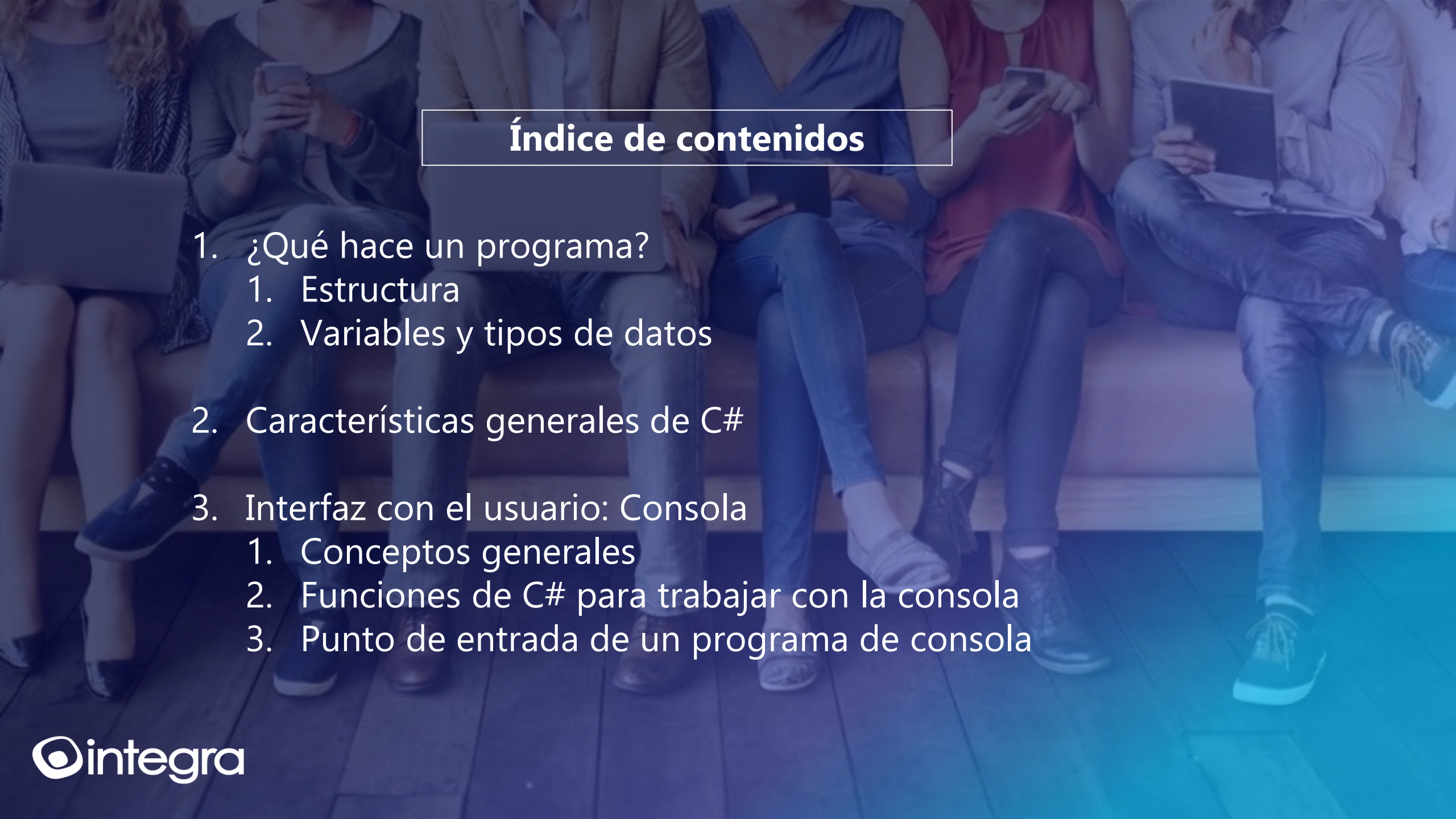
Módulo 1

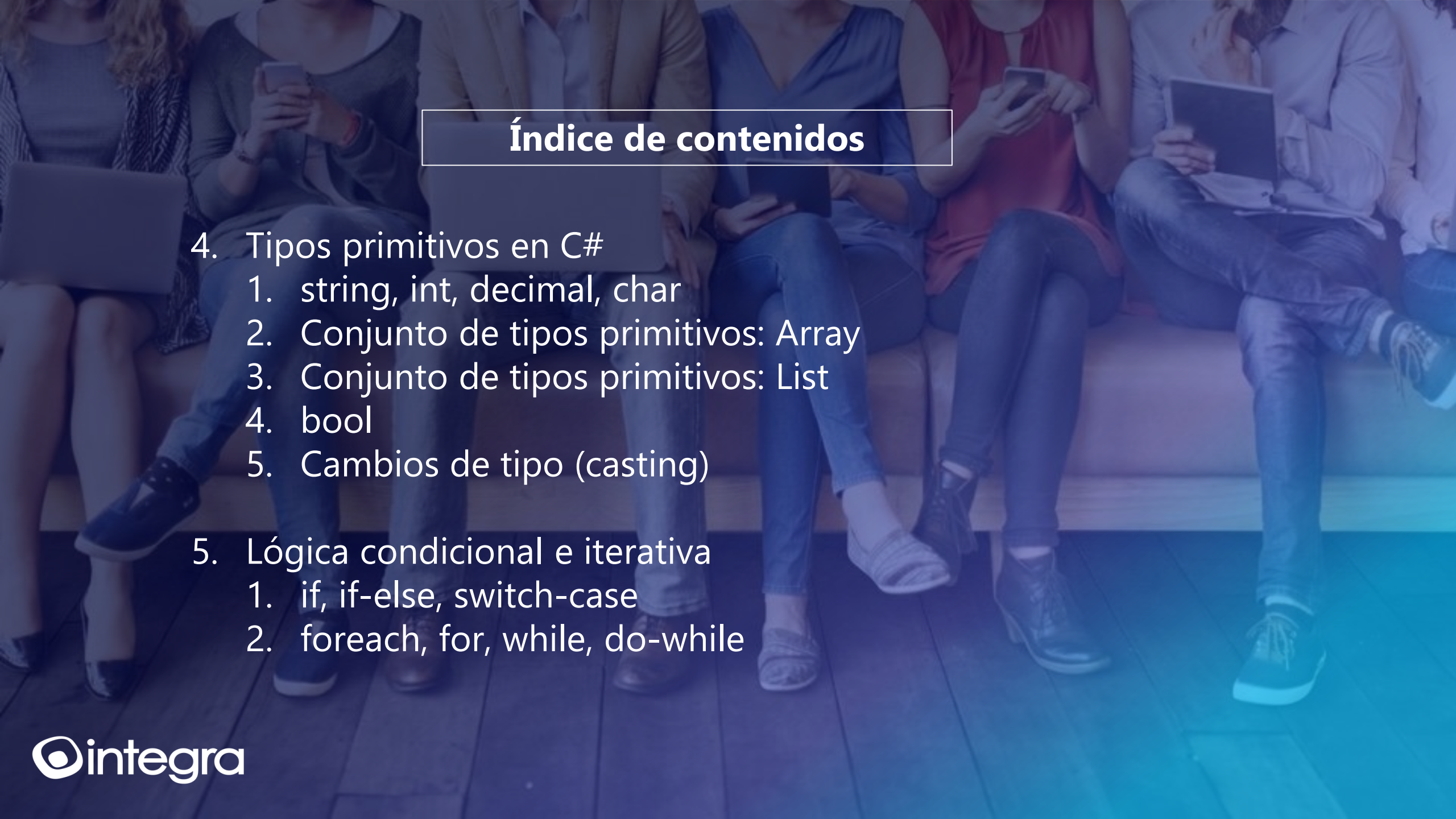
El lenguaje C#

Tecnara – Cursos de formación



Índice de contenidos

integra



Índice de contenidos

integra

01

¿Qué hace un programa?

