

Tema 1: Introducción al C#

Asignatura: Desarrollo web en entorno servidor
CS Desarrollo de Aplicaciones Web



Introducción

- En este capítulo veremos aspectos como:
 - Variables
 - Operadores y sentencias de selección simples
 - Sentencias de repetición
 - Arrays
 - Funciones y métodos

El lenguaje C#

- **Nuevo lenguaje diseñado expresamente para .NET**
- Autores: Scott Wiltamuth & Anders Hejlsberg (Este último creó el Turbo Pascal y Delphi)
- **Es el lenguaje nativo de .NET** (gran parte de la librería de clases se ha escrito en C#)

El lenguaje C#

- Características generales
 - **Orientado a objetos**, sencillo, fácil de usar
 - **Recoge lo mejor de Java y C++** (productividad + potencia)
 - Tiene una sintaxis parecida a Java, pero lo mejora con nuevas posibilidades (propiedades, indexadores, redefinición de operadores, etc.)

El lenguaje C#

- Realmente es un lenguaje OO (encapsulación, herencia y polimorfismo).
- El modelo de objetos es homogéneo (todo el código se escribe en una clase y todos los tipos de datos son objetos)
- Permite el uso de estructuras (clases que se almacenan en la pila) -> más rapidez

El lenguaje C#

- **Es fuertemente tipado.** Se controlan la conversiones de tipos exhaustivamente -> menos errores, mayor seguridad
- **Recolector de basura automático.** Libera al programador de eliminar las referencias a los objetos que no se usan
- Soporte nativo para delegados y eventos. Muy útil en las aplicaciones con interfaces gráficas.

El lenguaje C#

- Incorpora propiedades mejorando el acceso a los miembros de una clase
- Se permite la redefinición de operadores para los tipos definidos por el usuario mejorando la legibilidad y la escritura
- Admite atributos: informaciones adicionales en las clases útiles para el compilador.

El lenguaje C#. Un programa en C#

- **Pasos: edición, compilación y ejecución**
- **Usaremos .NET Framework SDK**
- Necesitamos un editor de texto ASCII (por ejemplo el Notepad de Windows)
- Se teclea el código y se guarda en un archivo de disco con extensión '.cs'.

El lenguaje C#. Depuración

- La depuración **permite ejecutar la aplicación paso a paso por sentencias o métodos**
- Nos permite observar el flujo de ejecución (traza) y los resultados intermedios
- Para depurar:
 - Depurar -> Paso a paso por instrucciones (F11) o Paso a paso por procedimientos (F10)

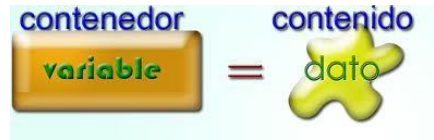
Actividad 1

- 1. Crea una aplicación de consola C# en Visual Studio*
- 2. Haz que muestre “Hola mundo!” por pantalla*
- 3. Parar la depuración en la instrucción donde se muestra por pantalla*

5 MINUTOS

Concepto de variables.

- Los programas necesitan datos para poder realizar los cálculos que les pedimos.
- Estos datos deben ser almacenados para que puedan ser accedidos por el programa cuando los necesite.
- Las variables estarán formadas por un espacio en la memoria y un nombre para referirse a dicho espacio:



- La sintaxis para **declarar una variable** en C# es la siguiente:

`[Tipo_Variable] [nombre_variable] ; → int x;`

- Tenemos una variable de nombre **x** y de tipo Integer.

`double` dob = 19.6 → Mal nombre

`double` temperatura = 19.6 → Bien

- Ojo: **Los nombres de las variables deben ser representativos del valor que estamos guardando.**

Concepto de variables

- Existen diferentes tipos de datos que ocuparán más o menos memoria.
- En general pueden clasificarse en :
 - **numéricos:**
 - Enteros
 - Reales
 - **alfanuméricos**
 - letras
 - caracteres especiales
 - números: dni , número de teléfono ...
 - Cualquier mezcla de los anteriores
 - **booleanos:** Verdadero o falso.

Tipos de datos en C#

- En función del dato que queramos almacenar en memoria utilizaremos un tipo de variables u otras.

Tipo C#	Espacio	Valores	Valor por defecto
Byte	1 Byte	0 a 255	0
Short	2 Bytes	de -32,768 a 32,767.	0
Integer (int)	4 Bytes	de -2,147,483,648 a 2,147,483,647	0
Long	8 Bytes		0
Single (float)	4 Bytes	Decimales precisión simple	0
Double	8 Bytes	Decimales precisión doble	0
Decimal	16 Bytes	Más capacidad, más precisión	0
Char	2 Bytes	Un caracter	"
String	-----	Cadena de caracteres	Null
Boolean	2 Bytes	Valor verdadero o falso	False

Declaración e inicialización de variables.

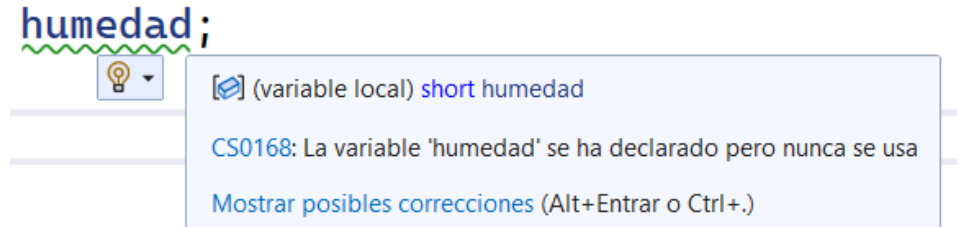
- Las variables deben de ser declaradas al principio del programa, evitando que aparezcan dispersas por el código.

```
int x, y, z;  
short temp, regist, humedad;
```

- Cuando queramos declarar varias variables del mismo tipo podemos hacerlo separando sus nombres por comas.
- Cuando declaramos una variable, esta almacena un valor por defecto.

Declaración e inicialización de variables.

- Si no le damos otro valor en el programa, nos avisa de que no estamos utilizando esa variable, subrayándola en verde. Esto es simplemente una advertencia, no un error.



- En caso de que queramos asignarle otro valor utilizaremos el operador =

```
//Inicializo la variable
```

```
double dob = 19.6;
```

```
//Modifico el valor de la variable
```

```
dob = 50.6;
```

Nombres de variables:

- Los nombres de las variables deben cumplir los siguientes requisitos:
 - No deben contener espacios.
Ej: sueldo neto (Mal) → sueldoNeto (Bien)
 - No pueden comenzar por números , aunque si pueden contener números.
Ej: 1x (Mal) → x1 (Bien)
 - No pueden contener caracteres como *, \$, &, . El guión bajo si está aceptado.
Ej: &x (Mal) → _x (Bien)
 - Cuando el nombre de una variable está formado por dos palabras. la segunda comienza en mayúsculas.
Ej: presionVapor , sueldoNeto, valorAbsoluto, raizNumero

Ejemplo 2.1: Suma de tres variables

- Construiremos un programa que almacene 3 números en tres variables de tipo short y calcule la suma guardándola en una variable de tipo Integer.
- A continuación el programa informará al usuario del valor de la suma:

```
//Declaramos las variables que vayamos a utilizar
//Tres variables de tipo short para los números

short num1, num2, num3;


//Inicializamos las variables a los valores que queremos sumar
num1 = 25;
num2 = 30;
num3 = 35;

//Una variable de tipo Integer para guardar el resultado, la suma
int suma;

//Indicamos la suma de los 3 números
suma = num1 + num2 + num3;

//Mostramos por pantalla la suma
Console.WriteLine("La suma es : " + suma);
Console.ReadLine();
```

Para mezclar texto con variables
debemos de utilizar el operador +



Constantes.

