1.1. ¿Qué es C #? ¿Qué entorno usaremos?

C# es un lenguaje de programación de ordenadores. Se trata de un lenguaje moderno, evolucionado a partir de C y C++, y con una sintaxis muy similar a la de Java. Los programas creados con C# no suelen ser tan rápidos como los creados con C, pero a cambio la productividad del programador es mucho mayor y es más difícil cometer errores.

Se trata de un lenguaje creado por Microsoft para realizar programas para su plataforma .NET, pero fue estandarizado posteriormente por ECMA y por ISO, y existe una implementación de código abierto: ".**NET Core**", que poco a poco sustituirá a la anterior y que está disponible para Windows, Linux, Mac OS X y otros sistemas operativos.

Nosotros usaremos tanto **Visual Studio Code** como **Visual Studio Comunity** para el desarrollo con C#

Los **pasos** que seguiremos para crear un programa en C# serán:

1.1.1. Escribir el programa en lenguaje C# (**fichero fuente**), con cualquier editor de textos.
1.1.2. Compilarlo con nuestro compilador. Esto creará un "**fichero ejecutable**".
1.1.3. Lanzar el fichero ejecutable.

La mayoría de los compiladores actuales permiten dar todos estos pasos desde un único entorno, en el que escribimos nuestros programas, los compilamos, y los depuramos en caso de que exista algún fallo.

Tras el siguiente apartado veremos un ejemplo de entorno desde el que realizar nuestros programas, dónde localizarlo y cómo instalarlo.

1.2. Escribir un texto en C#

Vamos con un primer ejemplo de programa en C#, posiblemente el más sencillo de los que "hacen algo útil". Se trata de escribir un texto en pantalla. La apariencia de este programa la vimos en el tema anterior. Vamos a analizarlo ahora con más detalle:

```
public class Ejemplo_01_02a
{
    public static void Main()
    {
        System.Console.WriteLine("Hola");
    }
}
```

Esto escribe "Hola" en la pantalla. Pero hay muchas "cosas raras" alrededor de ese "Hola", de modo vamos a comentarlas antes de proseguir, aunque muchos de los detalles los aplazaremos para más adelante. En este primer análisis, iremos desde dentro hacia fuera:

- WriteLine("Hola"); : "Hola" es el texto que queremos escribir, y WriteLine es la orden encargada de escribir (Write) una línea (Line) de texto en pantalla.
- Console.WriteLine("Hola"); : WriteLine siemprp irá precedido de "Console." porque es una orden de manejo de la "consola" (la pantalla "negra" en modo texto del sistema operativo).
- System.Console.WriteLine("Hola"); : Las órdenes relacionadas con el manejo de consola (Console) pertenecen a la categoría de sistema (System).
- Las llaves { y } se usan para delimitar un bloque de programa. En nuestro caso, se trata del bloque principal del programa (Main).
- public static void Main() : Main indica cual es "el cuerpo del programa", la parte principal (un programa puede estar dividido en varios fragmentos, como veremos más adelante).
- Todos los programas tienen que tener un bloque "Main". Los detalles de por qué hay que poner delante "public static void" y de por qué se pone después un paréntesis vacío los iremos aclarando más tarde. De momento, deberemos memorizar que ésa será la forma correcta de escribir "Main".
- public class Ejemplo_01_02a : De momento pensaremos que "Ejemplo_01_02a" es el nombre de nuestro programa. Una línea como esa deberá existir también siempre en nuestros programas (aunque el nombre no tiene por qué ser tan "rebuscado"), y eso de "public class" será obligatorio. Nuevamente, aplazamos para más tarde los detalles sobre qué quiere decir "class" y por qué debe ser "public".

Como se puede ver, mucha parte de este programa todavía es casi un "acto de fe" para nosotros. Debemos creernos que "se debe hacer así". Poco a poco iremos detallando el por qué de "public", de "static", de "void", de "class"... Por ahora nos limitaremos a "rellenar" el cuerpo del programa para entender los conceptos básicos de programación.