¿Qué hace un programa?

- Un programa es una ‘receta’, una secuencia de instrucciones que permite resolver un problema informático, planteado en un enunciado
- El enunciado debe plantear un problema que se pueda solucionar usando las operaciones que proporciona el lenguaje de programación que vayamos a utilizar

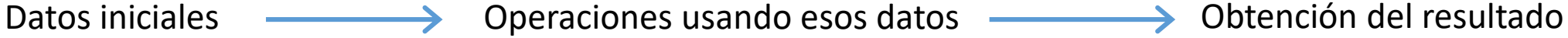
¿Qué hace un programa?

- Antes de poder asegurar que un enunciado plantea un problema que se puede solucionar mediante un programa escrito en un determinado lenguaje de programación, es necesario conocer las operaciones disponibles para los tipos de datos con los que permite trabajar dicho lenguaje

- Además de eso, para que el programa solucione el problema, TODA la información que el programa vaya a necesitar para resolver el problema debe existir y estar disponible para que el programa la obtenga cuando la necesite.
  - Enunciados que no se pueden resolver con un programa:
    - ¿Cómo encontrar la felicidad?, ¿Cuándo caerá un rayo delante de mi casa?
  - Enunciados que sí se pueden resolver con un programa:
    - calcula el área de un círculo a partir de su radio, ¿Cuándo volverá a pasar el Cometa Halley?

¿Qué hace un programa? Estructura

- Estructura general de un programa:



¿Qué hace un programa? Estructura

- Estructura general de un programa:

Datos iniciales → Operaciones usando esos datos → Obtención del resultado

Datos iniciales:

- Puede que no haga falta ninguno (en el programa HelloWorld, por ejemplo, no hacen falta)
- Puede que haya que obtenerlos de algún almacén EXTERIOR al programa (Ficheros, Base de datos…)
- Puede que haya que pedírselos al usuario que está ejecutando el programa (elegir una opción, etc)
- En cualquier caso, una vez obtenidos (del exterior o del usuario) hay que guardarlos en una variable

¿Qué hace un programa? Estructura

- Estructura general de un programa:

Datos iniciales → **Operaciones usando esos datos** → Obtención del resultado

Operaciones con datos:

- Solo estarán disponibles las que permita el lenguaje de programación utilizado (veremos unas cuantas de C#)

- Cada operación individual se puede entender como un mini-programa, con sus:
  - Datos iniciales: si hacen falta (puede que no), tienen que estar guardados en variables*
  - Resultado obtenido: si hace falta (puede que no), tiene que guardarse en alguna variable*
    - Hará falta si lo va a utilizar como dato inicial alguna operación posterior, o si es el resultado final

* Con el tiempo veremos que esto no es obligatorio, pero sí muy conveniente durante el aprendizaje para entender mejor los programas

¿Qué hace un programa? Estructura

- Estructura general de un programa:

Datos iniciales → Operaciones usando esos datos → **Obtención del resultado**

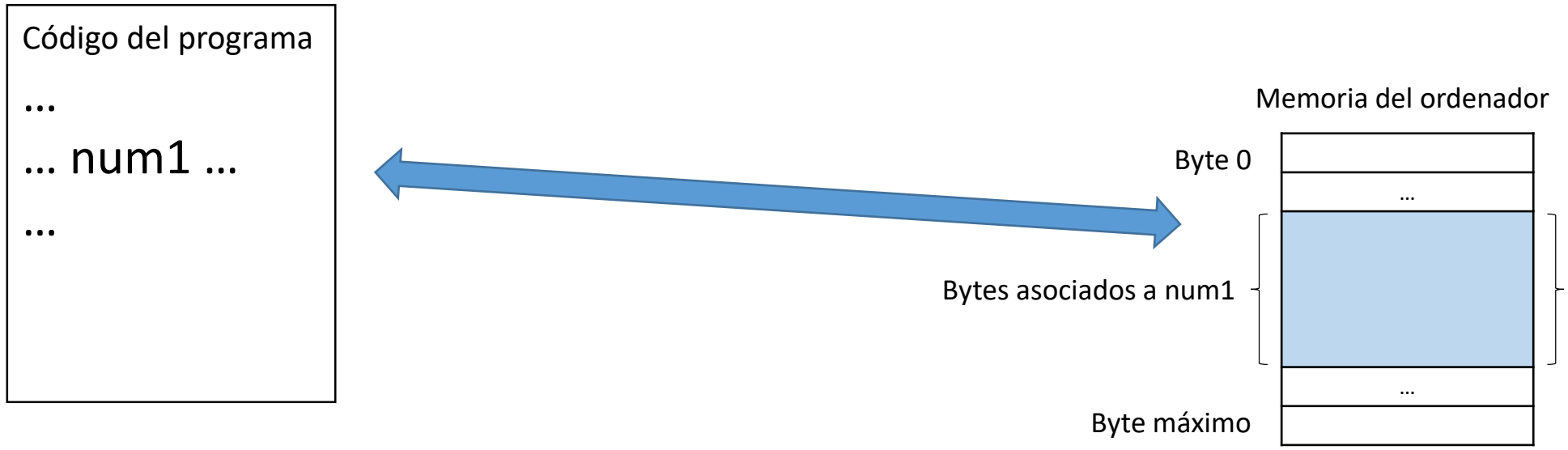
Resultado deseado:

- Puede que haya que guardarlo en alguna variable para usarlo más adelante
- Puede que haya que guardarlo en algún almacén EXTERIOR al programa para conservarlo aunque el programa termine
- Puede que haya que comunicárselo al usuario que está ejecutando el programa
- Puede que haya que hacer una mezcla de los tres casos anteriores

¿Qué hace un programa? Variable

Variable

- Trozo de memoria (contenedor de 1s y 0s, definido por: posición inicial + tamaño ocupado) asociado a un nombre y a un tipo de dato dentro de un programa

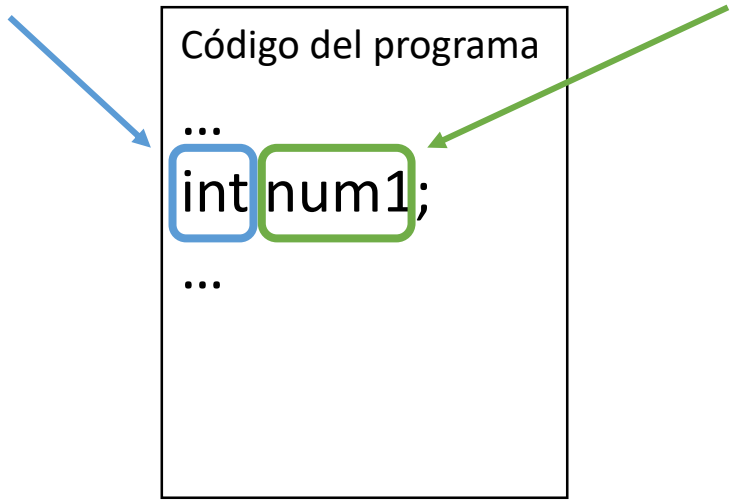


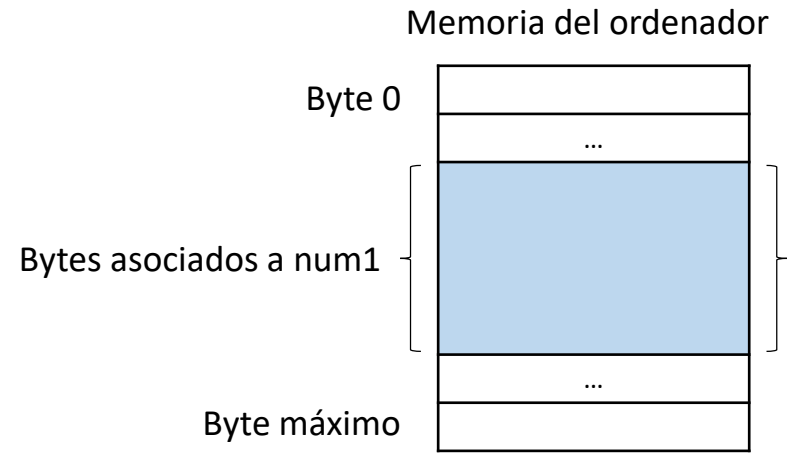
¿Qué hace un programa? Variable

Variable

- Para usar una variable, primero hay que declararla en el código
- Sintaxis de la declaración de una variable:

{nombre del tipo de dato} {nombre de la variable};





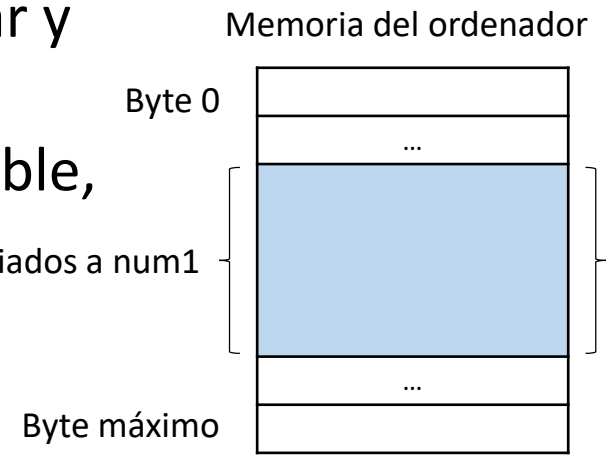
¿Qué hace un programa? Variable

Variable

- Para usar una variable, primero hay que declararla en el código
- Sintaxis de la declaración de una variable:

{nombre del tipo de dato} {nombre de la variable};

- El Sistema Operativo ya se encarga automáticamente de buscar y asignar la posición inicial del trozo de memoria en donde se guardarán los 1s y 0s que codifiquen el valor dado a la variable, por eso en la declaración no se menciona esa posición



¿Qué hace un programa? Variable

Variable: Tipo de dato

- Para usar una variable, primero hay que declararla en el código
- Sintaxis de la declaración de una variable:

{nombre del tipo de dato} {nombre de la variable};

- El tipo de dato define el número de 1s y 0s que ocupará la variable en memoria, y la forma de interpretar esos 1s y 0s

- Existen tipos de datos definidos en .NET Framework por defecto, y se pueden crear tipos personalizados si se desean

Nombre del tipo primitivo
byte
int
long
float
double
char
bool
object
string
decimal

¿Qué hace un programa? Variable

Variable: Tipo de dato

Tipos de datos posibles para declarar una variable:

- Tipos simples primitivos del lenguaje: int, decimal, char, bool, …
- Clases existentes de .NET Framework: string, DateTime, Regex, …
- Tipos enumerados
- Clases nuevas creadas específicamente para resolver un problema determinado
- (Otros: Interfaces, Delegates, Structs, var, …)

¿Qué hace un programa? Variable

Variable: Tipo de dato

Existen distintos tipos de datos que se pueden guardar en memoria:

- Por tanto, existen distintos tipos de variables con las que hacer operaciones en un programa
- Cada tipo tiene asociado un conjunto de valores representables
- Cada tipo tiene asociado un valor por defecto asignado en declaraciones sin inicialización
- Para cada valor, si hay que escribirlo en el programa (constante o literal), debe tener un formato determinado para que el compilador lo sepa traducir a los 1s y 0s correspondientes
- Además, cada valor que haya que mostrar por pantalla, según el tipo de dato se escribirá con un formato diferente
- Si el usuario tiene que escribir un dato por pantalla que el programa necesita que sea de un tipo determinado, el usuario deberá escribirlo con el formato que corresponda a ese tipo de dato, o el programa deberá traducirlo a ese formato antes de operar con él (ej: tipo decimal)

¿Qué hace un programa? Variable

Variable: Tipo de dato

Cada tipo de dato en C# se puede utilizar en un programa si se conoce lo siguiente:

- Formato de constante: cómo se escriben valores constantes del tipo en el código
  - Formatos de constantes leídas de pantalla: pueden ser diferentes (decimal, comas y puntos)
- Sintaxis de declaración: cómo crear variables del tipo sin darles un valor inicial personalizado
- Valor por defecto: valor inicial no personalizado dado a variables declaradas sin inicializar
- Declaración + inicialización: cómo declarar y a la vez dar valor inicial personalizado a una variable
- Leer valor de variable: dónde la variable se sustituye por su valor actual en memoria
- Escribir valor de variable: dónde la variable está modificando su valor en memoria

- Lista de **Operadores y funciones** existentes para trabajar con variables del tipo

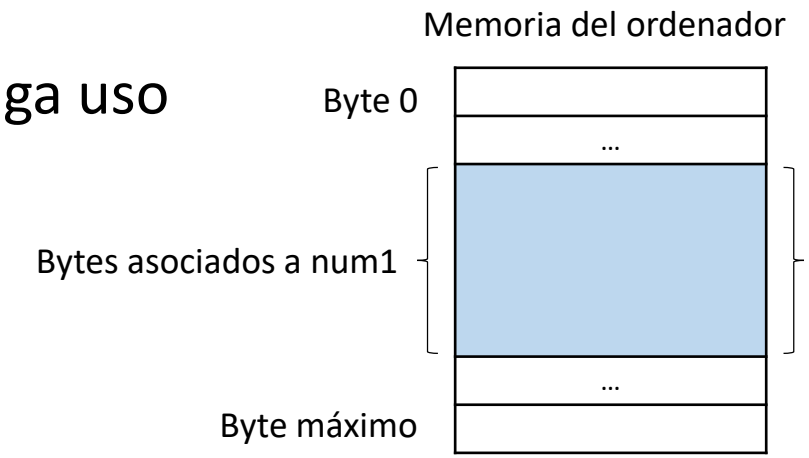
¿Qué hace un programa? Variable

Variable: Nombre asociado

- Para usar una variable, primero hay que declararla en el código
- Sintaxis de la declaración de una variable:

{nombre del tipo de dato} {nombre de la variable};

- El nombre sirve para identificar unívocamente esa posición de memoria y no otra mientras la variable tenga uso



¿Qué hace un programa? Variable

Variable : Nombre asociado

Nombres válidos (normas léxicas) para declarar una variable

- Deben comenzar por alguno de los siguientes caracteres:
  - un carácter alfabético (letra mayúscula o minúscula, solo alfabeto anglosajón)
  - un carácter barra baja, underscore en inglés (‘_’)
- Después del primer carácter, pueden contener letras (a-z, A-Z), dígitos (0-9) o ‘_’ en cualquier orden y repetidas tantas veces como se desee hasta la longitud máxima
- La longitud máxima admitida para el nombre de una variable es 511 caracteres