- Cuando queremos guardar un valor que sabemos que nunca va a ser modificado, podremos guardarlo en una constante.
- La sintaxis es similar a la vista para las variables:

```
const double pi = 3.14151623;
```

- Una constante ya nunca podrá ser modificada ya que dará error de compilación:

```
const double pi = 3.14151623;
```

```
pi = 9;
```

 (constante local) double pi = 3.14151623

CS0131: La parte izquierda de una asignación debe ser una variable, una propiedad o un indizador

[Mostrar posibles correcciones \(Alt+Entrar o Ctrl+.\)](#)

Lecturas de datos por teclado.

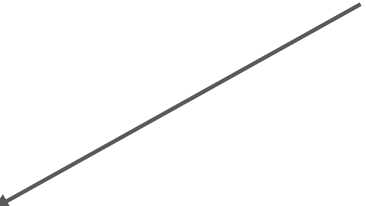
- Para solicitar al usuario que introduzca los datos por teclado debemos realizar 2 pasos:
 - 1º: Enviaremos un mensaje informándole del dato que queremos que introduzca
 - 2º: Guardaremos el dato en la variable mediante la función **Console.ReadLine()**

Lecturas de datos por teclado.

- Para solicitar al usuario que introduzca los datos por teclado debemos realizar 2 pasos:

```
string nombre;  
int edad;  
  
//Solicitamos al usuario el dato  
Console.WriteLine("Introduzca el nombre");  
  
//Leemos el dato con ReadLine  
nombre = Console.ReadLine();  
  
//Solicitamos la edad  
Console.WriteLine("Introduzca su edad: ");  
edad = Convert.ToInt32(Console.ReadLine());  
  
//Mostramos los datos al usuario, simplemente para verificar  
Console.WriteLine("Bienvenido: " + nombre + " tiene usted " + edad + " años");  
Console.ReadLine();
```

La función `Console.ReadLine` siempre lee el dato como si fuera un **String**, por eso cuando queramos leer otro tipo de variables, como en este caso la edad, deberemos convertirla en el tipo adecuado.



Ejemplo 2.2: Lectura datos teclado

- Solicitaremos todos los datos de un determinado cliente: DNI, Nombre, edad y saldo:

```
//Declaramos variables que necesitamos
string dni, nombre;
short edad;
int saldo;

//Solicitamos los datos
Console.WriteLine("Introduzca el DNI:");

//Guardamos el dato que va a introducir en la variable DNI
dni = Console.ReadLine();
Console.WriteLine("Introduzca el nombre: ");
nombre = Console.ReadLine();
Console.WriteLine("Introduzca la edad: ");

//Guardamos la edad convirtiéndola previamente en una variable de tipo Short
edad = Convert.ToInt16(Console.ReadLine());
Console.WriteLine("Introduzca el saldo: ");
saldo = Convert.ToInt32(Console.ReadLine());
```

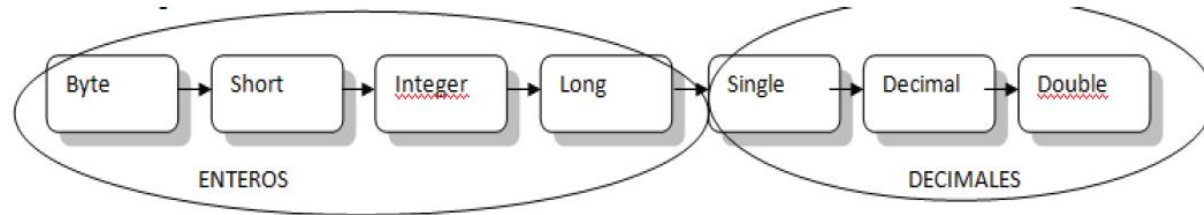
Ejemplo 2.2: Lectura datos teclado

- Mostraremos esos datos por pantalla:

```
Console.WriteLine("Sus datos son los siguientes: ");  
Console.WriteLine("DNI: " + dni);  
Console.WriteLine("Nombre: " + nombre);  
Console.WriteLine("Edad: " + edad);  
Console.WriteLine("Saldo: " + saldo);  
Console.ReadLine();
```

Conversión de tipos.

- Cada variable ocupa un espacio en memoria en función del tipo de dato que queramos guardar en ella.
- Esto les proporciona mayor o menor precisión según sea el caso.
- Podemos ordenar las variables de menor a mayor precisión de la siguiente manera:



Conversión de tipos.

- Las variables de menor precisión ***siempre podrán ser guardadas*** en las variables de mayor precisión(salvo la excepción de Decimal)

```
byte numByte = 220;  
short numShort = numByte;  
int numInt = numShort;  
long numLong = numInt;
```

```
float numSingle = 10.5f;  
double numDouble = numSingle;  
decimal numDecimal = numDouble; //Da error
```

- Este tipo de conversiones se conocen como conversiones implícitas porque es el lenguaje el que se encarga de hacerlas automáticamente,

Conversión de tipos

- Las conversiones entre tipos primitivos se realizan **cuando en una expresión aparecen operandos de diferente tipo o bien asignamos a una variable un valor de distinto tipo al definido para la variable.**
- Por ejemplo, para poder sumar dos variables hace falta que ambas sean del mismo tipo. Si una es int y otra float, el valor de la primera se convierte a float antes de realizar la operación.

Conversión de tipos

- La sintaxis es:

```
(tipo) expresión
```

- Una vez evaluada la expresión, se convierte dicho valor resultante al tipo que hayamos señalado entre paréntesis.

Conversión implícita vs Conversión explícita

Conversión implícita

- Ocurre de forma automática.
- Siempre tiene éxito.
- Nunca se pierde información en la conversión.

Conversión explícita

- Requiere usar **cast** o **Convert.Toxxx(var)**
- Puede no tener éxito.
- Puede perderse información en la conversión.



Conversión implícita vs Conversión explícita

```
int intValue = 123;
long longValue = intValue; // implícita de int a long
intValue = (int) longValue; // explícita de long a int con cast
int x = 123456;
long y = x; // implícita
short z = (short)x; // explícita
double d = 1.2345678901234;
float f = (float)d; // explícita
long l = (long)d; // explícita
//uso de la clase Convert:
int c = 154;
string s = Convert.ToString(c);
```

Conversión implícita segura

- La conversión implícita es segura entre estos tipos:

Tipo	Se convierte de forma segura a:
Byte	short, ushort, int, uint, long, ulong, float, double, decimal
Sbyte	short, int, long, float, double, decimal
Short	int, long, float, double, decimal
Ushort	int, uint, long, ulong, float, double, decimal
Int	long, float, double, decimal
UInt	long, ulong, float, double, decimal
Long	float, double, decimal
Ulong	float, double, decimal
Float	double
Char	ushort, int, uint, long, ulong, float, double, decimal

Conversión utilizando Parsing

- Permite poder convertir la información contenida en un string a un tipo de dato.
- Esta acción es facilitada por el método Parse() que implementan los tipos numéricos y de fecha.

```
string numero= "1234";  
int i= int.Parse(numero);  
long l = long.Parse(numero);  
byte b = byte.Parse(numero); // OverflowException
```

Conversión utilizando Parsing

- C# dispone de un método para realizar la conversión realizando una comprobación utilizando un método booleano **.TryParse()**.
- Es el **MÁS RECOMENDADO**, pues controla las excepciones

```
double numero;  
string texto = "25.542";  
  
if (Double.TryParse(texto, out numero))  
{  
    Console.WriteLine("{0} --> {1}", texto, numero);  
}  
else  
{  
    Console.WriteLine("No se puede parsear '{0}'.", texto);  
}
```

Funciones librería Math

- La librería **System.Math** incluye una serie de funciones que nos serán muy útiles para simplificar los cálculos matemáticos:
 - **Math.sqrt**(numero) as **Double**: Calcula la raíz del número que se le pasa como parámetro y la devuelve como un Double.
 - **Math.Pow**(num1,num2) as Double: Calcula el primer numero elevado al segundo que se le pasan como parámetro. El dato que devuelve es de tipo Double.

