; i < baraja.Count; i++)
{
//Escribe la posición de la carta (i + 1)
//Utilizamos "Write" en lugar de "WriteLine" para
que no haga un salto de línea
Console.Write((i + 1) + ". ");
}
//Escribe la carta de la posición i de la lista
baraja[i].escribeCarta();
}
}
//Mezcla (baraja) las cartas en la lista
public void Barajar()
{
/* Creamos una variable tipo Random y otra int para
guardar una posición aleatoria de 1 a 48 */
Random r = new Random();
int posicion;
int i;
for (i = 0; i < 48; i++)
{
posicion = r.Next(0, 48);
baraja.Insert(posicion, baraja[0]);
baraja.Remove(baraja[0]);
}
}
}
}
class Program
{
static void Main(string[] args)
{
/* Creamos una baraja, ya no necesitamos crear cartas,
pues en el constructor del objeto baraja se
crean las 48 cartas de la baraja */
Baraja baraja = new Baraja();
/* Llamamos a los métodos como queramos, os dejo esta
secuencia, pero jugad con el orden y valores para ver
otros resultados */
baraja.escribeBaraja();
baraja.Barajar();
baraja.escribeBaraja();
baraja.robaCarta();
baraja.cogeCarta(5);
baraja.cogeCartaAlAzar();
baraja.numeroCartas();
Console.ReadKey();
}
}
}

```

Ejercicios de C# en varios IDEs (para Unity, Visual Studio, Visual Studio Code)

Ejercicio 1: Mostrar por consola "Hola Mundo"

Para Unity:

- ▶ Creamos un nuevo proyecto: + **New Project**
- ▶ Sobre la carpeta *Assets*, pulsamos el botón secundario del mouse y escogemos: Create – C#Script: **Ejer1**
- ▶ Pulsamos doble clic para abrirlo con Visual Studio (o el antiguo Monodevelop)
- ▶ Escribimos dentro de la función Start()

```
// ---- Opción 1 sin variable:  
{  
    Debug.Log("Hola Mundo");  
}
```

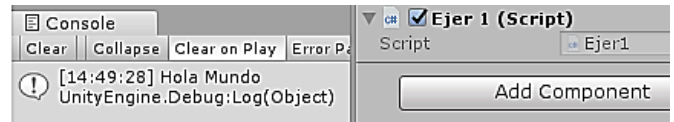
- ▶ Guardamos todo y le asignamos este "script" a cualquier objeto en escena como la cámara, Arrastrando el script *Ejer1* sobre Main Camera. Luego comprobamos en el *Inspector* que en la cámara aparece el Script *Ejer1* como un componente.

- ▶ Reproducimos en Unity ▶

y debería mostrarse el mensaje en la Console:

- ▶ Si todo va bien, cambiamos el código por este otro:

```
// ---- Opción 2 con variable:  
{  
    string mensaje;           //declaro una nueva variable del tipo texto  
    mensaje = "Hola mundo";  //inicio la variable asignándole un texto  
    Debug.Log(mensaje);      //escribe el mensaje en la consola    }  
}
```



Nota: Si cambias el nombre del script, deberás cambiar el nombre de la clase en el código: `public class Ejer1`
Una clase es un conjunto de funciones y métodos para los objetos, definidos por el usuario.

Para Visual Studio:

- ▶ Creamos un nuevo proyecto: Archivo > Nuevo > Proyecto (*New Project*)
- ▶ Escogemos: Aplicación de consola (.NET Framework): Nombre: **Ejer1** Ubicación: `C:\Users\Usuario\source\repos`
- ▶ Una vez escrito el código, pulsamos el botón **Iniciar** ▶ para probarlo.

Para Visual Studio Code:

- ▶ En el Terminal, escribe el comando: **dotnet new console** para iniciar un nuevo program.cs
- ▶ Una vez escrito el código, escribimos en la consola: **dotnet run** para probarlo.

Código:

Ejemplo 1 sin variable:

```
namespace Ejer1  
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            Console.WriteLine("Hola mundo"); // muestra mensaje en la consola  
            Console.WriteLine("Hola");      // muestra mensaje y salta de línea  
            Console.ReadKey();              // pausa de espera tecla  
        }  
    }  
}
```

- ▶ Si todo va bien, cambiamos el código por este otro:

Opción 2 con variable:

```
namespace Ejer1  
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            string mensaje;           //declaro una nueva variable del tipo texto  
            mensaje = "Hola mundo";  //inicio la variable asignándole un texto  
            Console.WriteLine(mensaje); //escribe el mensaje en la consola  
            Console.ReadKey();        //pausa de espera tecla  
        }  
    }  
}
```

- ▶ Si todo va bien, guarda el proyecto o solución.

Ejercicio 2: Declarar, iniciar y mostrar variablesModo Unity:

- ▶ Creamos un nuevo *script*: Create – C#Script: **Ejer2**
- ▶ Escribimos dentro de la función Start()


```

{
    int nivel = 2;           // variable numérica entera. Puede ser positivo o negativo
    float nota = 5.5f;     // variable numérica decimal. El separador decimal es un punto
    float puntos = 0f;     // variable numérica decimal. El valor siempre finaliza con f
    string nombre = "Pedro"; // variable de cadena de texto. El texto entre comillas " "
    bool vivo = true;      // variable lógica booleaniana. Verdadera o falsa.
    puntos = nota * nivel;
    if (puntos > 5) vivo = true; //Otro modo es: (puntos > 5)? True:false;
    else vivo = false;
    Debug.Log("Hola " + nombre);
    Debug.Log("Tu nivel es " + nivel);
    Debug.Log("Tu nota es de: " + nota);
    Debug.Log("Tu puntuación es de " + puntos);
    if (vivo == true) Debug.Log("Estas vivo"); //usar doble igual para comparar ==
    else Debug.Log("Estas muerto");
}

```
- ▶ Guardamos el *script* y se lo asignamos a la cámara, quitando el scrip de la cámara anterior.
- ▶ Reproducimos en Unity ▶ y comprobamos en la Consola. Si todo va bien, guardamos la escena.

Modo VisualStudio o VS Code:

```

class Program
{
    static void Main(string[] args)
    {
        int nivel = 2;           // variable numérica entera. Puede ser positivo o negativo
        float nota = 5.5f;     // variable numérica decimal. El separador decimal es un punto
        float puntos = 0f;     // variable numérica decimal. El valor siempre finaliza con f
        string nombre = "Pedro"; // variable de cadena de texto. El texto entre comillas " "
        bool vivo = true;      // variable lógica booleaniana. Verdadera o falsa.
        puntos = nota * nivel;
        if (puntos > 5) vivo = true; //Otro modo es: (puntos > 5)? True:false;
        else vivo = false;
        Console.Write("Hola " + nombre+"\n");
        Console.Write("Tu nivel es " + nivel + "\n");
        Console.Write("Tu nota es de: " + nota + "\n");
        Console.Write("Tu puntuación es de " + puntos + "\n");
        if (vivo == true) Console.Write("Estas vivo"); //usar doble igual para comparar ==
        else Console.Write("Estas muerto");
        Console.ReadKey(); //pausa de espera tecla
    }
}

```

Ejercicio 3: Declarar, iniciar y mostrar arrays

Arrays, vectores o matrices: es un conjunto de variables del mismo tipo ordenadas en filas y columnas y representadas por el nombre del array. Útil para almacenar varios elementos del mismo tipo: nombres de jugadores, niveles, posiciones...

Modo Unity:

- ▶ Creamos un nuevo *script*: Create – C#Script: **Ejer3**
- ▶ Escribimos dentro de la función Start()


```

{
    int[] puntos = new int[10];           //declarados 10 elementos del tipo numérico
    entero
    int[] nivel = new int[3] { 1, 2, 3 }; //declarados e inicializados 3 elementos
    float[] notas= { 5f, 6.5f, 7f };     //inicializados tres elementos del tipo
    decimal
    string[] eltiempo = { "lluvia", "viento","sol"}; //inicializados tres elementos de texto
    int numeroAleatorio = Random.Range(0, 2); //iniciamos un valor aleatorio del 0 al 2
    Debug.Log(puntos.Length);             //muestra la longitud o cantidad de elementos
    puntos[0] = 5;                         //asigna un cinco al primer elemento
    Debug.Log(puntos[0]);                  //muestra el valor del primer elemento
    Debug.Log("Mañana va a hacer "+ eltiempo[numeroAleatorio]);
}

```
- ▶ Guardamos el *script* y se lo asignamos a la cámara, quitando el *script* de la cámara anterior.
- ▶ Reproducimos en Unity ▶ y comprobamos en la Consola. Si todo va bien, guardamos la escena.

Modo consola Visual Studio C#:

```
{
class Program
{
    static void Main(string[] args)
    {
        int[] puntos = new int[2]; //Array con 2 números enteros, 1 para cada jugador
        int[] nivel = new int[3] { 1, 2, 3 }; //declarados e inicializados 3 elementos
        float[] notas= { 5f, 6.5f, 7f }; //inicializados tres elementos del tipo decimal
        string[] eltiempo = { "lluvia", "viento","sol"}; //inicializados tres elementos de texto
        int num=3;
        Random rnd = new Random(); //creamos objeto rnd del tipo Random
        int numeroAleatorio = rnd.Next(num); //iniciamos un valor aleatorio del 0 al 2
        Console.WriteLine("Jugadores: "+ puntos.Length); //muestra la longitud o cantidad de elementos
        puntos[0] = 5; //asigna un cinco al primer elemento
        puntos[2] = 7; //asigna un cinco al sgundo elemento
        Console.WriteLine("Puntos Jugador 1: " +puntos[0]); //muestra el valor del primer elemento
        Console.WriteLine("Puntos Jugador 2: " +puntos[1]); //muestra el valor del segundo elemento
        Console.WriteLine("Mañana va a hacer " + eltiempo[numeroAleatorio]);
    }
}
}
```

- › Para probarlo. En Visual Studio: Pulsa en **Iniciar** ►
- › **Para Visual Studio Code:** En el Terminal, escribe el comando: **dotnet run**

Ejercicio 5: Operador condicional y valor aleatorio

Expresión condicional: `if(condición){caso_verdadero;} else {caso_falso;}`

Para Unity

- › Creamos un nuevo *script*: Create – C#Script: **Ejer4**
- › Borramos la función Start() y añadimos este código:

```
void Start()
{
    if (Random.Range(0,2)==0)
    { mensaje = "cara"; }
    else
    { mensaje = "cruz"; }
    Debug.Log("Has sacado " + mensaje); // En VS equivale a Console.Write("Has sacado " + mensaje);
}
```
- › Guardamos el *script* y se lo asignamos a la cámara, quitando el scrip de la cámara anterior.
- › Reproducimos en Unity ► y comprobamos en la consola. Si todo va bien, guardamos la escena.

Para Consola Visual Studio:

```
class Program
{
    static void Main(string[] args)
    {
        String mensaje;
        int min=0;
        int max=2;
        Random aleatorio = new Random();
        if (aleatorio.Next(min,max)==0)
        { mensaje = "cara"; }
        else
        { mensaje = "cruz"; }
        Console.Write("Has sacado " + mensaje);
    }
}
```

- › Para probarlo. En *Visual Studio*: Pulsa en **Iniciar** ► En *Visual Studio Code*, comando: **dotnet run**

Ejercicio 5: Operador condicional y expresiones

Expresión condicional en una sola línea: (condición) ? caso_verdadero : caso_falso

- ▶ Creamos un nuevo *script*: Create – C#Script: **Ejer5**
- ▶ Borramos la función Start() y añadimos este código:

```
public class Ejer5 : MonoBehaviour
{
    int vidas = 3;
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space)) //si se ha pulsado la tecla espaciadora
        {
            vidas--; //quitamos una vida
            string mensaje = (vidas <= 0) ? "Game over" : "Te quedan "+vidas +" vidas";
            Debug.Log(mensaje);
        }
    }
}
```

- ▶ Guardamos el *script* y se lo asignamos a la cámara, quitando el *script* de la cámara anterior.
- ▶ Reproducimos en Unity ▶ y comprobamos en la consola. Si todo va bien, guardamos la escena.

Creación de aplicaciones de consola de .NET con Visual Studio Code

Requisitos previos

- Visual Studio Code con la extensión de C# instalada (desde el Marketplace de extensiones de VS Code).
- SDK de .NET 5.0 o posterior

Creación de la aplicación

- › Inicia Visual Studio Code.
- › Seleccione **Archivo ▶ Abrir carpeta**
- › En el cuadro de diálogo Abrir carpeta, **crea una carpeta de HelloWorld** y haz clic en Seleccionar carpeta.
- › De forma predeterminada, el nombre de la carpeta se convierte en el nombre del proyecto y del espacio de nombres. Se supone que el espacio de nombres del proyecto es HelloWorld.
- › Para abrir el Terminal en Visual Studio Code, selecciona: **Ver ▶ Terminal** en el menú principal.
- › Se abre el Terminal con el símbolo del sistema en la carpeta HelloWorld.
- › En el Terminal, escriba este comando: **dotnet new console**

La plantilla crea una aplicación "Hola mundo" sencilla. Llama al método `Console.WriteLine(String)` para mostrar "Hello World!" en la ventana de la consola.

El código de plantilla define una clase, `Program`, con un solo método, `Main`, que toma una matriz de `String` como argumento: **`static void Main(string[] args)`**

`Main` es el punto de entrada de la aplicación, el método que se llama automáticamente mediante el tiempo de ejecución cuando inicia la aplicación. Los argumentos de línea de comandos proporcionados cuando se inicia la aplicación están disponibles en la matriz `args`

Ejecutar la aplicación

Ejecuta este comando en el Terminal: **dotnet run**

Mejora de la aplicación

- › En el panel lateral, haz clic en el archivo **Program.cs** para abrirlo. La primera vez que se abre un archivo de C# en Visual Studio Code, se carga OmniSharp en el editor.
Selecciona Sí cuando Visual Studio Code pida que agregues los recursos que faltan para compilar.
- › Reemplaza el contenido del método `Main` en `Program.cs`, por este otro código:

```
{
    Console.WriteLine("Cómo te llamas?");
    var nombre = Console.ReadLine();
    var hoy = DateTime.Now;
    Console.WriteLine($"{Environment.NewLine}Hola, {nombre}, estamos a {hoy:d} a las {hoy:t}!");
    Console.Write($"{Environment.NewLine}Pulsa una tecla para salir...");
    Console.ReadKey(true);
}
```

Este código muestra un mensaje en la ventana de la consola y espera a que el usuario escriba un nombre y, luego, presione Entrar. Almacena esta cadena en una variable denominada `nombre`. También recoge el valor de `DateTime.Now`, que contiene la hora local actual, y se lo asigna a una variable denominada `hoy`. Asimismo, muestra estos valores en la ventana de la consola. Por último, muestra un mensaje en la ventana de la consola y llama al método `Console.ReadKey(Boolean)` para esperar a la entrada del usuario.

`Environment.NewLine` es una manera independiente de la plataforma y del lenguaje de representar un salto de línea. Las alternativas son `\n` en C# y `\r\n` en Visual Basic.

El signo de dólar (\$) delante de una cadena permite colocar expresiones como nombres de variable entre llaves en la cadena. El valor de la expresión se inserta en la cadena en lugar de la expresión. Esta sintaxis se conoce como cadenas interpoladas.

- › Guarda los cambios y ejecuta el programa otra vez: **dotnet run**
Presiona cualquier tecla para salir de la aplicación.

Punto de interrupción y depuración del programa:


- › Abra el archivo Program.cs.
- › Establece un punto de interrupción en la línea que muestre el nombre, la fecha y la hora; para ello, haz clic en el margen izquierdo de la ventana de código o presiona F9 o selecciona: Ejecutar > Alternar punto de interrupción. Visual Studio Code marca la línea donde se establece el punto de interrupción con un punto rojo en el margen.

Configuración para la entrada de terminal:

Como el punto de interrupción no funciona con la consola-terminal interno, debemos de cambiarlo a integratedTerminal:

- › Abre: .vscode/launch.json.
- › Cambie la opción console de internalConsole a integratedTerminal: "console": "integratedTerminal",
- › Guarda los cambios.

Iniciar depuración

- › Abre la vista *Depurar* mediante el icono de depuración que hay en el menú de la izquierda. 
- › Selecciona la flecha verde en la parte superior del panel, junto a .NET Core Launch (console). Otras maneras son presionar F5 o elegir Ejecutar > Iniciar depuración en el menú. La ejecución del programa se detiene cuando llega al punto de interrupción. La sección Variables locales de la muestra los valores de las variables actuales. Pulsa en ▶ Continue (F5) para seguir y salir de la pausa.

Creación de aplicaciones C y C++ con Visual Studio Code

- Visual Studio Code con la extensión de C++ instalada (desde el Marketplace de extensiones de VS Code).
- Instalar MinGW con el compilador, comprueba que está añadido: g++.exe cpp.exe dbd.exe en carpeta bin
- Agregaremos a MinGW a las variables del entorno del sistema, esto nos permitirá compilar desde el command prompt o CMD de Windows.
 - Para compilar: Añadir el script: launch.json
 - Para linkar: Añadir el script: task.json
 - Para ejecutar: Instalar Code Runner.

Launch.json	Task.json
<pre> { "version": "0.2.0", "configurations": [{ "name": "g++.exe - Build and debug active file", "type": "cppdbg", "request": "launch", "program": "\${fileDirname}\\\${fileBasenameNoExtension}.exe", "args": [], "stopAtEntry": false, "cwd": "\${workspaceFolder}", "environment": [], "externalConsole": false, "MIMode": "gdb", "miDebuggerPath": "C:\\mingw\\bin\\gdb.exe", "setupCommands": [{ "description": "Enable pretty-printing for gdb", "text": "-enable-pretty-printing", "ignoreFailures": true }], "preLaunchTask": "C/C++: g++.exe build active file" }] } </pre>	<pre> { "version": "2.0.0", "tasks": [{ "type": "shell", "label": "C/C++: g++.exe build active file", "command": "C:\\mingw\\bin\\g++.exe", "args": ["-g", "\${file}", "-o", "\${fileDirname}\\\${fileBasenameNoExtension}.exe"], "options": { "cwd": "\${workspaceFolder}" }, "problemMatcher": ["\$gcc"], "group": { "kind": "build", "isDefault": true } }] } </pre>

INVERTIR NÚMERO DE DOS CIFRAS

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            int NUM, AUX, DEC, UNI;
            string linea;
            Console.WriteLine ("INGRESE NÚMERO DE DOS
CIFRAS :");
            linea = Console.ReadLine();
            NUM = int.Parse(linea);
            DEC = NUM/10;
            UNI = NUM % 10;
            AUX = (UNI * 10) + DEC;
            Console.WriteLine("NÚMERO INVERTIDO ES: " +
AUX);
            Console.WriteLine("Pulse una Tecla:");
            Console.ReadLine();
        }
    }
}
```

OPERACIONES BÁSICAS

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            int NUM1, NUM2, RESULT;
            string linea;
            Console.Write("PRIMER NÚMERO :");
            linea = Console.ReadLine();
            NUM1 = int.Parse(linea);
            Console.Write("SEGUNDO NÚMERO :");
            linea = Console.ReadLine();
            NUM2 = int.Parse(linea);
            Console.WriteLine();
            RESULT = NUM1 + NUM2;
            Console.WriteLine("LA SUMA ES {0}: ", RESULT);
            RESULT = NUM1 - NUM2;
            Console.WriteLine("LA RESTA ES: {0} - {1} = {2} ",
NUM1, NUM2, RESULT);
            RESULT = NUM1 * NUM2;
            Console.WriteLine("LA MULTIPLICACIÓN ES: " +
RESULT);
            RESULT = NUM1 / NUM2;
            Console.WriteLine("LA DIVISIÓN ES: " + RESULT);
            RESULT = NUM1 % NUM2;
            Console.WriteLine("EL RESIDUO ES: " + RESULT);
            Console.Write("Pulse una Tecla:");
            Console.ReadLine();
        }
    }
}
```

COMPRA EN RESTAURANT

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication1
{
class Program
{
static void Main(string[] args)
{
byte CANB, CANH, CANP;
double APAGAR;
string linea;
const double PRECIOB = 0.8;
const double PRECIOH = 2;
const double PRECIOP = 1.2;
Console.WriteLine("CANTIDAD DE HAMBURGUESAS :");linea =
Console.ReadLine();
CANH = byte.Parse (linea);
Console.WriteLine("CANTIDAD DE PAPAS :");linea =
Console.ReadLine();
CANP = byte.Parse (linea);
Console.WriteLine("CANTIDAD DE BEBIDAS :");linea =
Console.ReadLine();
CANB = byte.Parse (linea);
Console.WriteLine();
APAGAR = (CANH * PRECIOH) + (CANP * PRECIOP) +
(CANB * PRECIOB);
Console.WriteLine("VALOR A PAGAR: " + APAGAR);
Console.WriteLine("Pulse una
Tecla:");Console.ReadLine();
}
}
}

