¿Que es C#?

Es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET. C# es uno de los lenguajes de programación diseñados para la infraestructura de lenguaje común.

Su sintaxis básica deriva de C/C++ y utiliza el modelo de objetos de la plataforma .NET, similar al de Java, aunque incluye mejoras derivadas de otros lenguajes. Su nombre viene de añadir otros dos ++ a C++, lo cual resultaría en #.

Lenguaje case sensitive (Sensible a Mayusc.Minusc)

Es intellisense. En algunos ED nos ayuda con la opcion mas adecuada

Siempre hay que terminar las sentencias con ;

¿Qué es el framework .NET?

Plataforma de Desarrollo y ejecución de aplicaciones compuesta de:

- Bibliotecas de Funcionalidad (Class Library)
- Lenguajes de Programación
- Compiladores
- Herramientas de Desarrollo (IDE & Tools)
- Entorno de Ejecución (Runtime)

En **C#** los **Namespaces** se componen de un conjunto de objetos relacionados como clases, delegados, estructuras, interfaces, etc. Así mismo con los **Namespaces** organizamos mejor nuestro código y lo mantenemos bien limpio y estructurado.

VARIABLE EN C#

Deberemos siempre crearla antes de utilizarla

tipo nombre; (los nombre siempre descriptivo la primra palabra minúscula y las siguiente empiezan en Mayusculas)

Podemos asignar a la vez que la creamos

tipo nombre = “Pepe” ;

tipo nombre = “Pepe”, apellidos=”Lopez”;

Tipos:

sbyte	De -128 a 127	Entero de 8 bits con signo
byte	De 0 a 255	Entero de 8 bits sin signo
short	De -32 768 a 32 767	Entero de 16 bits con signo
ushort	De 0 a 65.535	Entero de 16 bits sin signo
int	De -2.147.483.648 a 2.147.483.647	Entero de 32 bits con signo
uint	De 0 a 4.294.967.295	Entero de 32 bits sin signo
long	De -9.223.372.036.854.775.808 a 9.223.372.036.854.775.807	Entero de 64 bits con signo
ulong	De 0 a 18.446.744.073.709.551.615	Entero de 64 bits sin signo
float	De $\pm 1{,}5 \times 10^{-45}$ a $\pm 3{,}4 \times 10^{38}$	7 dígitos
double	De $\pm 5{,}0 \times 10^{-324}$ a $\pm 1{,}7 \times 10^{308}$	15-16 dígitos
decimal	De $\pm 1{,}0 \times 10^{-28}$ to $\pm 7{,}9228 \times 10^{28}$	28-29 dígitos significativos
char	U+0000 a U+FFFF	Carácter Unicode de 16 bits

bool	Booleano	true, false
String	Cadena de caracteres	

Mostrar el valor de una variable en pantalla

Una vez que sabemos cómo mostrar un número en pantalla, es sencillo mostrar el valor de una variable. Para un número hacíamos cosas como

Console.WriteLine(3+4); pero si se trata de una variable es idéntico:

Console.WriteLine(suma);

O bien, si queremos mostrar un texto además del valor de la variable, podemos indicar el texto entre comillas, detallando con {0} en qué parte del texto queremos que aparezca el valor de la variable, de la siguiente forma:

Console.WriteLine("La suma es {0}", suma);

Si se trata de más de una variable, indicaremos todas ellas tras el texto, y detallaremos dónde debe aparecer cada una de ellas, usando {0}, {1} y así sucesivamente:

Console.WriteLine("La suma de {0} y {1} es {2}", primerNumero, segundoNumero, suma);

Console.Wirte(“xxx); hace lo mismo pero sin salto de linea

Datos por el usuario: ReadLine

Si queremos que sea el usuario de nuestro programa quien teclee los valores, necesitamos una nueva orden, que nos permita leer desde teclado. Pues bien, al igual que tenemos Console.WriteLine ("escribir línea), también existe Console.ReadLine ("leer línea").