Funciones librería Math

- **Math.round(num): Redondea al entero más cercano**
- **Math.Round**(num, digitos) as Double: Redondea el número decimal que recibe como parámetro
- **Math.Floor**(num) as Double: Redondea al número entero más bajo.
- **Math.Ceiling**(num) as Double: Redondea al número entero más alto.
- **Math.Min**(num1,num2)
- **Math.Max**(num1,num2)

Funciones librería Math

```
double raiz = Math.Sqrt(100);  
double potencia = Math.Pow(10, 2);  
double redondeo = Math.Round(3.54, 1);  
double suelo = Math.Floor(4.67);  
double techo = Math.Ceiling(4.67);  
double minimo = Math.Min(suelo, techo);  
double maximo = Math.Max(suelo, techo);
```

Actividad 2

Crea un programa donde ingreses 5 números por pantalla y muestres su promedio.

¿Debemos usar conversión implícita o explícita?

Controla qué pasaría si introducimos una letra.

5 MINUTOS

Operadores.

- Son símbolos que indican cómo son manipulados los datos sobre los que actúan (operandos)
- Por ejemplo en la expresión:

```
int resultado = 10 + 50;
```

- Aparecen dos operadores:
 - **+**: Es un operador binario ya que trabaja con dos operandos, se encarga de sumar el número que aparece a su izquierda y el que aparece a su derecha, y devolver el resultado.
 - **=**: Es un operador binario que en este caso recoge el resultado de la expresión que aparece a su derecha y le asigna ese valor a la variable que aparece a su izquierda.

Operadores.

- Algunos operadores tienen funcionalidades diferentes dependiendo de dónde estén situados (están sobrecargados)
- Los operadores se pueden clasificar en:
 - Aritméticos
 - Lógicos
 - Relacionales
 - De asignación

Estructuras. Operadores aritméticos

- Devuelven un número como fruto de la operación que representan. Los operandos deben representar valores numéricos.

Operador	Función	Ejemplo
<code>+, -, *, /</code>	Suma, resta, producto y división real.	<code>int resultado = 25 + 10;</code>
<code>\</code>	División entera.	<code>float resto = 25 / 10;</code>
<code>%</code>	Resto de la división entera.	<code>int resto = 25 % 10;</code>
<code>^</code>	Exponenciación.	<code>int resultado = 5 ^ 10;</code>

Estructuras. Operadores de asignación

- Estos operadores no devuelven ningún dato, simplemente asignan al operando que está a la izquierda el valor del operando de la derecha.

Operador	Función	Ejemplo
=	Asignación simple	<code>int resultado = 25;</code>
+=	Suma más asignación	<code>int resultado += 25;</code>
-=	Resta más asignación	<code>int resultado -= 25;</code>
*=	Multip más asignación	<code>int resultado *= 25;</code>
/=	División más asignación	<code>int resultado /= 25;</code>

Estructuras. Operadores relacionales

- Estos operadores devuelven un dato de tipo bool es decir true o false, este resultado también lo podremos guardar en una variable de tipo bool:

Operadores	Función	Ejemplo
<	Menor que	resultado = 18 < 30;
>	Mayor que	resultado = 19 > 30;
<=	Menor o igual	resultado = 25 <= 25;
>=	Mayor o igual	resultado = 26 >= 44;
==	Igual que	resultado = 66 == 66;
!=	Distinto que	resultado = "Hola" != "Hola";

Estructuras. Operadores lógicos

- Devuelven un número como fruto de la operación que representan. Los operandos deben representar valores numéricos.

Operador.	Función.	Ejemplo.
&&	Y Lógico	resultado = alturaRamon > 180 && edadRamon > 62;
 	O Lógico	resultado = alturaRamon > 180 edadRamon > 62;
!	Negación	resultado = !(cocheRamon == "Maserati");

Actividad 3

Crea un programa que lea dos números ingresados por el usuario y calcule la suma, la resta, el producto y la división de estos dos números. Luego, muestra el resultado en la consola.

5 MINUTOS

Sentencias de selección simple.

- Hasta ahora todos nuestros programas tenían un flujo lineal de ejecución.
- En ocasiones el programa debe tomar decisiones y evaluando algunas expresiones booleanas decidir que sentencias debe ejecutar.
- Para indicar al programa que debe decidir entre dos o más alternativas utilizaremos la sentencia de selección ***If - Else.***

```
if (quieroAprobar)
{
    Console.WriteLine("Don Luis es más guapo que Don Iván");
}
else
{
    Console.WriteLine("El senderismo es la mejor actividad deportiva");
}
```

- Si **condicion** es **True** (verdadera), se ejecutan las sentencias que están a continuación de, y si condicion es False (falsa), se ejecutan las sentencias que están a continuación de Else, si esta cláusula ha sido especificada (pues es opcional).

Ejemplo

```
//Programa que calcula la nota media y si supera el cinco indica que está aprobado

float nota1, nota2, nota3;
float media;
bool aprobado;

//Solicitamos las notas
Console.WriteLine("Introduzca la nota 1: ");
nota1 = Convert.ToSingle(Console.ReadLine());

Console.WriteLine("Introduzca la nota 2: ");

nota2 = Convert.ToSingle(Console.ReadLine());

Console.WriteLine("Introduzca la nota 3: ");

nota3 = Convert.ToSingle(Console.ReadLine());

//Calculamos la media de las 3 notas

media = (nota1 + nota2 + nota3) / 3;
```

Ejemplo

```
//Establecer variable booleana aprobado

aprobado = media >= 5;

//Evaluamos variable booleana

if (aprobado)
{
    Console.WriteLine("Alumno aprobado");
}
else
{
    Console.WriteLine("Alumno suspenso.");
}
```

Ejemplo

```
//Este programa calcula la raíz de un número si es positivo
int numero;
double raiz;
bool positivo = false;

//Leemos el número por teclado
Console.WriteLine("Introduzca un número: ");
numero = Convert.ToInt32(Console.ReadLine());

//Establecemos variable booleana

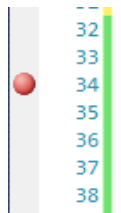
positivo = numero >= 0;

//Evaluamos variable booleana
if (positivo)
{
    raiz = Math.Sqrt(numero);
    Console.WriteLine("La raíz es: " + raiz);
    Console.WriteLine("Número negativo, no tiene raíz");
}

Console.ReadLine();
```

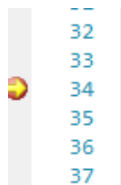
Puntos de interrupción

- Cuando los programas se complican puede que se produzcan salidas inesperados y no seamos capaces de entender porqué se producen.
- Para facilitar esta tarea el programador puede establecer puntos de interrupción en el código, de esta manera la depuración se detendrá en dicha línea del código y el programador podrá ejecutarla paso por paso e ir comprobando como toman valores las diferentes variables o porqué rutas avanza la depuración del programa.
- Para establecer un punto de interrupción simplemente lo marcamos a la izquierda del código:



```
--  
32  
33  
34  
35  
36  
37  
38  
  
Dim i As Integer = 5  
  
If i > 0 And i < 5 Then  
    i = 0  
Else  
    i += 5  
End If
```

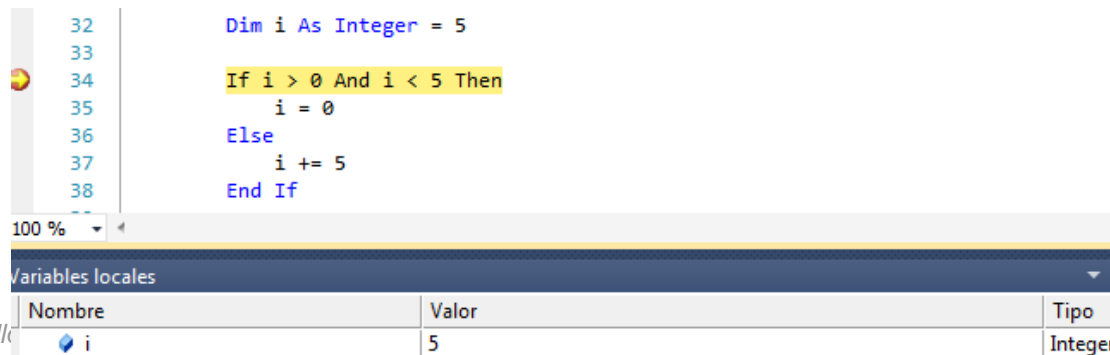
- A continuación iniciamos la depuración y esta se detendrá en dicho punto



```
--  
32  
33  
34  
35  
36  
37  
  
Dim i As Integer = 5  
  
If i > 0 And i < 5 Then  
    i = 0  
Else  
    i += 5
```