Solo un par de cosas más antes de seguir adelante:

- Cada orden de C# debe terminar con un **punto y coma** (;)
- C# es un lenguaje de formato libre, de modo que puede haber varias órdenes en una misma línea, u órdenes separadas por varias líneas o espacios entre medias. Lo que realmente indica donde termina una orden y donde empieza la siguiente son los puntos y coma y las llaves.
  Por ese motivo, el programa anterior se podría haber escrito también así (aunque no es aconsejable, porque puede resultar menos legible):

```
public class
Ejemplo_01_02b
{
public
static
void Main() { System.Console.WriteLine("Hola"); }  }
```

- De hecho, hay dos formas especialmente frecuentes de colocar la llave de comienzo, y yo usaré ambas indistintamente. Una es como hemos hecho en el primer ejemplo: situar la llave de apertura en una línea, sola, y justo encima de la llave de cierre correspondiente.

Esto es lo que muchos autores llaman el "estilo C". La segunda forma habitual es situándola a continuación del nombre del bloque que comienza (el "estilo Java"), así:

```
public class Ejemplo_01_02c {
      public static void Main(){
            System.Console.WriteLine("Hola");
      }
}
```

(esta es la forma que se empleará preferentemente en este texto cuando estemos trabajando con fuentes de mayor tamaño, para que ocupe un poco menos de espacio; en los primeros fuentes usaremos el "estilo C", que tiende a resultar más legible).

- La gran mayoría de las órdenes que encontraremos en el lenguaje C# son palabras en inglés o abreviaturas de éstas, pero hay que tener en cuenta que C# **distingue entre mayúsculas y minúsculas**, por lo que "WriteLine" es una palabra reconocida, pero "writeLine", "WRITELINE" o "Writeline" no lo son.

Ejercicio propuesto:

Ejercicio propuesto 1.2.1: *Crea un programa en C# que te salude por tu nombre (por ejemplo, "Hola, Pepe").*

1.3. Uso de Visual Studio Code

Puedes seguir esta guía para instalar y el uso básico de Visual Studio Code con C#:

[Visual Studio Code: cómo preparar un entorno de trabajo para .NET Core]

1.4. Mostrar números enteros en pantalla

Cuando queremos escribir un texto "tal cual", como en el ejemplo anterior, lo encerramos entre comillas. Pero no siempre querremos escribir textos prefijados. En muchos casos, se tratará de algo que habrá que calcular.

El ejemplo más sencillo es el de una operación matemática. La forma de realizarla es simple: no usar comillas en WriteLine. Entonces, C# intentará analizar el contenido para ver qué puede significar. Por ejemplo, para sumar 3 y 4 bastaría hacer:

```
public class Ejemplo_01_04a
{
    public static void Main()
    {
        System.Console.WriteLine(3+4);
    }
}
```

Esto mostrará un 7 en la pantalla. Pero hay mucho alrededor de ese "7", que iremos comentando más adelante.

Ejercicios propuestos:

Ejercicio propuesto 1.4.1: *Crea un programa que diga el resultado de sumar 118 y 56.*

Ejercicio propuesto 1.4.2: *Crea un programa que diga el resultado de sumar 12345 y 67890.*

(**Recomendación**: no "copies y pegues" aunque dos ejercicios se parezcan. Volver a teclear cada nuevo ejercicio te ayudará a memorizar las estructuras básicas del lenguaje).