  - Ejemplos: num1, MaxLongitud, _name, day_of_week, _A03_456F26

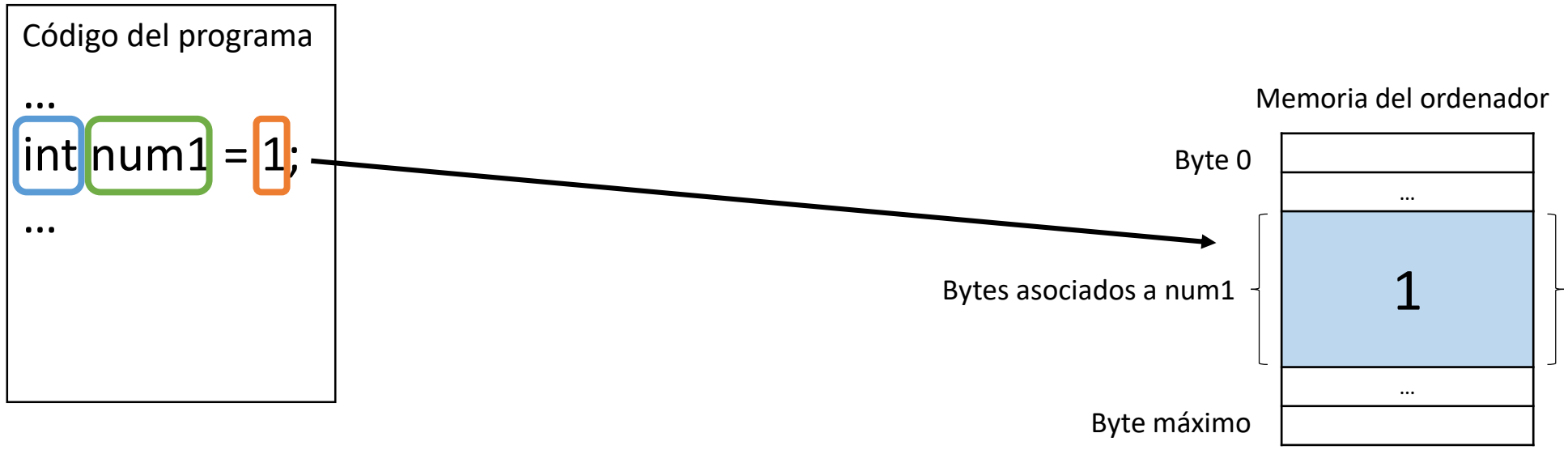
¿Qué hace un programa? Variable

Variable: Valor inicial

Cuando se declara una variable, se le puede dar un valor inicial (declaración + inicialización)

- Sintaxis de declaración + inicialización de una variable

{nombre del tipo de dato} {nombre de la variable} = {valor inicial de la variable};



¿Qué hace un programa? Variable

Variable : Valor inicial

Si se le da un valor inicial que pertenece a un tipo de dato distinto del declarado, que no se puede convertir al tipo de dato declarado, el programa contendrá un error de compilación, y el corrector de Visual Studio avisará del error subrayando en rojo el problema

Código del programa

```
...
int num1 = "a";
...
```

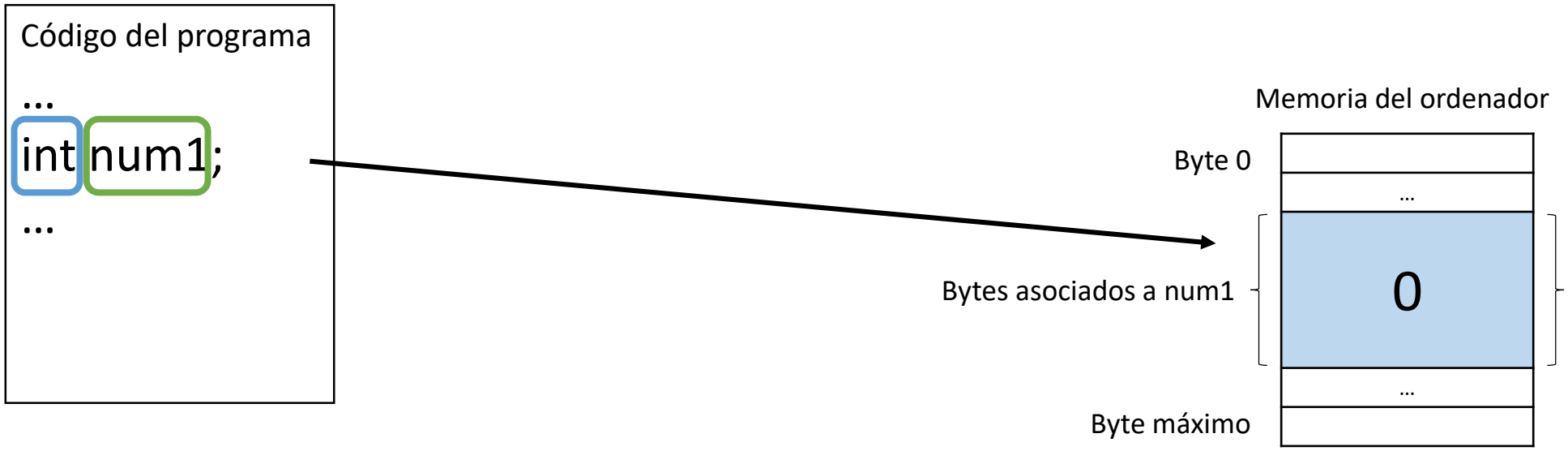
ERROR DE COMPILACIÓN

¿Qué hace un programa? Variable

Variable: Valor inicial

Si no se le da un valor inicial explícito, C# le dará un valor inicial por defecto

- El valor por defecto es distinto para cada tipo de dato. Para int es el valor 0



¿Qué hace un programa? Variable

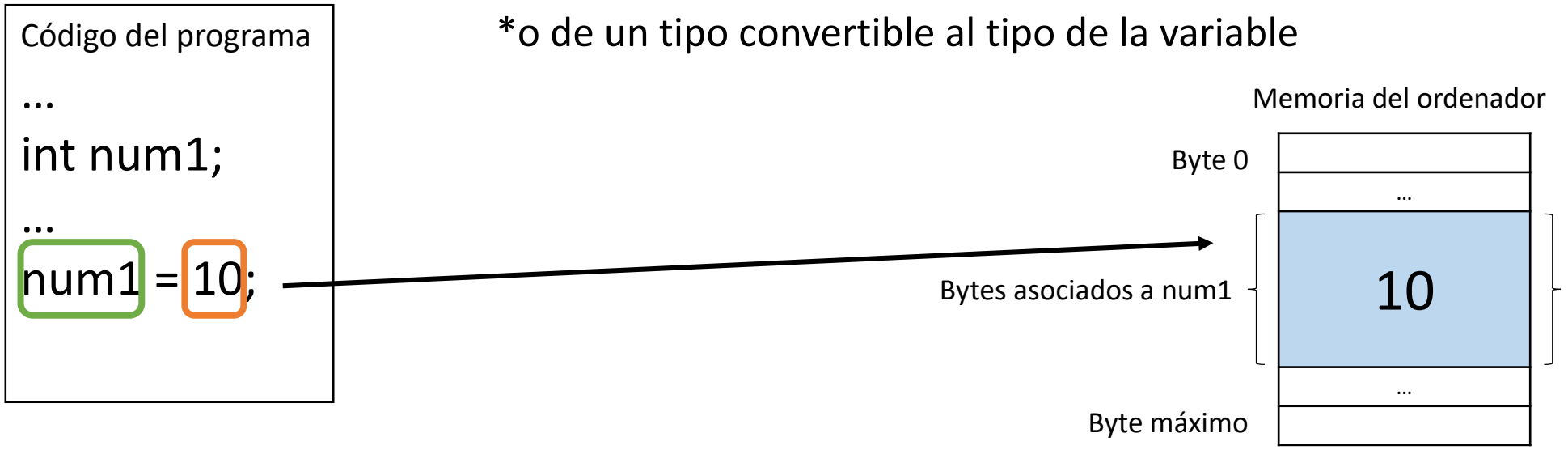
Variable: Asignación de valor

Tras la declaración, se puede cambiar cuando se desee (asignación) el valor de la variable

- Sintaxis de asignación de una variable

{nombre de la variable} = {expresión que devuelve valor del mismo tipo que la variable*};

*o de un tipo convertible al tipo de la variable



¿Qué hace un programa? Variable

Variable: Asignación de valor

Tras la declaración, se puede cambiar cuando se desee (asignación) el valor de la variable

- Sintaxis de asignación de una variable

{nombre de la variable} = {expresión que devuelva el mismo tipo de dato que la variable};

*o un tipo convertible al tipo de dato de la variable

Una expresión puede ser uno de los siguientes elementos:

- constante, tal vez cambiada de tipo con casting explícito. Ejemplos para int: -1, (int) 'a'
- otra variable, tal vez cambiada de tipo con casting explícito. Ejemplo para int: (int) float1
- operación con uno o varios operandos. Ejemplos para int: 3 + 5, num1 - 6, alto * ancho
- invocación a una función que devuelva resultado. Ejemplo para int: Max(num1, num2)