Para leer textos, haríamos

texto =Console.ReadLine();

pero eso ocurrirá en el próximo tema, cuando veamos cómo manejar textos. De momento, nosotros sólo sabemos manipular números enteros, así que deberemos convertir ese dato a un número entero, usando Convert.ToInt32:

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

Operadores

= de asignación

Operador de incremento ++ o decremento --

El operador de incremento unario ++ incrementa su operando en 1. El de decremento lo sustrae en 1 unidad. El operando debe ser una variable, un acceso de propiedad o un acceso de indexador.

El operador de incremento se admite en dos formas: el operador de incremento posfijo (x++) y el operador de incremento prefijo (++x).

CUIDADO

El resultado de x++ es el valor de x*antes* de la operación, tal y como se muestra en el ejemplo siguiente:

```
int i = 3;
Console.WriteLine(i);   // output: 3
Console.WriteLine(i++); // output: 3
Console.WriteLine(i);   // output: 4
```

El resultado de ++x es el valor de x*después* de la operación, tal y como se muestra en el ejemplo siguiente:

```
double a = 1.5;
Console.WriteLine(a);   // output: 1.5
Console.WriteLine(++a); // output: 2.5
Console.WriteLine(a);   // output: 2.5
```

Operadores aritmeticos

Operador *	Multiplicación
Operador /	División
Operador %	Resto de división entera (mod)

Operador +	Suma
Operador -	Resta

Asignación compuesta

Para un operador binario op, una expresión de asignación compuesta con el formato

x op= y es equivalente a x = x op y

salvo que x solo se evalúa una vez.

```
int a = 5;

a += 9;

Console.WriteLine(a);  // output: 14


a -= 4;

Console.WriteLine(a);  // output: 10


a *= 2;

Console.WriteLine(a);  // output: 20


a /= 4;

Console.WriteLine(a);  // output: 5


a %= 3;

Console.WriteLine(a);  // output: 2
```

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

Orden de prioridad de los operadores Sencillo: En primer lugar se realizarán las operaciones indicadas entre paréntesis. Luego la negación. Después las multiplicaciones, divisiones y el resto de la división. Finalmente, las sumas y las restas. En caso de tener igual prioridad, se analizan de izquierda a derecha.

Operador Operación

<	Menor que
>	Mayor que
<=	Menor o igual que
>=	Mayor o igual que
==	Igual a
!=	No igual a (distinto de)

CONDICIONALES

```
if (PRUEBA LOGICA)
{
    Console.WriteLine("Se cumple la prueba.");
}
else
{
    Console.WriteLine("No se cumple la prueba.");
}
```

Si las sentencias son únicas, puedo quitar las llaves

```
if (num==10)
    Console.WriteLine("El número es igual a 10");
else
    Console.WriteLine("El número no es igual a 10");
```

If anidados

```
if (num==10)
{
    Console.WriteLine("El número es igual a 10");
}
else if (num>5)
{
    Console.WriteLine("El número es mayor que 5");
}
else if (num>15)
{
    Console.WriteLine("El número es mayor que 15");
}
else
{
    Console.WriteLine("El número no es 10 ni mayor que 5");
}
```

operador condicional ? (ternario)

variable = (condicion) **?** valorVerdadero **:** valorFalso**;**

Operadores de comparación

Operador	Operación	Ejemplo	Resultado
^	O exclusivo	If (test1)^(test2)	Cierto si alguno es cierto y el otro NO lo e
!	Negación	If ! test	Invierte el resultado de test
&&	Y lógico	If (test1)&&(test2)	Igual que &, pero solo evalua test2 si test1 es cierto
||	O lógico	If (test1)||(test2)	Igual que |, pero solo evalua test2 si test1 es falso
&	Y lógico	If (test1)&(test2)	Cierto si ambos son ciertos bit a bit
|	O lógico	If (test1) | (test2)	Cierto si alguno de los dos es cierto bit a b

Diferencia entre & y &&. & compara en binario

(1 & 2) false
(1 &&& 2) true

```
int a = 60; // En binario, 60 0011 1100
int b = 13; // En binario, 13 0000 1101
int c = a & b; // "c" dará 12 0000 1100,
```