1.5. Operaciones aritméticas básicas

1.5.1. Operadores

Parece evidente que el símbolo de la suma será un +, y podemos esperar cuál será el de la resta, pero alguna de las operaciones matemáticas habituales tienen símbolos menos intuitivos. Veamos cuales son los más importantes:

Operador	Operación
+	Suma
-	Resta, negación
*	Multiplicación
/	División
%	Resto de la división ("módulo")

Así, podemos calcular el resto de la división entre dos números de la siguiente forma:

```
public class Ejemplo_01_05_01a
{
    public static void Main()
    {
        System.Console.WriteLine("El resto de dividir 19 entre 5 es");
        System.Console.WriteLine(19 % 5);
    }
}
```

Ejercicios propuestos:

Ejercicio propuesto 1.5.1.1: *Hacer un programa que calcule el producto de los números 12 y 13.*

Ejercicio propuesto 1.5.1.2: *Hacer un programa que calcule la diferencia (resta) entre 321 y 213.*

Ejercicio propuesto 1.5.1.3: *Hacer un programa que calcule el resultado de dividir 301 entre 3.*

Ejercicio propuesto 1.5.1.4: *Hacer un programa que calcule el resto de la división de 301 entre 3.*

1.5.2. 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.

Así, el siguiente ejemplo da como resultado 23 (primero se multiplica 4*5 y luego se le suma 3) en vez de 35 (no se suma 3+4 antes de multiplicar, aunque aparezca a la izquierda, porque la prioridad de la suma es menor que la de la multiplicación).

```
public class Ejemplo_01_05_02a
{
    public static void Main()
    {
        System.Console.WriteLine("Ejemplo de precedencia de
        operadores"); System.Console.WriteLine("3+4*5=");
        System.Console.WriteLine(3+4*5);
    }
}
```

Ejercicios propuestos: Calcular (a mano y después comprobar desde C#) el resultado de las siguientes operaciones:

Ejercicio propuesto 1.5.2.1: *Calcular el resultado de -2 + 3 * 5*

Ejercicio propuesto 1.5.2.2: *Calcular el resultado de (20+5) % 6*

Ejercicio propuesto 1.5.2.3: *Calcular el resultado de 15 + -5*6 / 10*

Ejercicio propuesto 1.5.2.4: *Calcular el resultado de 2 + 10 / 5 * 2 - 7 % 1*

1.5.3. Introducción a los problemas de desbordamiento

El espacio del que disponemos para almacenar los números es limitado. Veremos los límites exactos más adelante, pero de momento nos basta saber que si el resultado de una operación es un número "demasiado grande", obtendremos un mensaje de error o un resultado erróneo. Por eso en los primeros ejemplos usaremos números pequeños. Más adelante veremos a qué se debe realmente este problema y cómo evitarlo. Como anticipo, el siguiente programa ni siquiera compila, porque el compilador sabe que el resultado va a ser "demasiado grande":

```
public class Ejemplo_01_05_03a
{
    public static void Main()
    {
        System.Console.WriteLine(10000000*10000000);
    }
}
```

1.6. Introducción a las variables: int

Las **variables** son algo que no contiene un valor predeterminado, un espacio de memoria al que nosotros asignamos un nombre y en el que podremos almacenar datos.

El primer ejemplo nos permitía escribir "Hola". El segundo llegaba un poco más allá y nos permitía sumar dos números que habíamos prefijado en nuestro programa. Pero esto tampoco es "lo habitual", sino que esos números dependerán de valores que haya tecleado el usuario o de cálculos anteriores.

Por eso necesitaremos usar variables, zonas de memoria en las que guardemos los datos con los que vamos a trabajar y también los resultados temporales. Como primer ejemplo, vamos a ver lo que haríamos para sumar dos números enteros que fijásemos en el programa.

1.6.1. Definición de variables: números enteros

Para usar una cierta variable primero hay que **declararla**: indicar su nombre y el tipo de datos que querremos guardar.

El primer tipo de datos que usaremos serán números enteros (sin decimales), que se indican con "int" (abreviatura del inglés "integer"). Después de esta palabra se indica el nombre que tendrá la variable:

```
int primerNumero;
```

Esa orden reserva espacio para almacenar un número entero, que podrá tomar distintos valores, y al que nos referiremos con el nombre "primerNumero".