¿Qué hace un programa? Variable

Variable: Asignación de valor

Tras la declaración, se puede cambiar cuando se desee (asignación) el valor de la variable

- Sintaxis de asignación de una variable

{nombre de la variable} = {expresión que devuelva el mismo tipo de dato que la variable};

*o un tipo convertible al tipo de dato de la variable

Una expresión puede ser una mezcla de los elementos anteriores:

- En una operación, cada operando puede ser a su vez una expresión completa:
  3 + Max(num1, num2), 6 – num1 / num2 * num3 + num4

¿Qué hace un programa? Variable

Variable: Asignación de valor

Tras la declaración, se puede cambiar cuando se desee (asignación) el valor de la variable

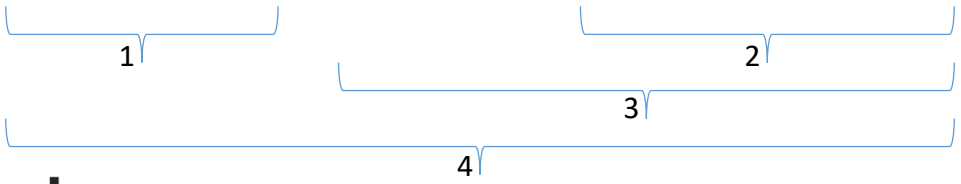
- Sintaxis de asignación de una variable

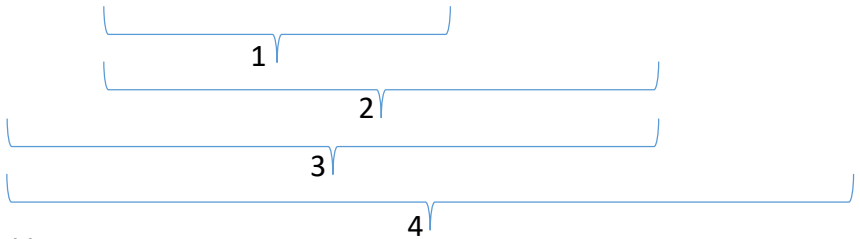
{nombre de la variable} = {expresión que devuelva el mismo tipo de dato que la variable};

*o un tipo convertible al tipo de dato de la variable

Cuando se juntan varias operaciones, unas se ejecutan antes que otras por defecto (precedencia de operadores). Para personalizar el orden de ejecución, se usan paréntesis:

(6 – num1) / (num2 * (num3 + num4)) **VS** 6 – num1 / num2 * num3 + num





¿Qué hace un programa? Variable

Variable: Asignación de valor

Tras la declaración, se puede cambiar cuando se desee (asignación) el valor de la variable

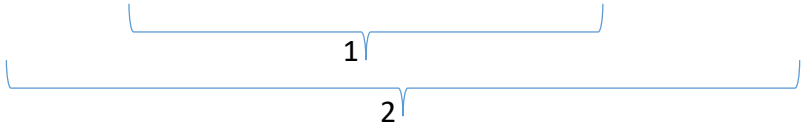
- Sintaxis de asignación de una variable

{nombre de la variable} = {expresión que devuelva el mismo tipo de dato que la variable};

*o un tipo convertible al tipo de dato de la variable

En una invocación, cada parámetro puede ser una expresión, incluso otra invocación (funciones anidadas):

```
Max(Min(num1, num2), num3)
```

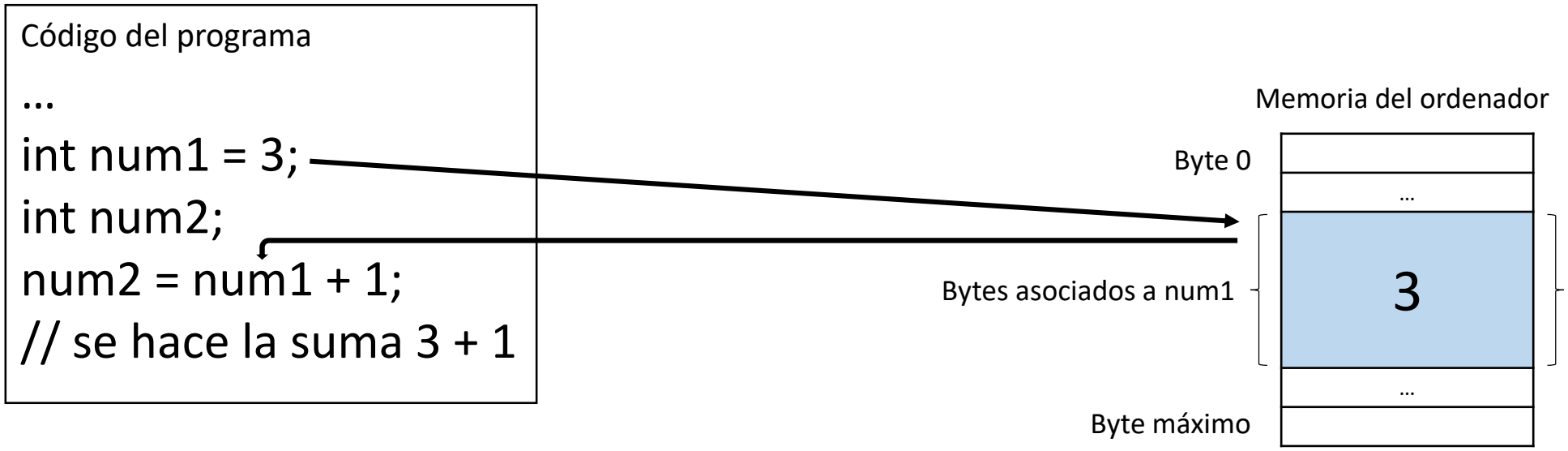


¿Qué hace un programa? Variable

Variable: lectura de valor

Una vez declarada, cada vez que aparezca en el programa el nombre de la variable sin estar a la derecha de un "=" (sin ser una asignación), será para consultar (leer) su valor

- Cuando esto ocurre, su nombre se sustituye por su valor actual guardado en memoria



En resumen...

Un programa que se ejecuta:

- Trabaja con datos asignados a variables de distintos tipos
- Puede obtener alguno de esos datos del usuario y/o de un almacén externo
- Usa esos datos para operar con ellos y generar uno o varios resultados
- Puede mostrar resultados al usuario, o guardarlos en un almacén externo
  - ¡IMPORTANTE! cuando el programa acaba, sus variables desaparecen (pierden sus valores guardados)
  - Todos los resultados no mostrados por pantalla o guardados en un almacén externo se perderán
- Cada tipo de dato tiene definidos operadores y funciones para realizar operaciones
  - Existe la posibilidad de definir nuevos tipos de datos y nuevas operaciones, si lo que hay ya hecho no es suficiente

02

Características generales de C#

Características generales de C#

C# es un lenguaje sensible a mayúsculas y minúsculas

- Los nombres de variables ‘opción’ y ‘Opcion’ se refieren a variables diferentes
- Los nombres de funciones ‘miFuncion()’ y ‘MiFuncion()’ se refieren a funciones diferentes
- ¡¡Recordad que al llamar a funciones de las bibliotecas de .NET es necesario que mayúsculas y minúsculas coincidan exactamente con las de la función original!!
  - System.Console.WriteLine(“¡hola mundo!”); → OK
  - System.Console.writeLine(“¡hola mundo!”); → KO (error, vamos…)

Características generales de C#

Se pueden escribir en una misma línea varias instrucciones (incluso una función entera)

- Pero para aprender, no se recomienda, a menos que se tenga muy claro TODO lo que se va a ejecutar a la vez en una línea que incluya varias instrucciones

```
void Main() {
        System.Console.WriteLine("Hola");
}

void Main() { System.Console.WriteLine("Hola"); }
```