No utilizar &

switch

Si queremos ver **varios posibles valores**, seria muy pesado tener que hacerlo con muchos
"if" seguidos o encadenados. La alternativa es la orden "switch", cuya sintaxis es

```
switch (expresión)
{
        case valor1:
                sentencia1;
                break;
        case valor2:
                sentencia2;
                sentencia2b;
                break;
...
        case valorN:
                sentenciaN;
                break;
        default:
                otraSentencia;
                break;
}
```

Estructuras repetitivas

While

se repita mientras se cumpla una cierta condición

```
while (condición)
        sentencia;
```

Es decir puede NO EJECUTARSE NINGUNA VEZ

do ... while

la condicion se comprueba **al final**.

```
do
                sentencia;
while (condición)
```

Al igual que en el caso anterior, si queremos que se repitan varias ordenes (es lo habitual), deberemos encerrarlas entre llaves.

Es decir AL MENOS SE EJECUTA VEZ

for

Esta es la orden que usaremos habitualmente para crear bucles un cierto numero de veces que ya sabemos.

```
for (valorInicial; CondiciónRepetición; Incremento)
      Sentencia;
```

Es muy habitual usar la letra "i" como contador,

```
for (i=1; i<=10; i=i+1)
...
```

La orden para incrementar el valor de una variable ("i = i+1") se puede escribir de la forma abreviada "i++"

Podemos salir de un bucle "for" antes de tiempo con la orden "**break**":

```
for (contador=1; contador<=10; contador++)
{
if (contador==5)
break;
Console.Write("{0} ", contador);
}
```

Podemos saltar alguna repeticion de un bucle con la orden "**continue**":

```
for (contador=1; contador<=10; contador++)
{
if (contador==5)
continue;
Console.Write("{0} ", contador);
}
```

El resultado de este programa es:
1 2 3 4 6 7 8 9 10
En el podemos observar que no aparece el valor 5.

Ejercicios:

Crear un programa que calcule el factorial de un numero dado

Crear un programa que nos pida una clave de usuario, hasta que pongamos *.
Cuando esto ocurra.
Mostraremos por consola "La clave es: xxxx" (sin el *)

Es decir podemos poner: Clave: 22222* No sdira 2222

Crear un programa que nos muestre los primeros X números primos.
El valor de X nos lo dará el usuario

Crear un programa que muestre las letras de la Z (mayuscula) a la A (mayuscula, descendiendo).

Para "contar" no necesariamente hay que usar números. Por ejemplo, podemos usar letras, si el contador lo declaramos como "char" y los valores inicial y final se detallan entre comillas simples, así:

```
char letra;
```

```
for (letra='a'; letra<='z'; letra++)
    Console.Writeline("{0} ", letra);
```

Secuencias de escape: \n y otras.

Como hemos visto, los textos que aparecen en pantalla se escriben con WriteLine, indicados entre paréntesis y entre comillas dobles. Entonces surge una dificultad: ¿cómo escribimos una comilla doble en pantalla? La forma de conseguirlo es usando ciertos caracteres especiales, lo que se conoce como "secuencias de escape".

Existen ciertos caracteres especiales que se pueden escribir después de una barra invertida (\) y que nos permiten conseguir escribir esas comillas dobles y algún otro carácter poco habitual.

Por ejemplo, con \" se escribirán unas comillas dobles, y con \' unas comillas simples, o con \n se avanzará a la línea siguiente de pantalla. Estas secuencias especiales son las siguientes:

Secuencia	Significado
\a	Emite un pitido
\b	Retroceso (permite borrar el último carácter)
\f	Avance de página (expulsa una hoja en la impresora)
\n	Avanza de línea (salta a la línea siguiente)
\r	Retorno de carro (va al principio de la línea)
\t	Salto de tabulación horizontal
\v	Salto de tabulación vertical
\'	Muestra una comilla simple
\"	Muestra una comilla doble
\\	Muestra una barra invertida
\0	Carácter nulo (NULL)

Ejercicio: Crear un programa que pida al usuario que teclee cuatro letras y las muestre en pantalla juntas, pero en orden inverso, y entre comillas dobles. Por ejemplo si las letras que se teclean son a, l, o, h, escribiría "hola".