Puntos de interrupción

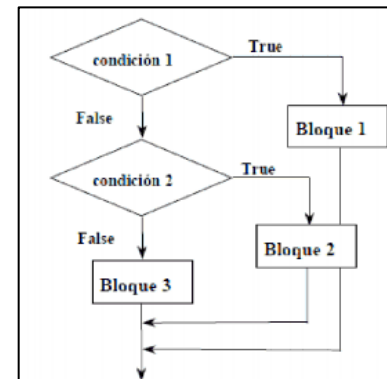
- Una vez ahí podremos avanzar línea por línea pulsando la tecla **F10** e incluso observar los valores que van tomando las variables.
- Para ver los valores de las variables pondremos el cursor encima de la variable en cuestión, también tendremos una lista de las variables y sus valores en la ventana de *Variables locales* que se nos mostrará en la parte inferior del Visual Studio.



If - Else IF

- Cuando tengamos más de dos condiciones excluyentes deberemos de expresarlas con la sentencia **Else IF** que se puede interpretar como *“Si no se cumplió la condición anterior pero se cumple esta segunda condición..”*

```
if (condicion1)
{
    Console.WriteLine("Se cumple la condición 1");
}
else if (condicion2)
{
    Console.WriteLine("Se cumple la condición 2");
}
else
{
    Console.WriteLine("No se cumple ninguna condición");
}
```



- Si se cumple la **condicion1** se ejecutan las **sentencias1**, y si no se cumple, se examinan secuencialmente las condiciones siguientes hasta Else, ejecutándose las sentencias correspondientes al primer Elself cuya condición se cumpla. Si todas las condiciones son falsas, se ejecutan las sentencias-n correspondientes a Else, que es la opción por defecto.

Ejemplo

- Ejemplo:
- Programa que clasifica un número como positivo, negativo o cero.

```
//Numero positivo, negativo o cero

int num;
Console.Write("Mete un número: ");
num = int.Parse(Console.ReadLine());

//Realizamos los cálculos

if (num > 0)
{
    Console.WriteLine("Número positivo");
}
else if (num == 0)
{
    Console.WriteLine("Número cero");
}
else
{
    Console.WriteLine("Número negativo");
}

Console.ReadLine();
```

Ejemplo

- Programa que comprueba si un número es **mayor que 100** y **múltiplo de 6** o alguna de las dos opciones.

```
int numero;  
Console.WriteLine("Introduzca un número: ");  
numero = int.Parse(Console.ReadLine());  
  
if (numero > 100 && (numero % 6) == 0)  
{  
    Console.WriteLine("Múltiplo de 6 y mayor que 100");  
}  
else if (numero > 100 && (numero % 6) != 0)  
{  
    Console.WriteLine("Mayor que cien y no es múltiplo de 6");  
}  
else if (numero < 100 && (numero % 6) == 0)  
{  
    Console.WriteLine("Número múltiplo de 6 pero menor que cien");  
}  
else  
{  
    Console.WriteLine("Número menor que 100 y no es múltiplo de 6.");  
}
```

Actividad 4

Crea un programa que lea un número ingresado por el usuario y determine si es un número par o impar. Luego, muestra el resultado en la consola.

5 MINUTOS

Sentencias de selección anidadas.

- Es posible introducir sentencias de selección dentro de otras sentencias de selección, pero debemos tener cuidado de ir cerrando correctamente todas las rutas abiertas. Por ejemplo podemos tener un if dentro de otro if:

```
if (condicion1)
{
    if (condicion2)
    {
        Console.WriteLine("Se cumplen las condiciones 1 y 2");
    }
    else
    {
        Console.WriteLine("Se cumple la condición 1 pero no la 2");
    }
}
else
{
    Console.WriteLine("No se cumple ninguna de las condiciones");
}
```

- Si las condiciones que se expresan con un And implican diferentes acciones podemos separarlas y resolver el problema con un If dentro de otro.

Ejemplo

```
int x = 5;

if (x % 6 == 0)
{
    if (x > 100)
    {
        //Si entra aqui es que el numero es multiplo de 6 y mayor que 100
        Console.WriteLine("Múltiplo de 6 y mayor que 100");
    }
    else
    {
        //Si entra aqui es que el numero es multiplo de 6 pero menor que 100
        Console.WriteLine("Número múltiplo de 6 pero menor que cien");
    }
}
else
{
    //Entra aquí si no es multiplo de 6
    if (x > 100)
    {
        //No multiplo pero mayor que 100
        Console.WriteLine("Mayor que cien y no es múltiplo de 6");
    }
    else
    {
        Console.WriteLine("Número menor que 100 y no es múltiplo de 6.");
    }
}
```

Actividad 5

Crea un programa que lea una calificación ingresada por el usuario (un número entre 0 y 100) y determine su equivalente en letras, según la siguiente tabla:

- *Si la calificación es mayor o igual a 90, se asigna la letra "A".*
- *Si la calificación es mayor o igual a 80 y menor a 90, se asigna la letra "B".*
- *Si la calificación es mayor o igual a 70 y menor a 80, se asigna la letra "C".*
- *Si la calificación es mayor o igual a 60 y menor a 70, se asigna la letra "D".*
- *Si la calificación es menor a 60, se asigna la letra "F".*

10 MINUTOS

Estructuras. Switch

- Se quiere comparar la misma variable con muchos valores diferentes, y ejecutar una parte de código distinta dependiendo de a qué valor es igual.
- Parecido al Elself

```
int edad = 20;  
  
switch (edad)  
{  
    case >= 18:  
    {  
        Console.WriteLine("Mayor de edad");  
        break;  
    }  
    case < 18:  
    {  
        Console.WriteLine("Menor de edad");  
        break;  
    }  
}
```


Estructuras. Switch

- Solo cuando se encuentra una sentencia **case** con un valor que coincide con el valor de la expresión switch se comienza a ejecutar las sentencias.
- C# continúa ejecutando las sentencias hasta el final del bloque switch, o la primera vez que vea una sentencia **break**.
- Si no se escribe una sentencia **break** al final de una lista de sentencias **case**, C# seguirá ejecutando las sentencias del siguiente case

Actividad 5B

Utilizando la estructura switch, crea un programa que lea un número del 1 al 7 ingresado por el usuario y determine el día de la semana correspondiente.

Luego, muestra el día de la semana en la consola.

5 MINUTOS



**KEEP
CALM
IT'S
KAHOOT
TIME**

Concepto de bucle.

- Algunos algoritmos implican la repetición de una serie de acciones mientras se cumplan unas determinadas condiciones.
- Por ejemplo, si queremos ver todos los nombres de los alumnos que tenemos guardado en un determinado fichero o base de datos, debemos indicar que “*mientras haya registros, continúa leyendo*”
- En este caso la condición para que el algoritmo de lectura se siga ejecutando es que haya registros.
- Otro ejemplo, si queremos mostrar por pantalla todos los números inferiores a 50, deberemos indicar: “*mientras número sea inferior a 50, muestra por pantalla el número.*”
- En el ejemplo anterior también necesitamos una variable que vaya tomando todos los valores desde 1 hasta 50, esta variable se conoce como **variable de control** y suele estar implicada en la **condición de continuidad**.

Partes de un bucle.