Características generales de C#

- Comentarios de una línea

```
//Obtiene los datos del usuario
```

- Comentarios de varias líneas

```
/*Respuesta.Type = ResponseSaveDTO.ResponseTypeEnum.Completada;
//Genera ticket
Security.GenerateTicket(UserDTO, System.Web.HttpContext.Current, FormsAuthentication.FormsCookieName,
(int)FormsAuthentication.Timeout.TotalMinutes);  */
```

- Los comentarios tienen dos usos principales:
  - Documentación: describir lo que hace el programa en ese punto para entenderlo mejor
  - Hacer que una o varias líneas de código NO se ejecuten hasta des-comentarlas

Características generales de C#

- Comentar líneas con Visual Studio

Para comentar una o varias líneas seleccionadas:

- atajo de teclado: CTRL+K, CTRL+C
- Botón en barra superior:

Para descomentar una o varias líneas seleccionadas que estén comentadas:

- atajo de teclado: CTRL+K, CTRL+U
- Botón en barra superior:

Características generales de C#

Todas las instrucciones terminan en ; (carácter punto y coma)

- Declaración de variables → int variableEntera**;** // inicializado por defecto con valor 0
- Declaración+inicialización de variables → int variableEntera = 1**;**
- Asignación de variables → variableEntera = 5**;**
- Llamada/Invocación a funciones → CalcularSuma(valor1, valor2)**;**

Características generales de C#

Se emplean los caracteres { y } para definir dónde empieza y dónde acaba el código de funciones o bloques (más adelante se verán bloques condicionales e iterativos)

```
void Main() {
        System.Console.WriteLine("Hola");
}
```

```
if (1 <= num && num <= 100)
{
    System.Console.WriteLine("El número que ha introducido es " + num);
    finish = true;
}
```

tecnara
Cluster TIC de Aragón / Aragón IT Cluster

03

Interfaz con el usuario: Consola

integra

¿Qué hace un programa? Interfaz

Interfaz

- Del inglés interface: inter=entre, face=superficie (no es la acepción principal)
- (en informática) → Conjunto de materiales, circuitos físicos y protocolos informáticos que permiten a un sistema comunicarse con otro sistema
  - Comunicación usuario-usuario: chat de Facebook
  - Comunicación usuario-máquina: puertas, botones e indicadores luminosos y sonoros de un ascensor
  - Comunicación máquina-máquina: sistema sensor de movimiento ←→ sistema de alarma y bloqueo

¿Qué hace un programa? Interfaz

Interfaz de usuario (antes conocida como Interfaz Hombre-Máquina, ahora como Interfaz Persona-Ordenador)

Canal de comunicación que utiliza un programa para:

- Pedir datos al usuario / Obtener datos del usuario sin pedírselos (Entrada)
- Comunicar datos al usuario (Salida)

- Diferentes tipos de interfaz:
  - Consola, Escritorio, Web, API Web, Móvil, Videojuego
  - Otras: reconocimiento/síntesis de voz (Cortana, Alexa, Siri)
  - Otras más: sensores(eyetracker, EEG)/actuadores(alarmas, indicadores luminosos)

Interfaz con el usuario: Consola

En la primera parte del curso haremos programas que se ejecutan en interfaz de consola

```
Introduzca un número entre 1 y 100 (0 para salir): 1
El número que ha introducido es 1

Introduzca un número entre 1 y 100 (0 para salir): 5
El número que ha introducido es 5

Introduzca un número entre 1 y 100 (0 para salir): 99
El número que ha introducido es 99

Introduzca un número entre 1 y 100 (0 para salir): -3
El número que ha introducido no está dentro del rango [1-100]

Introduzca un número entre 1 y 100 (0 para salir): 101
El número que ha introducido no está dentro del rango [1-100]

Introduzca un número entre 1 y 100 (0 para salir): a
El valor que ha introducido no es un número

Introduzca un número entre 1 y 100 (0 para salir): 0
```

¿Qué es la consola?

Interfaz basada en una ventana de texto mostrada en la pantalla del ordenador:

- Un **programa** puede escribir texto en la consola para que el **usuario** lo lea y pueda tomar decisiones
- Un **usuario** puede escribir texto en la consola para que el **programa** lo lea y pueda usarlo si lo necesita

¿Qué es la consola?

En el ejemplo de la derecha:

- Texto escrito por el programa:
  - Color amarillo
- Texto escrito por el usuario:
  - Color blanco

```
Introduzca un número entre 1 y 100 (0 para salir): 1
El número que ha introducido es 1

Introduzca un número entre 1 y 100 (0 para salir): 5
El número que ha introducido es 5

Introduzca un número entre 1 y 100 (0 para salir): 99
El número que ha introducido es 99

Introduzca un número entre 1 y 100 (0 para salir): -3
El número que ha introducido no está dentro del rango [1-100]

Introduzca un número entre 1 y 100 (0 para salir): 101
El número que ha introducido no está dentro del rango [1-100]

Introduzca un número entre 1 y 100 (0 para salir): a
El valor que ha introducido no es un número

Introduzca un número entre 1 y 100 (0 para salir): 0
```

Programa escribe en consola

Un programa puede escribir texto en la consola para:

- Mostrar un texto fijo

```
Hello, world!
```

- Solicitar al usuario que introduzca un valor desde teclado
  - También es un texto fijo, pero tiene la intención de informar al usuario de lo que el programa espera que haga

```
Introduzca un número entre 1 y 100 (0 para salir):
```

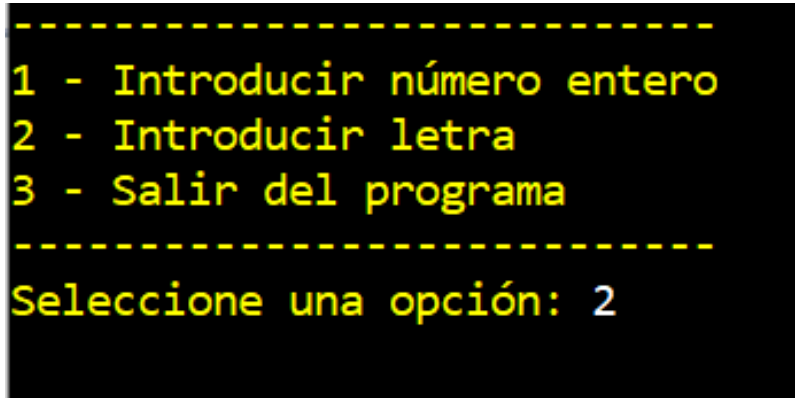
- Mostrar una respuesta tras haber procesado valores introducidos anteriormente
  - Suele ser una mezcla de texto fijo y valores calculados

```
El número que ha introducido es 3
```

Programa escribe en consola

Un programa puede escribir texto en la consola para:

- En programas con distintas opciones, mostrar un menú explicando lo que hace cada opción (descripción corta) antes de solicitar al usuario que introduzca la opción deseada



Tipo de dato para textos: string

Si la consola funciona solo mediante texto, está claro que el primer tipo de dato de C# que hay que conocer es el tipo creado para trabajar con textos: el tipo 'string'

- Ejemplos de constante → "Soy un texto", "123 es menor que 500", ""
- Declaración → string texto1;
- Declaración + inicialización → string texto1 = "Hola";
- Valor por defecto → null (Valor nulo NO es lo mismo que cadena vacía "")
- Leer valor de variable → string texto2 = texto1;
- Escribir valor de variable → texto1 = "Adiós";

Funciones para trabajar con la consola

Aunque más adelante se verán operadores y funciones más genéricos para trabajar con strings, en este punto nos interesan conocer las funciones que permiten:

- Escribir en la consola
- Leer de la consola

Funciones para trabajar con la consola

Para que el programa de consola pueda leer y escribir por pantalla es necesario llamar a funciones propias de .NET Framework, así que aunque las funciones se verán en detalle más adelante, aquí van unos pequeños adelantos teóricos para entender mejor lo que se tiene que escribir para llamarlas y lo que pasa mágicamente, sin que sepamos como, cuando se han llamado:

- Una llamada a una función se compone de un nombre seguido de un paréntesis “()”
- Puede tener parámetros de entrada (dentro del paréntesis, separados por comas)
- Puede devolver un resultado que se puede asignar a una variable
- En las funciones de consola que veremos, el nombre irá precedido de “System.Console.”