Las matrices o arrays (también llamados arreglos o vectores)

son estructuras que puede almacenar varios valores simultáneamente (siempre del mismo tipo). Cada uno de estos valores se identifica mediante un número al cual se llama índice. Así, para acceder al primer elemento del array habría que usar el índice cero, para el segundo el índice uno, para el tercero el índice dos, y así sucesivamente. Vamos a ver cómo se declara un array:

tipo[] variable; int[] edades;

Si sabemos desde el principio ***cuantos datos tenemos*** (por ejemplo 4), les reservaremos espacio con

int[] edades = new int[4];

edades[0] = 18;
edades[1] = 20;
edades[2] = 17;
edades[3] = 24;

Si sabemos desde el principio los **valores** de los datos, pondremos

int[] edades = new int[] {18, 20, 17, 24};

Por definición un arreglo es de una medida fija y por lo tanto **no puede ser extendido**, es decir no podemos redimensionarlo. Esto aplica para arreglos de cualquier tipo: int, string, char, etc.

Para arreglar esto lo haremos con listas, que ya lo veremos mas adelante

Referencia a los valores.

edades[0] será 18

con array es muy comun utilizar los bucles for

```
for (i=0; i<=4; i++)
```

```
    Console.WriteLine(edades[i]);
```

Tamaños de los arreglos

```
int tamanoEdades = edades.Length;
```

Ejercicio: Hacer una aplicación de consola que muestre por pantalla el valor mínimo del array tamaño 5 donde previamente el usuario ha introducido los valores del array

```
string[] cadenas = { "def", "agh", "abc", "efg", "bcd", "klm", "iao", "hñu" };
//Ordena el array
Array.Sort(cadenas);
```

```
//Recorre el array comenzando con el índice 0 y terminando con el índice Length-1
foreach (string texto in cadenas)
 {
     Console.WriteLine(texto);
 }
```

Ejercicios propuestos:

• Un programa que pida al usuario 5 números reales y luego los muestre en el orden contrario al que se introdujeron.

• Un programa que pida al usuario 10 números enteros y calcule (y muestre) cuál es el mayor de ellos.

Arrays bidimensionales o mas

Podemos declarar arrays de dos o más dimensiones. Por ejemplo, si queremos guardar datos de dos grupos de alumnos, cada uno de los cuales tiene 20 alumnos:

```
int datosAlumnos[2,20]
```

y entonces sabemos que los datos de la forma datosAlumnos[0,i] son los del primer grupo, y los datosAlumnos[1,i] son los del segundo.

```
int[,] notas1 = new int[2,3];      // array entero de 2x3
notas1[0,0] = 1;
notas1[0,1] = 2;
notas1[0,2] = 3;
notas1[1,0] = 3;
notas1[1,1] = 4;
notas1[1,2] = 5;

int[,] notas2 =        // el mismo que antes
{
   {1, 2, 3},
   {3, 4, 5}
 };
```

Tamaños de los arreglos

notas2.GetLength(0) // Da el número de filas (la primera dimensión)
notas2.GetLength(1) //Da el número de columnas (la segunda dimensión)

arrays de arrays

```
int[][] notas = new int[3][];
notas [0] = new int[5];
notas [1] = new int[4];
notas [2] = new int[2];
```

Cada uno de los elementos es una matriz unidimensional de enteros. El primer elemento es una matriz de 5 enteros, el segundo es una matriz de 4 enteros y el tercero es una matriz de 2 enteros.

```
int[][] notas = new int[3][];
```

```
notas [0] = new int[] { 1, 3, 5, 7, 9 };
notas [1] = new int[] { 0, 2, 4, 6 };
notas [2] = new int[] { 11, 22 };
```

Control de errores

La sentencia para manejo de errores Try-Catch en C# es utilizado en programación .Net para evitar romper el flujo de trabajo de una aplicación.