```

FUNCIONES BÁSICAS LIBRERÍA MATH

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication1
{
class Program
{
static void Main(string[] args)
{
int NUM1;
string linea;
long RESULT;
Console.WriteLine("DIGITE UN NÚMERO :");
linea = Console.ReadLine();
NUM1 = int.Parse(linea);
RESULT = Math.Abs(NUM1);
Console.WriteLine("VALOR ABSOLUTO : " + RESULT);
Console.WriteLine("POTENCIA : " + Math.Pow(NUM1,
3));
Console.WriteLine("RAIZ CUADRADA : " +
Math.Sqrt(NUM1));
Console.WriteLine("SENO : " + Math.Sin(NUM1 *
Math.PI / 180));
Console.WriteLine("COSENO : " + Math.Cos(NUM1 *
Math.PI / 180));
Console.WriteLine("NÚMERO MÁXIMO : " +
Math.Max(NUM1, 50));
Console.WriteLine("NÚMERO MÍNIMO : " +
Math.Min(NUM1, 50));
Console.WriteLine("PARTE ENTERA : " +
Math.Truncate(18.78));
Console.WriteLine("REDONDEO : " +
Math.Round(18.78));
Console.WriteLine("Pulse una Tecla:");
Console.ReadLine();
}
}
}

```

FORMATOS DE SALIDA

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication1
{
class Program
{
static void Main(string[] args)
{
double BASE, ALTURA, RESULT;
string linea;
Console.WriteLine("DIGITE LA BASE :"); linea =
Console.ReadLine();
BASE = double.Parse (linea);
Console.WriteLine("DIGITE LA ALTURA:"); linea =
Console.ReadLine();
ALTURA= double.Parse (linea);
RESULT = (BASE * ALTURA) / 2;
Console.WriteLine("AREA TRIANGULO : " +
String.Format("{0:###.00}", RESULT));
Console.WriteLine("AREA TRIANGULO : " +
String.Format("{0:c}", RESULT));
Console.WriteLine("AREA TRIANGULO : " +
String.Format("{0:f}", RESULT));
Console.WriteLine("AREA TRIANGULO : " +
String.Format("{0:g}", RESULT));
Console.WriteLine();
Console.WriteLine("HOY ES: " + String.Format("Hoy
es {0:F}", DateTime.Now));
Console.WriteLine("HOY ES: " + String.Format("Hoy
es {0:ddd}{0:dd/MM/yyyy}", DateTime.Now));
Console.WriteLine("Pulse una Tecla:");
Console.ReadLine();
} } }

```

MAYOR DE DOS NÚMEROS

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication1
{
class Program
{
static void Main(string[] args)
{
int NUM1, NUM2;
string linea;
Console.WriteLine("NÚMERO 1 :"); linea =
Console.ReadLine();
NUM1 = int.Parse(linea);
Console.WriteLine("NÚMERO 2 :"); linea =
Console.ReadLine();
NUM2 = int.Parse(linea);
if ((NUM1 > NUM2))
{Console.WriteLine("{0} ES MAYOR QUE {1}", NUM1,
NUM2);
}
else
{
if ((NUM1 == NUM2))
{Console.WriteLine("{0} ES IGUAL A {1}", NUM1,
NUM2);
}
else
{Console.WriteLine("{0} ES MENOR QUE {1}", NUM1,
NUM2);
} } }
Console.WriteLine();
Console.WriteLine("OTRA MANERA");
string RESULT;
if (NUM1 > NUM2 )
{
RESULT = "MAYOR";
}
else

```