Ejemplo: string texto = System.Console.ReadLine();

Funciones para trabajar con la consola

Para que el programa de consola pueda leer y escribir por pantalla es necesario llamar a funciones propias de .NET Framework, así que aunque las funciones se verán en detalle más adelante, aquí van unos pequeños adelantos teóricos para entender mejor lo que se tiene que escribir para llamarlas y lo que pasa mágicamente, sin que sepamos como, cuando se han llamado:

- En las próximas diapositivas se muestra la **signatura** de distintas funciones de consola (**nombre**, **nº y tipo de sus parámetros de entrada** y **tipo de su resultado devuelto**)
- En la signatura, el tipo de dato del resultado devuelto aparece antes del nombre de la función
- Si una función no devuelve ningún tipo de resultado, se indica la palabra reservada **void**

Funciones para trabajar con la consola

Funciones de .NET Framework para que el programa pueda escribir en la consola:

- void **System.Console.WriteLine**(string value);
  // escribe en pantalla la cadena guardada en el parámetro value, seguida de un salto de línea.

- void **System.Console.WriteLine**();
  // escribe en pantalla un salto de línea.

- void **System.Console.Write**(string value);
  // escribe en pantalla la cadena guardada en el parámetro value, SIN AÑADIR DESPUÉS un salto de línea.

Funciones para trabajar con la consola

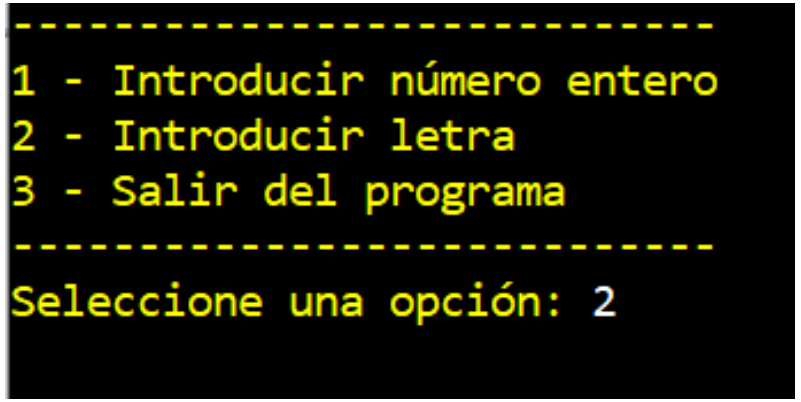
Programa 1: Hola Mundo. Probad a ejecutarlo paso a paso, con un breakpoint al principio

- Sin un Console.ReadKey() final, cuando el programa termina la última instrucción, la consola se cierra sola. Para que esto no ocurra, se añade una instrucción que bloquea el programa hasta que el usuario pulse cualquier tecla (ver más adelante apartado de funciones .NET para leer de consola)

Usuario escribe en consola

Un usuario puede escribir texto en la consola para:

- Seleccionar una opción entre varias posibles que ofrece el programa



- Comunicar al programa un valor de algún tipo (entero, decimal, carácter, texto, …)

Usuario escribe en consola

Un usuario puede escribir texto en la consola para:

- Pasar al programa una o varias líneas con un formato determinado que el programa sepa parsear (interpretar) para extraer los datos que necesita

```
Introduzca una lista de números enteros separados por comas.
(máximo 10) Ej: -1, 0, 4, 2000
-100, 34, w, 1234
- El 1º valor introducido (-100) se ha leído correctamente
- El 2º valor introducido (34) se ha leído correctamente
- ¡El 3º valor introducido no es un número entero!
- El 4º valor introducido (1234) se ha leído correctamente
```

Funciones para trabajar con la consola

Funciones de .NET Framework para que el programa pueda leer lo que el usuario escribe en la consola:

- ConsoleKeyInfo **System.Console.ReadKey**();
  /* Bloquea el programa hasta que usuario pulsa cualquier tecla. Después, devuelve un objeto con información sobre la tecla pulsada. No necesita esperar a que el usuario pulse ENTER para ejecutarse */

Funciones para trabajar con la consola

Funciones de .NET Framework para que el programa pueda leer lo que el usuario escribe en la consola:

- int **System.Console.Read**();
  /* Si no hay ningún carácter pendiente de leer, bloquea el programa hasta que el usuario pulsa ENTER. Si hay caracteres pendientes de leer, lee y devuelve el valor ASCII del siguiente carácter pendiente de leer */

- string **System.Console.ReadLine**();
  /* Bloquea el programa hasta que usuario pulsa ENTER. Después, devuelve la cadena que ha escrito el usuario antes del ENTER */

Funciones para trabajar con la consola

Funciones de .NET Framework para que el programa pueda leer lo que el usuario escribe en la consola:

- Lo habitual es usar la función ReadLine() para obtener toda la línea escrita por el usuario
- El problema de esta función es que devuelve un valor de tipo string, pensado para guardar textos…
- ¿Qué pasa si queremos usar lo que escriba el usuario como algo distinto de un texto: número entero, número decimal, fecha…?
- En esos casos, existen funciones que comprueban si un texto determinado contiene los caracteres correctos para entenderlo como un valor de otro tipo: “3” → 3 | “4,5” → 4.5
- Se verán cuando se expliquen esos tipos

Funciones para trabajar con la consola

Variables de .NET Framework relacionadas con el color de la consola:

- **System.Console.ForegroundColor**: variable interna para el color del texto (blanco por defecto)
  System.Console.ForegroundColor = ConsoleColor.Yellow;
  /* En adelante, todo lo que se escriba en la consola (tanto por el programa como por el usuario) se escribirá con caracteres en color amarillo. */

- **System.Console.BackgroundColor**: variable interna para el color del fondo (negro por defecto)
  System.Console.BackgroundColor = ConsoleColor.Blue;
  /* En adelante, todo lo que se escriba en la consola (tanto por el programa como por el usuario) se escribirá con caracteres con fondo azul. */

Funciones para trabajar con la consola

Funciones de .NET Framework relacionadas con el color de la consola:

- **System.Console.Clear**(): re-pinta la consola con el color de fondo actual y eliminando todo el texto. Los textos futuros tendrán el color de texto actual
  System.Console.ForegroundColor = ConsoleColor.Yellow;
  System.Console.BackgroundColor = ConsoleColor.Blue;
  System.Console.Clear(); // vacía consola con fondo azul. Futuros textos en amarillo

- **System.Console.ResetColor**(): deja los colores de texto y fondo con sus valores por defecto (blanco y negro, respectivamente)
  System.Console.ResetColor(); // no hace nada en pantalla, solo modifica variables internas
  System.Console.Clear(); // vacía consola con fondo negro. Futuros textos en blanco

Programa de consola: punto de entrada

Punto de entrada del programa de consola: clase Program, función static void Main() (sean lo que sean namespace, class y static…)

```
namespace Program
{
    class Program
    {
        static void Main(string[] args)
        {
            /* lo que se escriba aquí será lo primero
             * que se ejecute del programa de consola*/
        }
    }
}
```

04

Tipos primitivos en C#

Tipos primitivos en C#

En la siguiente sección veremos con algo de detalle distintos tipos de datos, e iremos haciendo programas cada vez un poco más complejos que usen lo que vayamos viendo

Vamos a conocer algunos tipos de datos primitivos de C# que permiten trabajar con:

- Textos o caracteres individuales
- Números enteros o números con decimales
- Valores lógicos booleanos (de verdadero/falso)