- **Variable de control:** Es la variable que utilizaremos para indicar el número de veces que se repetirá el bucle. La nombraremos siempre con la letra i.
- **Inicialización de la variable de control:** Para controlar las iteraciones deberemos indicar el primer valor que debe tomar la variable de control.
- **Condición de control o continuidad:** Es una expresión booleana que indicará cuando se debe salir del bucle.
- **Incremento:** Deberemos indicar como va variando la variable de control desde el valor inicial hasta el valor final expresado en la condición de control. Sino lo hacemos probablemente el bucle sea un bucle infinito.
- **Instrucciones:** Son las líneas de código que se ejecutan con cada iteración.

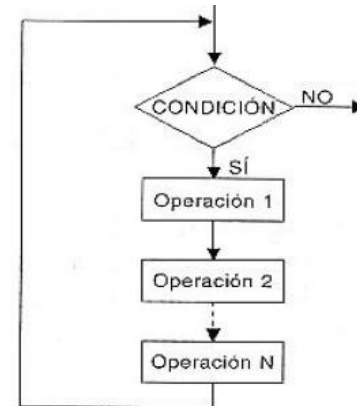
Tipos de bucle.

- Existen dos tipos de bucles que básicamente son equivalentes, el bucle **For** (para) y el bucle **While**(mientras)
- El pseudocódigo para estas instrucciones es el siguiente:

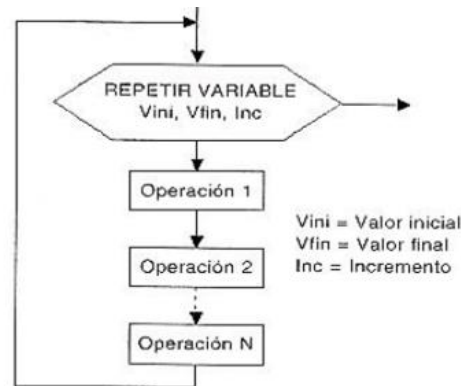
mientras condición hacer
instrucciones
fin mientras

para $i \leftarrow x$ hasta n a incrementos de s hacer
instrucciones
fin para

- Para representar el bucle while utilizaremos el siguiente esquema , formado por una condición y unas instrucciones que se repitan :



- Para representar el bucle for:



Bucle While.

El bucle while (y todos los bucles en general)

tiene tres partes bien diferenciadas e imprescindibles:

- Inicialización de la variable de control.
- Condición de continuidad.
- Incremento o decremento de la variable de control.

Si no implementamos este paso corremos el

riesgo de generar **bucles infinitos** que cuelguen la aplicación o generen una excepción del tipo **overflow**.

```
//declaramos la variable de control
int i;

//inicializamos la variable de control
i = 1;

//expresamos la condición
while (i <= 100)
{
    Console.WriteLine(i + " ");

    //indicamos como variar la variable de control
    i = i + 1;
}
Console.ReadLine();
```

Bucle While. Ejemplo

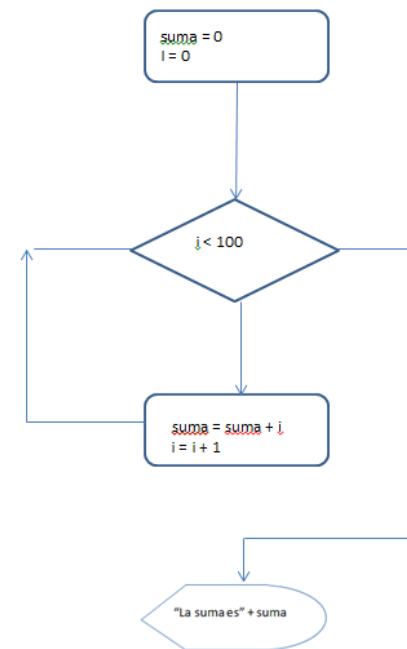
- Calcular la suma de los 100 primeros números: $1 + 2 + 3 + \dots + 100$

```
//inicializamos la variable de control
i = 1;

//inicializamos la variable suma
suma = 0;

//expresamos la condición
while (i <= 100)
{
    //guardamos en la variable suma los diferentes resultados
    suma = suma + i;

    //indicamos como variar la variable de control
    i = i + 1;
}
Console.WriteLine("La suma es: " + suma);
Console.ReadLine();
```



Actividad 7A

Usando el bucle **while**, crea un programa que le pida al usuario ingresar un número y luego calcule la suma de los números desde 1 hasta el número ingresado.

El programa debe repetir esta acción hasta que el usuario ingrese un 0.

10 MINUTOS

Bucle For.

- Repite sentencias un número de veces determinado.

Su sintaxis es la siguiente :

```
for (inicializacion; condicion iteración; incremento)
{
    sentencias código
}
```

- El bucle for contiene las 4 partes siguientes:
 - **La parte de inicialización:** inicializa las variables de control del bucle.
 - **La parte de condición de iteración** contiene una expresión lógica que hace que el bucle realice las iteraciones de las sentencias.
 - **La parte de incremento** (o decremento) modifica la variable o variables de control del bucle.
 - **Las sentencias**, acciones que se ejecutarán por cada iteración del bucle

```
//declaramos la variable suma
int suma = 0;

for (int i = 0; i < 100; i++)
{
    suma = suma + i;
}

Console.WriteLine("La suma es: " + suma);

Console.ReadLine();
```

Actividad 7B

Usando el bucle **for**, crea un programa que calcule la suma de los números impares desde 1 hasta un número ingresado por el usuario.

Luego, muestra la suma en la consola.

5 MINUTOS

Bucles infinitos

- Cuando la condición de continuidad siempre es verdadera, el programa se verá abocado a repetir las instrucciones que hay dentro del bucle sin parar, esto dará lugar a un bucle infinito que acabará por cerrar la aplicación.
- Un bucle infinito puede darse por varias situaciones:
 - No modificar la variable de control.
 - Expresar mal la condición de control o continuidad

```
//declaramos la variable de control
int i = 0;

//inicializamos la variable suma
int suma = 0;

//expresamos la condición
while (i <= 100)
{
    //guardamos en la variable suma los diferentes resultados
    suma = suma + i;
}

while (true)
{
    Console.WriteLine("w ");
}
```

Salir de un bucle.

- Cuando nos interese forzar la salida de un bucle aun cuando la condición de control siga siendo cierta, utilizaremos la palabra **break** para indicarlo:

```
for (int i = 3; i < 100; i++)  
{  
    if (i % 3 == 0 && i % 4 == 0 && i % 5 == 0)  
    {  
        Console.WriteLine(i + " es el primer múltiplo de 3,4 y 5");  
        break;  
    }  
}
```

Continuar un bucle.

- Cuando nos interese saltar a la siguiente iteración del bucle, utilizaremos la palabra `continue` para indicarlo:

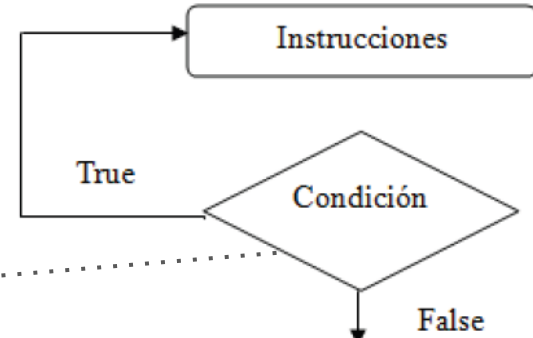
```
for (int i = 1; i < 10; i++)  
{  
    if (i == 5)  
    {  
        continue;  
    }  
}
```

Bucle do - while.

- La principal diferencia de este bucle con los anteriores es que al menos las instrucciones se ejecutarán una vez.
- Las continuará ejecutando mientras se cumpla una determinada condición.
- Se suele utilizar para procesar los datos de entrada y solicitarlos otra vez en caso de que no tengan el formato correcto.

```
//Definimos las variables
char car;

do
{
    Console.WriteLine("¿Desea continuar s/n (sí o no)");
    car = Convert.ToChar(Console.Read());
    Console.ReadLine();
} while (car != 's' && car != 'n');
```



Actividad 7C

Usando el bucle **do-while**, crea un programa que le pida al usuario ingresar un número y luego calcule el factorial del número ingresado.