```

if (NUM1 == NUM2 )
{
RESUL = "IGUAL";
}
else
{
RESUL = "MENOR";
}
Console.WriteLine("{0} ES {1} QUE {2}", NUM1,
RESUL, NUM2);
Console.Write("Pulse una Tecla:");
Console.ReadLine();
} } }

```

MAYOR DE TRES NÚMEROS

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication1
{
class Program
{
static void Main()
{
byte MAY, MEN, NUM1, NUM2, NUM3;
string linea;
Console.Write("NÚMERO 1 :");
linea = Console.ReadLine();
NUM1 = byte.Parse(linea);
Console.Write("NÚMERO 2 :");
linea = Console.ReadLine();
NUM2 = byte.Parse(linea);
Console.Write("NÚMERO 3 :");
linea = Console.ReadLine();
NUM3 = byte.Parse(linea);
MAY = NUM1; MEN = NUM1;
if ((NUM2 > MAY)) MAY = NUM2;
if ((NUM3 > MAY)) MAY = NUM3;
if ((NUM2 > MEN)) MEN = NUM2;
if ((NUM3 < MEN)) MEN = NUM3;
Console.WriteLine("MAYOR ES:" + MAY);
Console.WriteLine("MENOR ES:" + MEN);
Console.WriteLine("Pulse una Tecla:");
Console.ReadLine();
} } }

```

DESGLOSE DE BILLETES

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication1
{
class Program
{
static void Main()
{
int CAN, C100, C50, C20, C10, C5;
C100 = 0;
C50 = 0;
C20 = 0;
C10 = 0;
C5 = 0;
string linea;
Console.Write("DIGITE UNA CANTIDAD :");
linea = Console.ReadLine();
CAN = int.Parse (linea);
if( (CAN >= 100) )
{
C100 = (CAN / 100);
CAN = CAN - (C100 * 100);
}
if( (CAN >= 50) )
{
C50 = (CAN / 50);
CAN = CAN - (C50 * 50);
}
if( (CAN >= 20) )

```

```

{
C20 = (CAN / 20);
CAN = CAN - (C20 * 20);
}
if( (CAN >= 10) )
{
C10 = (CAN / 10);
CAN = CAN - (C10 * 10);
}
if( (CAN >= 5) )
{
C5 = (CAN / 5);
CAN = CAN - (C5 * 5);
}
Console.WriteLine("BILLETES DE A 100: " + C100);
Console.WriteLine("BILLETES DE A 50 : " + C50);
Console.WriteLine("BILLETES DE A 20 : " + C20);
Console.WriteLine("BILLETES DE A 10 : " + C10);
Console.WriteLine("BILLETES DE A 5 : " + C5);
Console.WriteLine("BILLETES DE A 1 : " + CAN);
Console.Write("Pulse una Tecla:");
Console.ReadLine();
} } }

```

Ejemplo bucle for

```

// Muestra los números pares del 1 al 100:
using System;
namespace basicos3
{
class Program
{
public static void Main()
{
for (int i = 2; i<=100 ; i+=2)
{
Console.WriteLine(i);
}
Console.ReadLine();
}
}
}

```

Mostar un menú con bucle do - while

```

using System;
namespace carpeta
{
class Program
{
static void Main()
{
int opcion=0;
do //inicio del bucle
{
mostrar_menu(); //llama a la función menu
opcion = Convert.ToInt32( Console.ReadLine() );
} while (opcion !=0); //condición del bucle
}

static void mostrar_menu() //función mostrar menu
{
Console.Clear();
Console.WriteLine("1. Instrucciones");
Console.WriteLine("2. Cargar partida");
Console.WriteLine("3. Grabar partida");
Console.WriteLine("4. Jugar");
Console.WriteLine("0. Salir");
Console.WriteLine();
Console.Write("Escoge opción: ");
}
}
}

```