Ejercicios propuestos:

Ejercicio propuesto 1.6.1.1: *Amplía el "ejemplo 01.05.02a" para declarar tres variables, llamadas n1, n2, n3.*

1.6.2. Asignación de valores

Podemos darle un valor a esa variable durante el programa haciendo

```
primerNumero = 234;
```

O también podemos darles un valor inicial ("inicializarlas") antes de que empiece el programa, en el mismo momento en que las definimos:

```
int primerNumero = 234;
```

O incluso podemos definir e inicializar más de una variable a la vez

```
int primerNumero = 234, segundoNumero = 567;
```

(esta línea reserva espacio para dos variables, que usaremos para almacenar números enteros; una de ellas se llama primerNumero y tiene como valor inicial 234 y la otra se llama segundoNumero y tiene como valor inicial 567).

Después ya podemos hacer operaciones con las variables, igual que las hacíamos con los números:

```
suma = primerNumero + segundoNumero;
```

Ejercicios propuestos:

Ejercicio propuesto 1.6.2.1: *Amplía el ejercicio 1.6.1.1, para que las tres variables n1, n2, n3 estén declaradas en la misma línea y tengan valores iniciales.*

1.6.3. 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

```
System.Console.WriteLine(3+4);
```

pero si se trata de una variable es idéntico (sin comillas, para que el compilador analice su valor de antes de escribir):

```
System.Console.WriteLine(suma);
```

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

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

Si queremos mostrar de más de una variable, detallaremos en el texto dónde debe aparecer cada una de ellas, usando {0}, {1} y tantos números sucesivos como sea necesario, y tras el texto incluiremos los nombres de cada una de esas variables, separados por comas:

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

Ya sabemos todo lo suficiente para crear nuestro programa que sume dos números usando variables:

```
public class Ejemplo_01_06_03a
{
    public static void Main()
    {
        int primerNumero;
        int segundoNumero;
        int suma;
        primerNumero = 234;
        segundoNumero = 567;
        suma = primerNumero + segundoNumero;

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

Repasemos lo que hace:

- (Aplazamos todavía los detalles de qué significan "public", "class", "static" y "void").
- Main() indica donde comienza el cuerpo del programa, que se delimita entre llaves: { y }
- int primerNumero; reserva espacio para guardar un número entero, al que llamaremos primerNumero.
- int segundoNumero; reserva espacio para guardar otro número entero, al que llamaremos segundoNumero.
- int suma; reserva espacio para guardar un tercer número entero, al que llamaremos suma.
- primerNumero = 234; da el valor del primer número que queremos sumar
- segundoNumero = 567; da el valor del segundo número que queremos sumar
- suma = primerNumero + segundoNumero; halla la suma de esos dos números y la guarda en otra variable, en vez de mostrarla directamente en pantalla.
- System.Console.WriteLine("La suma de {0} y {1} es {2}", primerNumero, segundoNumero, suma); muestra en pantalla el texto y los valores de las tres variables (los dos números iniciales y su suma).

El resultado de este programa sería:

```
La suma de 234 y 567 es 801
```

Ejercicios propuestos:

Ejercicio propuesto 1.6.3.1: *Crea un programa que calcule el producto de los números 121 y 132, usando variables.*

Ejercicio propuesto 1.6.3.2: *Crea un programa que calcule la suma de 285 y 1396, usando variables.*

Ejercicio propuesto 1.6.3.3: *Crea un programa que calcule el resto de dividir 3784 entre 16, usando variables.*

Ejercicio propuesto 1.6.3.4: *Amplía el ejercicio 1.6.2.1, para que se muestre el resultado de la operación n1+n2*n3.*

1.7. Identificadores

Los nombres de variables (lo que se conoce como "**identificadores**") pueden estar formados por letras, números o el símbolo de subrayado (_) y deben comenzar por letra o subrayado. No deben tener espacios intermedios. También hay que recordar que las vocales acentuadas y la eñe son problemáticas, porque no son letras "estándar" en todos los idiomas, así que no se pueden utilizar como parte de un identificador en la mayoría de lenguajes de programación.

Por eso, no son nombres de variable válidos:

```
1numero         (empieza por número)
un número       (contiene un espacio)
Año1            (tiene una eñe)
MásDatos        (tiene una vocal acentuada)
```

(**Nota**: algunos entornos de programación modernos sí permitirán variables que contengan eñe y vocales acentuadas, pero como no es lo habitual en todos los lenguajes de programación, durante este curso introductorio nosotros no consideraremos válido un nombre de variable como "año", aun sabiendo que si estamos programando en C# con Visual Studio, el sistema sí lo consideraría aceptable).

Tampoco podremos usar como identificadores las **palabras reservadas** de C#. Por ejemplo, la palabra "int" se refiere a que cierta variable guardará un número entero, así que esa palabra "int" no la podremos usar tampoco como nombre de variable (pero no vamos a incluir ahora una lista de palabras reservadas de C#, ya nos iremos encontrando con ellas).

Hay que recordar que en C# las mayúsculas y minúsculas se consideran diferentes, de modo que si intentamos hacer

```
PrimerNumero = 0;
primernumero = 0;
```

o cualquier variación similar, el compilador protestará y nos dirá que no conoce esa variable, porque la habíamos declarado como

```
int primerNumero;
```

Ejercicios propuestos:

Ejercicio propuesto 1.7.1: *Crea un programa que calcule el producto de los números 87 y 94, usando variables llamadas "numero1" y "numero2".*

Ejercicio propuesto 1.7.2: *Intenta crear una nueva versión del programa que calcula el producto de los números 87 y 94, usando esta vez variables llamadas "1numero" y "2numero".*

Ejercicio propuesto 1.7.3: *Intenta crear una nueva versión del programa que calcula el producto de los números 87 y 94, usando esta vez variables llamadas "numero 1" y "numero 2".*

Ejercicio propuesto 1.7.4: *Crea una nueva versión del programa que calcula el producto de los números 87 y 94, usando esta vez variables llamadas "número1" y "número2".*

1.8. Comentarios

Podemos escribir comentarios, que el compilador ignorará, pero que pueden ser útiles para nosotros mismos, haciendo que sea más fácil recordar el cometido un fragmento del programa más adelante, cuando tengamos que ampliarlo o corregirlo.

Existen dos formas de indicar comentarios. En su forma más general, los escribiremos entre /* y */:

```
int suma;  /* Guardaré el valor para usarlo más tarde */
```

Es conveniente escribir comentarios que aclaren la misión de las partes de nuestros programas que puedan resultar menos claras a simple vista. Incluso suele ser aconsejable que el programa comience con un comentario, que nos recuerde qué hace el programa sin que necesitemos mirarlo de arriba a abajo. Un ejemplo casi exagerado podría ser:

```
/* ---- Ejemplo en C#: sumar dos números prefijados_____*/

public class Ejemplo_01_08a
{
    public static void Main()
    {
        int primerNumero = 234;
        int segundoNumero = 567;
        int suma;  /* Guardaré el valor para usarlo más tarde */

        /* Primero calculo la suma */
        suma = primerNumero + segundoNumero;

        /* Y después muestro su valor */
        System.Console.WriteLine("La suma de {0} y {1} es {2}", primerNumero,
        segundoNumero, suma);
    }
}
```

Un comentario puede empezar en una línea y terminar en otra distinta, así:

```
      /*  Esto
          es un
          comentario que
          ocupa más de
          una línea
      */
```

También es posible declarar otro tipo de comentarios, que comienzan con doble barra y terminan cuando se acaba la línea (estos comentarios, claramente, no podrán ocupar más

de una línea). Son los "comentarios de una línea" o "comentarios al estilo de C++" (a diferencia de los "comentarios de múltiples líneas" o "comentarios al estilo de C" que ya hemos visto):