La instrucción Try-Catch esta formado de un bloque try y seguida de este se coloca el catch.

Try: se encarga de encapsular todas las operaciones.

```
try
{
    //boque try con todas las operaciones
}
```

Catch: captura los errores generados en el bloque Try, aqui se manejan las diferentes excepciones.

```
catch (Exception ex) //bloque catch para captura de error
{
    //acción para manejar el error
}
```

Estructuras

Es una agrupación de datos, los cuales no necesariamente son del mismo tipo. Se definen con la palabra "**struct**". La serie de datos que van a formar.

Primero deberemos declarar cual va a ser la estructura, lo que no se puede hacer dentro de "Main".
Más adelante, ya dentro de "Main", podremos declarar variables de ese nuevo tipo.

```
using System;

namespace ConsoleApp8
{
    class Program
    {
        struct tipoPersona
        {
            public string nombre;
            public int edad;
            public float nota;
        }

        static void Main(string[] args)
        {
            tipoPersona persona;
            persona.nombre = "Juan";
            persona.edad = 20;
            persona.nota = 7.5F;

            Console.WriteLine("La edad de {0} es {1}", persona.nombre,
persona.edad);
            Console.ReadLine();

        }
    }
}
```

Arrays de estructuras

```
tipoPersona[] persona = new tipoPersona[100];


persona[0].nombre = "Juan";
persona[0].inicial = 'J';
 persona[0].edad = 20;
 persona[0].nota = 7.5f;
```

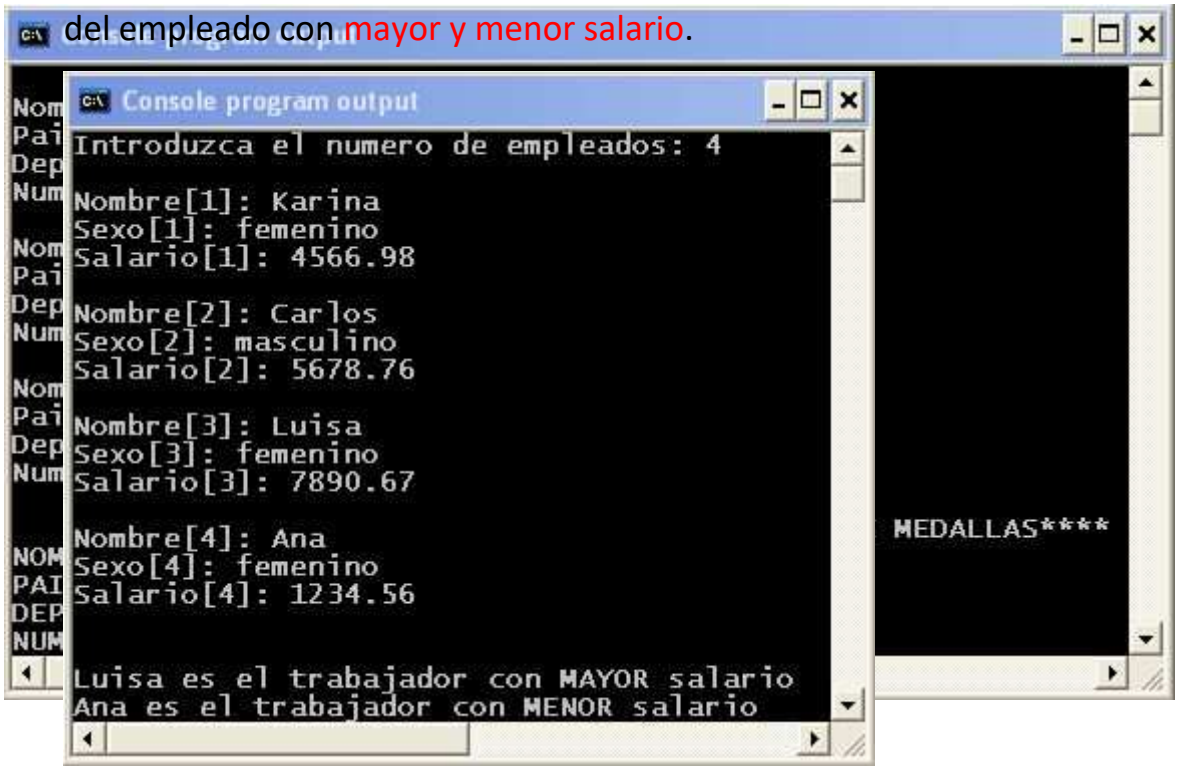
Ejercicio:

La información de todos los empleados de la empresa Bicicletas de Jerez está almacenada en una variable de tipo struct llamada “empleado”. La información con que se cuenta de cada empleado es:

- nombre
- sexo
- sueldo.

Se pide: Realizar un programa que lea en un array de estructuras los datos de los N trabajadores de la empresa y que imprima los datos

del empleado con mayor y menor salario.



Estructuras anidadas

A partir de la declaración de las siguientes estructuras realice un programa en que lea el array **"deportistas"** y devuelva los datos (nombre, país, deporte) del atleta que ha ganado mayor número de medallas.

```
struct datos {
   string nombre;
   string país;];
};

struct atleta {
    string deporte;
    datos persona;
    int nmedallas;
};
struct atleta deportistas[30];
```

FUNCIONES

Hasta ahora hemos estado pensando los pasos que deberíamos dar para resolver un cierto problema, y hemos creado programas a partir de cada uno de esos pasos. Esto es razonable cuando los problemas son sencillos, pero puede no ser la mejor forma de actuar cuando se trata de algo más complicado.

A partir de ahora vamos a empezar a intentar descomponer los problemas en trozos más pequeños, que sean más fáciles de resolver. Esto nos puede suponer varias ventajas:

- Cada "trozo de programa" independiente será más fácil de programar, al realizar una función breve y concreta.
- El "programa principal" será más fácil de leer, porque no necesitará contener todos los detalles de cómo se hace cada cosa.
- Evitaremos mucho código repetitivo.

Esos "trozos" de programa son lo que suele llamar "subrutinas", "procedimientos" o " **funciones** ".

Definicion

forma básica de **definir** una función será indicando su **nombre** seguido de unos **paréntesis** vacios (o no) y precediéndolo por palabras reservadas, como "public static **void**",

Después, entre llaves indicaremos todos los pasos que queremos que dé ese "trozo de programa".

```
public static void Saludar()
{
  Console.Write("Bienvenido al programa ");
  Console.WriteLine("de ejemplo");
  }
```

Ahora desde dentro del cuerpo de nuestro programa, podríamos "**llamar**" a esa función:

```
public static void Main()
{
  Saludar();
  ...
}
```

Un detalle importante: tanto la función habitual "Main" como la nueva función "Saludar" serían parte de nuestra "class", es decir, son independientes, (no están ninguna dentro de la otra)

Ejemplo:

```
LeerDatosDeFichero();
do {
    MostrarMenu();
    opcion = PedirOpcion();
    switch( opcion ) {
        case 1: BuscarDatos(); break;
        case 2: ModificarDatos(); break;
        case 3: AnadirDatos(); break;
        ...
```

Ejercicio:

Crea una función llamada "DibujarCuadrado10x10" y otra función llamada “borrar3lineas”, la primera dibuja un cuadrado formato por 10 filas con 10 asteriscos cada una, la segunda creara 3 lineas en blanco. En el "Main" llamara tres veces a cada función

Funciones con parametros

Es muy frecuente que nos interese indicarle a nuestra función ciertos datos con los que queremos que trabaje.
Los llamaremos "parámetros" y los indicaremos dentro del paréntesis que sigue al nombre de la función, separados por comas. Para cada uno de ellos, deberemos indicar su tipo de datos (por ejemplo "int") y luego su nombre.

```
public static void EscribirSuma( int a, int b )
 { ...
 }
```

El valor "**void**" Cuando queremos dejar claro que una función no tiene que devolver ningún valor.

```
public static int Suma ( int a, int b )
 {
   return a+b;
 }
```

Ejercicio
Programa que me pida un número y me devuelva el factorial del mismo

Variables locales y variables globales

Hasta ahora, hemos declarado las variables dentro de "Main". Ahora nuestros programas tienen varios "bloques", así que las variables se comportarán de forma distinta según donde las declaremos.