El programa debe ejecutarse **AL MENOS UNA VEZ** y debe repetir esta acción hasta que el usuario ingrese un número negativo.

10 MINUTOS

Anidación de bucles

- Los bucles al igual que las instrucciones if else, se pueden anidar uno dentro de otro.
- Generalmente utilizaremos dos niveles de anidación aunque pueden ser muchos más. La única condición es que el bucle interno debe estar totalmente contenido dentro del externo.
- Este tipo de algoritmos representa un bucle externo cuya misión es hacer que se repita n veces un algoritmo interno que contiene a su vez un bucle.
- Por ejemplo: Calcular todos los números primos por debajo de uno dado.

```
for (int i = 0; i < 10; i++)  
{  
    for (int j = 0; j < 10; j++)  
    {  
        Console.Write(j + " ");  
    }  
    Console.WriteLine();  
}  
Console.ReadLine();
```

```
int n = 5;  
  
for (int i = 0; i < n; i++)  
{  
    for (int j = 0; j < i; j++)  
    {  
        Console.Write(j + " ");  
    }  
    Console.WriteLine();  
}
```

Anidación de bucles. Ejemplo

- Programa que solicita las notas de n alumnos y calcula sus medias:

```
//Repetición de una media de una lista de números:

int notas, cont, nlistas, contlista;
float x, suma, media;
Console.WriteLine("¿De cuantos alumnos va a introducir las notas?");
nlistas = Convert.ToInt32(Console.ReadLine());

//bucle externo para procesar las listas de números

for (contlista = 1; contlista <= nlistas; contlista++)
{
    suma = 0;

    Console.WriteLine("Cuantos notas tiene el alumno: ");

    notas = Convert.ToInt32(Console.ReadLine());

    for (cont = 1; cont <= notas; cont++)
    {
        Console.WriteLine("Nota : " + cont);
        x = Convert.ToInt32(Console.ReadLine());
        suma += x;
    }
    media = suma / notas;

    Console.WriteLine("La media es: " + media);

} //Fin del bucle externo
```

Actividad 8A

Crea un programa que imprima los números del 1 al 10 en forma de pirámide

Debemos usar la instrucción `Console.Write()` y `Console.WriteLine()`

La salida debería ser la siguiente:

```
1
12
123
1234
12345
123456
1234567
12345678
123456789
1234567890
```

5 MINUTOS

Actividad 8B

Crea un programa que pida al usuario ingresar una tabla de multiplicación y luego muestre la tabla completa.

Por ejemplo, si el usuario ingresa 5, la salida debería ser la siguiente:

```
5 x 1 = 5  
5 x 2 = 10  
5 x 3 = 15  
5 x 4 = 20  
5 x 5 = 25  
5 x 6 = 30  
5 x 7 = 35  
5 x 8 = 40  
5 x 9 = 45  
5 x 10 = 50
```

5 MINUTOS



**KEEP
CALM
IT'S
KAHOOT
TIME**

Clase Random

- La clase Random del espacio de nombres System **define un conjunto de operaciones para la generación de números pseudoaleatorios:** (números que cumplen determinados requisitos estadísticos de aleatoriedad).
- Para ello se debe crear un objeto de la clase Random y posteriormente invocar a ciertos métodos para obtener los números pseudoaleatorios.

Clase Random

- Los constructores:
 - **public Random()**. Inicializa una nueva instancia de la clase Random mediante un valor de inicialización predeterminado que depende del tiempo.
 - **public Random(int semilla)**. Inicializa una nueva instancia de la clase Random() utilizando el valor de inicialización especificado.

Clase Random

- Los métodos:
 - **int Next();** devuelve un entero positivo en el rango [0, MaxValue]
 - **int Next(int max);** devuelve un entero positivo en el rango [0, max-1]
 - **int Next(int min, int max);** devuelve un entero positivo en el rango [min, max-1]
 - **double NextDouble();** devuelve un real en el rango [0,1).

Clase Random. Ejemplo

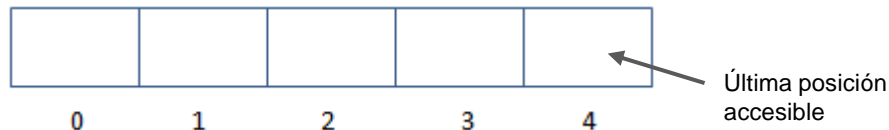
- Generar una clave aleatoria constituida por letras mayúsculas y con una longitud entre 6 y 8 letras:

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

namespace Aleatorio
{
    class Program
    {
        static void Main(string[] args)
        {
            Random rnd = new Random();
            string clave = "";
            int long_clave = rnd.Next(6, 9);
            for (int i = 1; i <= long_clave; i++)
            {
                clave = clave + (char)rnd.Next('A', 'Z' + 1);
            }
            Console.WriteLine(clave);
        }
    }
}
```

Concepto de matriz o array.

- Un array es una variable en la que podremos guardar muchos datos, todos del mismo tipo.
- A cada elemento de un array se le asigna un índice.
- A los diferentes valores que guarda un array podremos acceder a través del **nombre** de la variable y del **índice** que tenga asignado.
- El primer elemento ocupa la **posición 0** , el segundo la posición 1 y así sucesivamente.
- Por ejemplo un array con capacidad para 5 valores lo podríamos imaginar así:



Declaración de una variable de tipo Array.



FORMACIÓN
PROFESIONAL
MONTECASTELO

- La sintaxis general es la siguiente:

```
Tipo[] nombre = new Tipo[5];
```

- Por ejemplo si queremos declarar un array de 5 enteros :

```
int[] matriz1 = new int[5];
```

- Los array de tipo numérico almacenarán inicialmente en todas sus posiciones el valor 0, los de tipo String el valor *null*:

0	0	0	0	0
0	1	2	3	4

NULL	NULL	NULL	NULL	NULL
0	1	2	3	4

- Para acceder a una de las posiciones deberemos indicar nombre del array y el índice que queremos modificar o leer:

$x[0] = 5$

$x[1] = 3$

Inicialización de los valores del Array

- Si queremos darle valores iniciales a los elementos del Array deberemos cambiar ligeramente la sintaxis:

```
Tipo[] nombre = new Tipo[]{ 1, 3, 5, 7, 9 };
```

- Por ejemplo:

```
int[] array = new int[]{ 1, 3, 5, 7, 9 };
```

3	5	7	0	2
0	1	2	3	4

Ejemplo 6.1: Recorrer un array con un For y solicitar algún dato por teclado.

```
//Declaramos un Array de Integers
int[] registros = new int[9];

//Inicializamos los 3 primeros valores
registros[0] = 5;
registros[1] = 9;
registros[2] = 4;

//Solicitamos al usuario que introduzca valores para posiciones 3 y 4
Console.WriteLine("Introduzca el valor de la posición 3");
registros[3] = int.Parse(Console.ReadLine());
Console.WriteLine("Introduzca el valor de la posición 4");
registros[4] = int.Parse(Console.ReadLine());

//Mostramos todos los valores guardados con un for
for (int i = 0; i < registros.Length; i++)
{
    Console.Write(registros[i] + " ");
}
Console.ReadLine();
```

La i va tomando todos los valores posibles de las posiciones.

Propiedad length de las matrices

- Todas las variables de tipo array tienen la propiedad length que nos devuelve el tamaño de dicho array.
- Para llamarla deberemos poner el nombre del array y el operador punto:

```
int capacidad = registros.Length;
```

- Esta propiedad nos será muy útil para recorrer los arrays independientemente del tamaño que tengan:

```
for (int i = 0; i < registros.Length; i++) { }
```

- La ventaja es que si el array cambia de tamaño el bucle sigue recorriéndolo hasta su última posición sea cuál sea.

Ejemplo 6_2:

- En este programa el usuario escogerá el tamaño de la matriz, a continuación introducirá los datos, el programa muestra el contenido:

```
int n;  
Console.WriteLine("Introduzca el número de elementos: ");  
n = int.Parse(Console.ReadLine());  
int[] arrayTemporal = new int[n];  
  
//Guardamos los datos  
for (int i = 0; i < arrayTemporal.Length; i++)  
{  
    Console.WriteLine("Introduzca el valor del índice: " + i);  
    arrayTemporal[i] = int.Parse(Console.ReadLine());  
}  
  
//Una vez guardados los datos los mostramos por teclado  
for (int i = 0; i < arrayTemporal.Length; i++)  
{  
    Console.WriteLine(arrayTemporal[i]);  
}  
Console.ReadLine();
```

Comparación de arrays.

- En C# no se permiten operaciones que involucren arrays completos.
- Así, si a y b son arrays similares (mismo tamaño y tipo de datos), operaciones de asignación, comparación ... **deben realizarse elemento a elemento.**
- Si no se hace así, puede dar lugar a errores de compilación o en algunos casos más graves a un funcionamiento no deseado del algoritmo
- Si queremos comprobar si dos arrays contienen los mismos elementos debemos de ir comparando posición por posición.

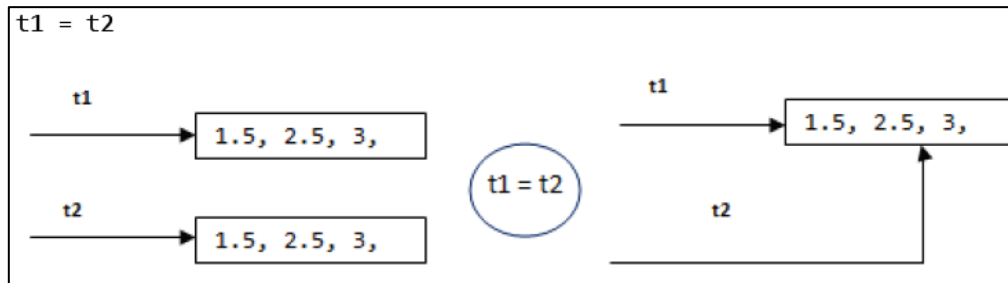
Ejemplo 6.3 : Comparación de Arrays.

```
float[] array1 = new float[] { 3.5f, 6.7f, 8.0f };
float[] array2 = new float[] { 3.5f, 6.7f, 8.0f };
int elementosDistintos = 0;

//Comparamos elemento a elemento
for (int i = 0; i < array1.Length; i++)
{
    if (array1[i] != array2[i])
    {
        elementosDistintos++;
        //Salimos del bucle en cuanto haya un elemento distinto
        break;
    }
}
if (elementosDistintos == 0)
{
    Console.WriteLine("Arrays iguales");
}
else
{
    Console.WriteLine("Arrays distintos");
}
Console.ReadLine();
```

Igualar dos Arrays

- Si tenemos dos arrays y queremos que uno de ellos almacene los mismos valores que el otro, no podremos hacerlo utilizando el operador = directamente, ya que esto lo que haría sería eliminar uno de los arrays:



- Lo que sucede es que las dos *referencias* manipulan un único array , el otro que se queda sin referenciar es eliminado de la memoria.
- Por lo tanto habría que realizarlo elemento a elemento.**

Actividad 9

Crea un array **notasAlumnos** con las notas del último examen:

{5, 8, 9, 4.6, 7, 6, 2.9, 10}

Crea otro array **copiaNotas** y cópialo.

Compara los dos arrays y di si son iguales o no.

Por último, muestra los dos por pantalla

5 MINUTOS

La clase Array.

- En programación, **una clase** es un componente que ya está diseñada y programada que se puede utilizar en diferentes proyectos.

- Visual contiene muchísimas clases ya diseñadas:

Console, Math, Integer, String ...

- La clase Array contiene una serie de métodos que facilita el trabajar con arrays:

Sort, BinarySearch, Reverse, Copy, Clear, Equals, IndexOf

- Para llamar a estas funciones utilizaremos la siguiente sintaxis:

Array.nombreFunción(...)

Método Sort y Reverse.

- Esta función ordena el Array según el orden natural de sus elementos.
- Esto quiere decir que si el Array está compuesto por números, los ordenará de menor a mayor, si el array está compuesto por strings , los ordenará alfabéticamente:

```
int[] matriz = new int[] { 7, 6, 5, 4, 3, 2, 1 };  
  
//Ordena en orden normal  
Array.Sort(matriz);  
foreach (var elemento in matriz)  
{  
    Console.Write(elemento + " ");  
}  
Console.WriteLine();  
//Ordena en orden contrario  
Array.Reverse(matriz);  
foreach (var elemento in matriz)  
{  
    Console.Write(elemento + " ");  
}
```

Actividad 10

Crea un programa en C# que permita al usuario ingresar una cantidad variable de números enteros, los guarde en una lista, ordene los elementos de la lista de menor a mayor con el método Sort y finalmente muestre la lista de números en orden inverso con el método Reverse.

5 MINUTOS

Método BinarySearch

- Busca la primera aparición de un valor que recibe como parámetro junto con la matriz en la que queremos realizar la búsqueda:
- El array debe de estar previamente ordenado.
- En caso de que no encuentre el valor buscado devuelve un número negativo.

```
//Búsqueda de la posición del número 6
Array.Sort(matriz);
int posicion = Array.BinarySearch(matriz, 6);

if (posicion < 0)
{
    Console.WriteLine("No se ha encontrado el elemento en la matriz");
}
else
{
    Console.WriteLine("Está en la posición : " + posicion);
}
```

Actividad 11

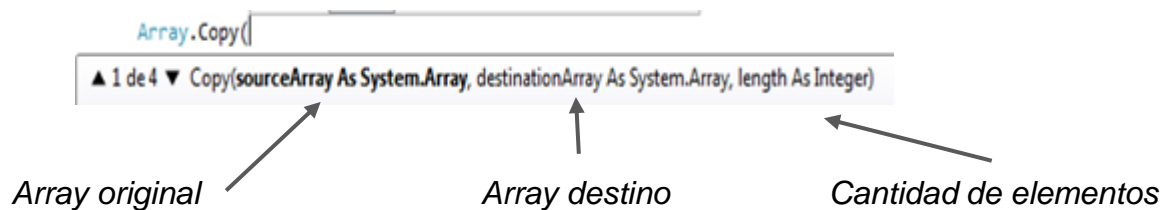
Crea un programa en C# que permita al usuario ingresar una cantidad variable de números enteros, los guarde en una lista, ordene los elementos de la lista de menor a mayor con el método **Sort**, luego pida al usuario que ingrese un número a buscar en la lista y utilice el método **BinarySearch** para determinar si el número se encuentra en la lista y en qué posición.

Finalmente, muestre un mensaje indicando si el número se encontró y en qué posición, o un mensaje de error si no se encontró.

10 MINUTOS

Método Copy

- Copia los elementos de un array en otro array que también es pasado como parámetro.



- Ejemplo:

```
int[] original = new int[] { 3, 0, 9, 8, 4 };  
int[] copia = new int[original.Length];  
  
Array.Copy(original, copia, original.Length);
```

Actividad 12

Crea un programa en C# que permita al usuario ingresar una cantidad variable de números enteros, los guarde en una lista, copie los elementos de la lista a otra lista utilizando el método Copy y finalmente muestre ambas listas, la original y la copia, en pantalla.

10 MINUTOS

Método IndexOf y LastIndexOf

- Devuelve el índice de la primera aparición del elemento buscado.