```
//  Este es un comentario "al estilo C++"
```

De modo que el programa anterior se podría reescribir usando comentarios de una línea:

```
// ---- Ejemplo en C#: sumar dos números prefijados ----

public class Ejemplo_01_08b
{
    public static void Main()
    {
        int primerNumero = 234;
        int segundoNumero = 567;
        int suma;  // Guardaré el valor para usarlo más tarde


        // Primero calculo la suma
        suma = primerNumero + segundoNumero;

        // Y después muestro su valor
        System.Console.WriteLine("La suma de {0} y {1} es {2}",
            primerNumero, segundoNumero, suma);
    }
}
```

En este texto, a partir de ahora los “fuentes” comenzarán con un comentario que resuma su cometido, y en ocasiones incluirán también comentarios intermedios.

Ejercicios propuestos:

Ejercicio propuesto 1.8.1: *Crea un programa que convierta una cantidad prefijada de metros (por ejemplo, 3000) a millas. La equivalencia es 1 milla = 1609 metros. Usa comentarios donde te parezca adecuado.*

1.9. Datos por el usuario: ReadLine

Hasta ahora hemos tenido datos prefijados, pero eso es poco frecuente en el mundo real. Es mucho más habitual que los datos los introduzca el usuario, o que se lean desde un fichero, o desde una base de datos, o se reciban de Internet o cualquier otra red. El primer caso que veremos será el de interaccionar directamente con el usuario.

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

```
texto = System.Console.ReadLine();
```

pero eso ocurrirá un poco más adelante, cuando veamos cómo manejar textos. De momento, nosotros solo sabemos manipular números enteros, así que deberemos convertir ese dato a un número entero, usando Convert.ToInt32:

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

Un ejemplo de programa que sume dos números tecleados por el usuario sería:

```
// Ejemplo en C#: sumar dos números introducidos por el usuario

public class Ejemplo_01_09a
{
    public static void Main()
    {
        int primerNumero;
        int segundoNumero;
        int suma;


        System.Console.WriteLine("Introduce el primer número");
        primerNumero = System.Convert.ToInt32(System.Console.ReadLine());
        System.Console.WriteLine("Introduce el segundo número");
        segundoNumero = System.Convert.ToInt32(System.Console.ReadLine());
        suma = primerNumero + segundoNumero;

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

Ejercicios propuestos:

Ejercicio propuesto 1.9.1: *Crea un programa que calcule el producto de dos números introducidos por el usuario.*

Ejercicio propuesto 1.9.2: *Crea un programa que calcule la división de dos números introducidos por el usuario, así como el resto de esa división.*

Ejercicio propuesto 1.9.3: *Suma tres números tecleados por usuario.*

Ejercicio propuesto 1.9.4: *Pide al usuario una cantidad de "millas náuticas" y muestra la equivalencia en metros, usando: 1 milla náutica = 1852 metros.*

1.10.using System

Va siendo hora de hacer una pequeña mejora: no es necesario repetir "System." al principio de la mayoría de las órdenes que tienen que ver con el sistema (por ahora, las de consola y las de conversión), si al principio del programa utilizamos "using System":

```
// Ejemplo en C#: "using System" en vez de "System.Console"


using System;

public class Ejemplo_01_10a
{
    public static void Main()
    {
        int primerNumero;
        int segundoNumero;
        int suma;

        Console.WriteLine("Introduce el primer número");
        primerNumero = Convert.ToInt32(Console.ReadLine());
        Console.WriteLine("Introduce el segundo número");
        segundoNumero = Convert.ToInt32(Console.ReadLine());
        suma = primerNumero + segundoNumero;

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

Si además declaramos varias variables a la vez, como vimos en el apartado 1.5.2, el programa podría ser aún más compacto:

```
// Ejemplo en C#: "using System" en vez de "System.Console"

using System;

public class Ejemplo_01_10a
{
    public static void Main()
    {
        int primerNumero, segundoNumero, suma;


        Console.WriteLine("Introduce el primer número");
        primerNumero = Convert.ToInt32(Console.ReadLine());
        Console.WriteLine("Introduce el segundo número");
        segundoNumero = Convert.ToInt32(Console.ReadLine());
        suma = primerNumero + segundoNumero;

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

Ejercicios propuestos:

Ejercicio propuesto 1.10.1: *Crea una nueva versión del programa que calcula el producto de dos números introducidos por el usuario (1.9.1), empleando "using System". El programa deberá contener un comentario al principio, que recuerde cuál es su objetivo.*

Ejercicio propuesto 1.10.2: *Crea una nueva versión del programa que calcula la división de dos números introducidos por el usuario, así como el resto de esa división (1.9.2), empleando "using System". Deberás incluir un comentario con tu nombre y la fecha en que has realizado el programa.*

1.11. Escribir sin avanzar de línea

En el apartado 1.6.3 vimos cómo usar {0} para escribir en una misma línea datos calculados y textos prefijados. Pero hay otra alternativa, que además nos permite también escribir un texto y pedir un dato a continuación, en la misma línea de pantalla: emplear "Write" en vez de "WriteLine", así:

```
// Ejemplo en C#: escribir sin avanzar de línea

using System;

public class Ejemplo_01_11a
{
    public static void Main()
    {
        int primerNumero, segundoNumero, suma;


        Console.Write("Introduce el primer número: ");
        primerNumero = Convert.ToInt32(Console.ReadLine());
        Console.Write("Introduce el segundo número: ");
        segundoNumero = Convert.ToInt32(Console.ReadLine());
        suma = primerNumero + segundoNumero;

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

Incluso el último "WriteLine" de varios datos se podría convertir en varios Write (aunque generalmente eso hará el programa más largo y no necesariamente más legible), así

```
// Ejemplo en C#: escribir sin avanzar de línea (2)

using System;

public class Ejemplo_01_11b
{
    public static void Main()
    {
        int primerNumero, segundoNumero, suma;

        Console.Write("Introduce el primer número: ");
        primerNumero = Convert.ToInt32(Console.ReadLine());
        Console.Write("Introduce el segundo número: ");
        segundoNumero = Convert.ToInt32(Console.ReadLine());
        suma = primerNumero + segundoNumero;
        Console.Write("La suma de ");
        Console.Write(primerNumero);
        Console.Write(" y ");
        Console.Write(segundoNumero);
        Console.Write(" es ");
        Console.WriteLine(suma);
    }
}
```

Ejercicios propuestos:

Ejercicio propuesto 1.11.1: *El usuario tecleará dos números (a y b), y el programa mostrará el resultado de la operación (a+b)*(a-b) y el resultado de la operación a^2-b^2. Ambos resultados se deben mostrar en la misma línea.*
Ejercicio propuesto 1.11.2: *Pedir al usuario un número y mostrar su tabla de multiplicar, usando {0}, {1} y {2}. Por ejemplo, si el número es el 3, debería escribirse algo como*

3 x 0 = 0
3 x 1 = 3
…
3 x 10 = 30

Ejercicio propuesto 1.11.3: *Crea una variante del programa anterior, que pide al usuario un número y muestra su tabla de multiplicar. Esta vez no deberás utilizar {0}, {1}, {2}, sino "Write".*
Ejercicio propuesto 1.11.4: *Crea un programa que convierta de grados Celsius (centígrados) a Kelvin y a Fahrenheit: pedirá al usuario la cantidad de grados centígrados y usará los siguientes tablas de conversión: kelvin = celsius + 273 ; fahrenheit = celsius x 18 / 10 + 32. Emplea "Write" en vez de "{0}" cuando debas mostrar varios datos en la misma línea.*