Las variables se pueden declarar dentro de un bloque (una función), y entonces sólo ese bloque las conocerá, no se podrán usar desde ningún otro bloque del programa. Es lo que llamaremos "**variables locales**".

Por el contrario, si declaramos una variable al comienzo del programa, fuera de todos los "bloques" de programa, será una "**variable global**", a la que se podrá acceder desde cualquier parte.
(Por ahora, una variable global deberá llevar siempre la palabra "static")

En general, deberemos intentar que la mayor cantidad de variables posible sean locales (lo ideal sería que todas lo fueran).
La forma correcta de pasar datos entre distintos trozos de programa no es a través de variables globales, sino usando los parámetros de cada función y los valores devueltos

Modificando parámetros. Parámetro por referencia

Podemos modificar el valor de un dato que recibamos como parámetro, pero posiblemente el resultado no será el que esperamos. Vamos a verlo con un ejemplo:
Supongamos el siguiente código

```
public static void Duplicar(int x)
{
        Console.WriteLine(" El valor recibido vale {0}", x);    -> 5
        x = x * 2;
       Console.WriteLine(" y ahora vale {0}", x);  -> 10
}
```

```
public static void Main()
{
  int n = 5;
 Console.WriteLine("n vale {0}", n);  -> 5
  Duplicar(n);
 Console.WriteLine("Ahora n vale {0}", n);  -> 5
}
```

Vemos que al salir de la función, no se conservan los cambios que hagamos a esa variable que se ha recibido como parámetro.

Si quisiéramos que la variable pasada como parámetro **SI** conserve los cambio tengo que hacerlo pasando dichos parámetros "**por referencia**", lo que se indica usando la palabra "**ref**", tanto en la definición de la función como en su llamada

```
public static void Duplicar(ref  int x)
{
         Console.WriteLine(" El valor recibido vale {0}", x);    -> 5
         x = x * 2;
        Console.WriteLine(" y ahora vale {0}", x);  -> 10
}

public static void Main()
{
  int n = 5;
 Console.WriteLine("n vale {0}", n);  -> 5
  Duplicar(ref  n);
 Console.WriteLine("Ahora n vale {0}", n);  -> 10
}
```

Esto lo podríamos hacer con mas de un parámetro.

```
public static void Intercambia(ref  int x, ref  int y)
```

Además de pasar parámetros por valor y por referencia en C# existe una posibilidad adicional que no existe en otros lenguajes, como los "**parámetros de salida**". Esto se hace con la palabra **out**

La diferencia es: los parámetros pasados como ref deben estar inicializados previamente (se va a modificar su valor), mientras que los parámetros pasados como out no es necesario que lo estén (se va a asignar un valor sin importarnos lo que tuviera previamente).

```
    public static void assignaValor(out int parametro1, out int
parametro2)
    {
       parametro1 = 14*2;
       parametro2 = 23*2;
    }

    public static void Main()
    {
       int num1, num2;
       assignaValor(out num1, out num2);
       Console.WriteLine("Numero 1 {0} Numero 2 {1}  ", num1, num2);
       Console.ReadLine();
    }
```

Ejercicios propuestos:
Crea una función "Intercambiar", que intercambie el valor de los dos números enteros que se le indiquen como parámetro. Crea también un programa que la pruebe.

Dime numero 1: 45
Dime numero 2: 56

El numero 1 es 45 y el 2 es 56
Llamais a la función
El numero 1 es 56 y el 2 es 45

Algunas funciones útiles de cadenas

Substring, **extraer** parte del contenido de una cadena

```
saludo = frase.Substring(0,4);
```

Recibe dos parámetros: la posición a partir de la que queremos empezar y la cantidad de caracteres que queremos obtener. Si omitir el segundo número, y entonces se extraerá desde la posición indicada hasta el final de la cadena.