```
int posicion = Array.IndexOf(matriz, 6);  
  
if (posicion < 0)  
{  
    Console.WriteLine("No se ha encontrado el elemento en la matriz");  
}  
else  
{  
    Console.WriteLine("Está en la posición : " + posicion2);  
}
```

Sentencia Foreach

- Para recorrer un array de forma segura sin preocuparnos por el tamaño del array podemos utilizar un bucle parecido al For.

- Su sintaxis es la siguiente:

```
foreach (int elemento in matriz) { }
```

- Ejemplo :

```
int[] numeros = new int[7] { 7, 6, 5, 4, 3, 2, 1 };
string[] nombres = new string[] { "Victoria", "Alba", "Elena" };

//Recorrer una matriz de numeros
foreach (int numero in numeros)
{
    if (numero < 0)
    {
        Console.WriteLine("Numero negativo " + numero);
    }
}

//Recorrer una matriz de strings
foreach (string elemento in nombres)
{
    Console.WriteLine(elemento);
}
```

Actividad 13

Crea un programa en C# que permita al usuario ingresar una cantidad variable de nombres, los guarde en una lista y los muestre en pantalla utilizando un bucle **foreach**.

10 MINUTOS

Operador Is.

- Cuando trabajamos con **matrices de Strings** es habitual querer comprobar si alguna posición de la matriz está vacía, para poder realizar esta comprobación podemos utilizar el operador **=**.
- Sin embargo existe también otro operador que nos puede resultar útil para realizar esta comprobación, se trata del operador **Is**.
- Aunque con Strings es indiferente usar uno u otro operador , es conveniente familiarizarse con el operador **Is** ya que con otros tipos de variables será el único que podremos utilizar.

Ejemplo 6.4

```
string[] nombres = new string[4];
nombres[0] = "Laura";
nombres[2] = "Juan";
nombres[3] = "Pablo";

//Algoritmo que muestra las posiciones vacías de la matriz
for (int i = 0; i < nombres.Length; i++)
{
    if (nombres[i] is null)
    {
        Console.WriteLine(i + " Posición vacía ");
    }
}

//Algoritmo que muestra los nombres guardados en la matriz
foreach (string item in nombres)
{
    if (item is not null)
    {
        Console.WriteLine(item);
    }
}
```

Actividad 14

Crea un programa en C# que rellene de forma aleatoria con numeros aleatorios entre 1 y 10 un array de **n** elementos.

El tamaño del array se lo debemos pasar por pantalla.

Comprueba en qué posiciones el array tiene elementos y en cual es nulo

10 MINUTOS

Redimensionar el tamaño de una matriz

- Cuando una matriz se queda sin espacio para almacenar más datos podemos aumentar su capacidad.
- En C#, la función **Array.Resize** nos permite esta operación:

```
// Redimensionamos el array a un tamaño mayor
Array.Resize(ref nombres, nombres.Length + 5);
for (int i = 0; i < nombres.Length; i++)
{
    Console.WriteLine(nombres[i]);
}

// Redimensionamos el array a un tamaño menor
Array.Resize(ref nombres, 2);
for (int i = 0; i < nombres.Length; i++)
{
    Console.WriteLine(nombres[i]);
}
```

Actividad 15

Crea un programa en C# que realice lo siguiente:

1. Define un array de 10 números enteros.
2. Rellena el array con números aleatorios entre 1 y 100.
3. Imprime en la consola cada número en el array utilizando un bucle foreach.
4. Calcula y muestra en la consola la suma de todos los números en el array.
5. Determina y muestra en la consola el número más grande y el más pequeño en el array

5 MINUTOS



**KEEP
CALM
IT'S
KAHOOT
TIME**

Definición de métodos

- Hasta ahora hemos utilizado métodos que ya estaban definidos, como `WriteLine` o `Sqrt`.
- Pero, ¿y si queremos implementar nuestros propios métodos?
- A continuación, veremos la definición de métodos.

Definición de métodos

- La definición de un método consiste en indicar al compilador:
 - **Nombre del método**
 - **Número y tipo de parámetros**, si es que los necesita
 - **Tipo del valor devuelto**, si es que devuelve algo.
- A continuación, se escribe el cuerpo del método con las sentencias que deben ejecutarse cuando se invoque al método

Definición de métodos

- La sintaxis sería:

```
tipo_retorno nombre_metodo([lista parámetros])  
{  
    declaración de variables locales  
    sentencias;  
    [return [expresión]];  
}
```

Definición de métodos

- **El tipo de retorno** sirve para **indicar el tipo del resultado que va a devolver el método. Si no va a devolver nada, se usa la palabra reservada void.**
- El nombre del método es el identificador que usaremos para invocarlo.

Definición de métodos

- La lista de parámetros es una **lista de variables con sus tipos separadas por comas**. Los parámetros se usan para pasar datos al método con los que trabajar. Si no hay que enviar valores, no habrá lista de parámetros, aunque los paréntesis deben mantenerse.
- Los parámetros son variables locales al método, por lo que sólo serán accesibles dentro del método.

Definición de métodos

- **La sentencia return es opcional:**
 - **Si existe**, provoca que se termine la ejecución del método y el control es devuelto al método que hizo la llamada, aunque queden sentencias por ejecutar en el método invocado.
 - **Si no existe** la sentencia return, el control vuelve al método que hizo la llamada cuando se acaben de ejecutar todas las sentencias del método invocado.
 - Pueden existir varias sentencias return, aunque solo se ejecutará una.

Actividad 16

Con lo que hemos visto hasta ahora, desarrolla un método que reciba un entero, calcule su cuadrado y lo muestre por pantalla de la siguiente forma:

- *El número recibido ha sido: [numero]*
- *El cuadrado de [numero] es $[numero]^2$*

5 MINUTOS

Definición de métodos

- A continuación, vamos a ver un ejemplo más complejo:
- Crear un método `Calcular()` que calcule la suma o la resta de dos números reales y devuelva el resultado

Definición de métodos. Ejemplo

```
class Ejemplo
{
    public static void Main(string[] args)
    {
        double v1 = 4, r;
        char tipo_op = '+';
        r = calcular(tipo_op, v1, 3);
        if (r != System.Double.MaxValue)
            System.Console.WriteLine("{0} {1} {2} = {3}", v1, tipo_op, 3, r);
        else
            System.Console.WriteLine("La operación no es válida...");
        System.Console.ReadLine();
    }
    public static double calcular(char c, double op1, double op2)
    {
        double res;
        if (c == '+')
            res = op1 + op2;
        else
            if (c == '-')
                res = op1 - op2;
            else
                res = System.Double.MaxValue;
        return res;
    }
}
```

Definición de métodos. Ejemplo2

```
class Ejemplo
{
    public static void Main(string[] args)
    {
        double v1 = 4;
        char tipo_op = '+';
        if (calcular(tipo_op, v1, 3) != System.Double.MaxValue)
            System.Console.WriteLine("{0} {1} {2} = {3}", v1, tipo_op, 3, calcular(tipo_op, v1, 3));
        else
            System.Console.WriteLine("La operación no es válida...");
        System.Console.ReadLine();
    }
    public static double calcular(char c, double op1, double op2)
    {
        if (c == '+')
            return op1 + op2;
        if (c == '-')
            return op1 - op2;
        else
            return System.Double.MaxValue;
    }
}
```

Definición de métodos

- Para el ejemplo anterior:
 - Definimos un método de nombre `calcular()` que recibirá tres parámetros:
 - El primero, de tipo `char`, recibirá el tipo de operación a realizar. Los valores permitidos serán el carácter '+' o el carácter '-'.
 - Los dos parámetros restantes, de tipo `double`, recibirán los valores que hay que sumar o restar.
 - El resultado de la operación, de tipo `double`, es el valor a retornar.

Actividad 17

Desarrolla un método llamado **GeneradorPassSuperSeguras()**:

Para eso, deberá crear claves con:

- 1 letra mayúscula
- Entre 6 y 8 dígitos aleatorios
- 1 símbolo
- 2 letras minúsculas

15 MINUTOS

Tema 1: Introducción al C#

Asignatura: Desarrollo web en entorno servidor
CS Desarrollo de Aplicaciones Web