IndexOf busca una cadena

nombre.**IndexOf**("Juan")

Para ver si una cadena contiene un cierto texto, podemos usar IndexOf ("posición de"), que nos dice en qué posición se encuentra, siendo 0 la primera posición (o devuelve el valor -1, si no aparece)

Podemos añadir un segundo parámetro opcional, que es la posición a partir de la que queremos buscar:

```
if (nombre.IndexOf("Juan", 5) >= 0) ...
```

De forma similar, **LastIndexOf** ("última posición de") indica la última aparición (es decir, busca de derecha a izquierda).

Si solamente queremos ver si aparece, pero no nos importa en qué posición está, nos bastará con usar "**Contains**":

```
if (nombre.Contains("Juan")) ...
```

Algunas más

ToUpper() convierte a mayúsculas:

```
nombreCorrecto = nombre.ToUpper();
```

ToLower() convierte a minúsculas:

```
password2 = password.ToLower();
```

Insert(int posición, string subcadena): Insertar una subcadena en una cierta posición de la cadena inicial:

```
nombreFormal = nombre.Insert(0,"Don");
```

Remove(int posición, int cantidad): Elimina una cantidad de caracteres en cierta posición:

```
apellidos = nombreCompleto.Remove(0,6);
```

Replace(string textoASustituir, string cadenaSustituta): Sustituye una cadena (todas las veces que aparezca) por otra:
nombreCorregido = nombre.Replace("Pepe", "Jose");

CompareTo: Compara cadenas, al giaul que el > en numero
if (frase.CompareTo("hola") > 0) Console.WriteLine("La frase es mayor que hola");

Si NO quisiera distinguir entre mayúscula y minúscula
if (String.Compare(frase, "hola", true) > 0) Console.WriteLine("Es mayor que hola (mays o mins)");

Algunas funciones útiles de numeros

Numeros aleatorios: debemos crear un objeto de tipo "**Random**" (una única vez), y luego llamaremos a "**Next**" cada vez que queramos obtener valores entre dos extremos:

```
// Creamos un objeto Random
Random generador = new Random();

// Generamos un número entre dos valores dados // (el segundo límite no está incluido)
int aleatorio = generador.Next(1, 101);
```

Funciones matematicas. Todas ellas se usan precedidas por "**Math**."

Abs(x): Valor absoluto
Acos(x): Arco coseno
Asin(x): Arco seno
Atan(x): Arco tangente
Atan2(y,x): Arco tangente de y/x (por si x o y son 0)
Ceiling(x): El valor entero superior a x y más cercano a él
Cos(x): Coseno
Cosh(x): Coseno hiperbólico
Exp(x): Exponencial de x (e elevado a x)
Floor(x): El mayor valor entero que es menor que x
Log(x): Logaritmo natural (o neperiano, en base "e")
Log10(x): Logaritmo en base 10
Pow(x,y): x elevado a y

Round(x, cifras): Redondea un número
Sin(x): Seno
Sinh(x): Seno hiperbólico
Sqrt(x): Raíz cuadrada
Tan(x): Tangente
Tanh(x): Tangente hiperbólica

- La raíz cuadrada de 4 se calcularía haciendo `x = Math.Sqrt(4);`
- La potencia: para elevar 2 al cubo haríamos `y = Math.Pow(2, 3);`
- El valor absoluto: para trabajar sólo con números positivos usaríamos `n = Math.Abs(x);`

Ejercicios

Queremos simular el lanzamiento de un dado 50.000 veces
Una vez realizado esto queremos lo siguiente

- Introduzca numero (entre el 1 y el 6): 5
- El numero 5 ha aparecido 3567 veces, es decir el 23%

Crear una función que le llaméis numVeces(4) ->

Programa que pida una frase y devuelva la palabras que hay en ella

Frase: Ha de antiguo la costumbre mi padre el baron de Mies
Palabra 1: Ha
Palabra 2: de
Palabra 3: antiguo
Palabra 4: la
…….