Tipos primitivos en C#

Nombre del tipo primitivo	Clase .NET asociada	Nº bits que ocupa	Valores representables
byte	Byte	8	0 a 255
int	Int32	32	-2.147.483.648 a 2.147.483.647
long	Int64	64	-922337203685477508 a 922337203685477507
float	Single	32	-3,402823e38 a 3,402823e38
double	Double	64	-1,79769313486232e308 a 1,79769313486232e308
char	Char	16	Símbolos Unicode utilizados en el texto
bool	Boolean	8	true o false
object	Object		
string	String		
decimal	Decimal	128	±1.0 × 10e−28 a ±7.9 × 10e28

Tipos primitivos en C#: string

Tipo de dato para guardar textos: string (sí, otra vez la misma diapositiva 😉)

- Ejemplos de constante → “Soy un texto”, “123 es menor que 500”, “”
- Declaración → string texto1;
- Declaración + inicialización → string texto1 = “Hola”;
- Valor por defecto → null (Valor nulo NO es lo mismo que cadena vacía “”)
- Leer valor de variable → string texto2 = texto1;
- Escribir valor de variable → texto1 = “Adiós”;

Tipos primitivos en C#: string

Operadores y funciones para trabajar con variables de tipo string:

- Concatenación (unión una detrás de otra) de cadenas de texto

```
string mensaje = “hola” + “mundo”; // aquí mensaje tendrá valor holamundo
```

- Otro ejemplo, mezclando literales (constantes) y variables

```
string str1 = “hola”;    string str2 = “mundo”;
string mensaje = “¡” + str1 + “ “ + str2 + “!”; // aquí mensaje tendrá valor ¡hola mundo!
```

Tipos primitivos en C#: string

Más operadores y funciones para trabajar con variables de tipo string :

- Otra manera de concatenar varios strings usando la función string.Format():

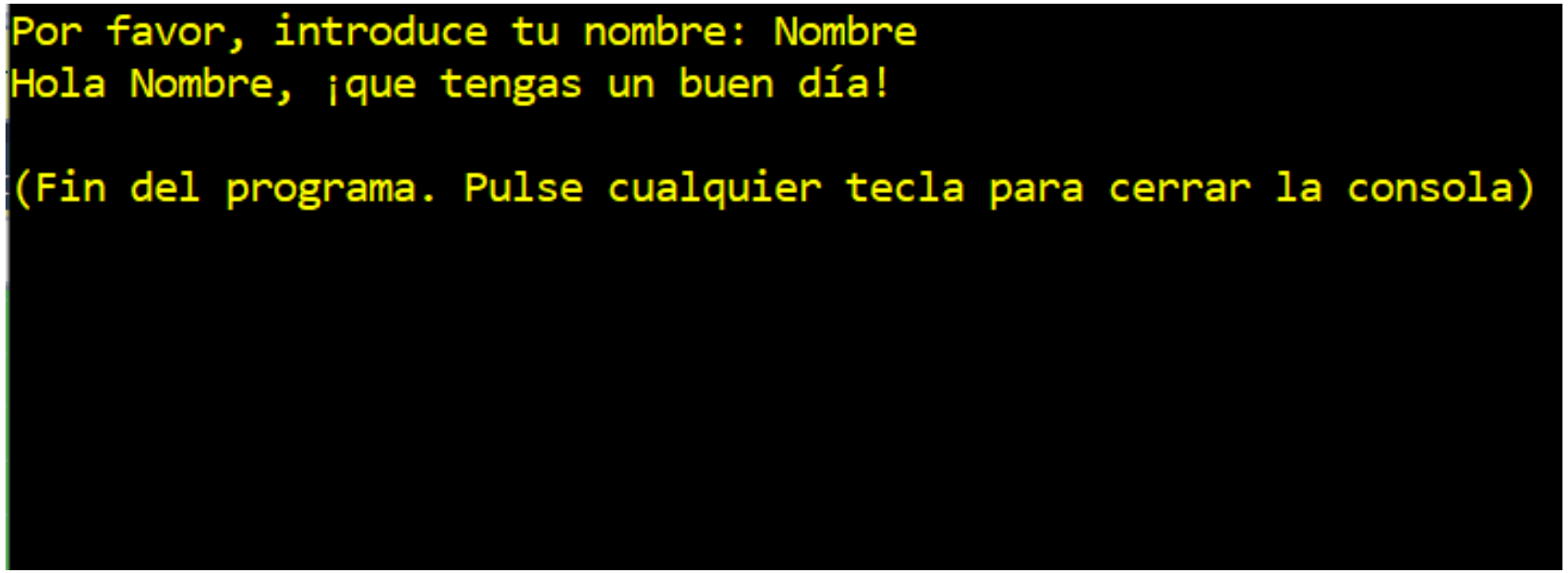
```
string str1 = “hola”;    string str2 = “mundo”;
string mensaje = String.Format(“¡{0} {1}!”, str1, str2); // aquí mensaje tendrá valor ¡hola mundo!
```

- Una tercera manera de concatenar strings incluyendo variables en un literal de tipo string

```
string str1 = “hola”;    string str2 = “mundo”;
string mensaje = $“¡{str1} {str2}!”; // aquí mensaje tendrá valor ¡hola mundo!
```

Tipos primitivos en C#: string

Programa 2: Saludo. Programa escribe texto en pantalla, usuario escribe texto en pantalla, programa lee texto, lo concatena con un string que ya tiene definido, y escribe por pantalla la concatenación



Tipos primitivos en C#: string

Más operadores y funciones para trabajar con variables de tipo string:

string cadena = “HOLA MUNDO”;

- Obtener longitud del string:

int size = cadena.Length; // aquí size tendrá valor 10

- Extraer un trozo del string completo, definido por un índice inicial* y una longitud

string trozo = cadena.Substring(1,2); // aquí trozo tendrá valor “OL”

*más adelante se ven el tipo char y los arrays. Un string es un array de chars, con índice inicial 0

Tipos primitivos en C#: string

Más operadores y funciones para trabajar con variables de tipo string:

string cadena = “HOLA MUNDO”;

- Sustituir en una cadena todas las apariciones de un carácter por otro:

string modificada = cadena.Replace(“O”, “8”); // aquí cadena tendrá valor “H8LA MUND8”

- Otro uso de Replace(), con el 2º parámetro igual a cadena vacía (“”): Eliminar carácter del string

string modificada = cadena.Replace(“O”, “”); // aquí cadena tendrá valor “HLA MUND”

// eliminaría todas las O de cadena. Eliminar todas las apariciones de un carácter en una cadena
// puede ser útil en algunos programas

Tipos primitivos en C#: string

Más operadores y funciones para trabajar con variables de tipo string:

Documentación oficial en español:
https://docs.microsoft.com/es-es/dotnet/api/system.string?view=net-5.0

Documentación oficial en inglés (sin errores debidos a mala traducción):
https://docs.microsoft.com/en-us/dotnet/api/system.string?view=net-5.0

Tipos primitivos en C#: string

Programa 3: Letras inicial y final. Programa pide al usuario que escriba una frase, programa la lee y muestra por pantalla el primer carácter **alfabético** y el último carácter **alfabético** de esa frase.

```
Por favor, introduce una frase: Hola. Estoy muy feliz de escribir aquí
- la primera letra de la frase es H
- la última letra de la frase es í

(Fin del programa. Pulse cualquier tecla para cerrar la consola)
```

Tipos primitivos en C#: string

Programa 3 (continuación del enunciado)

- Usar la función Substring() vista para obtener el primer y último carácter alfabético.

Mejora propuesta (no obligatoria):

- Usar la función Replace() con el 2º parámetro a “” para eliminar de la frase uno a uno todos los caracteres que no sean alfabéticos que creáis que pueden ser escritos en la frase (por ejemplo, en español: ‘¡’ y ‘!’).

- Ejecutad el programa escribiendo una frase que tenga caracteres no alfabéticos al principio y al final (alguno de los que hayáis decidido quitar con Replace, como ‘¡’ y ‘!’), y comprobad que se eliminan antes de empezar a extraer el primer y último carácter con Substring()

Tipos primitivos en C#: string

Programa 3 (continuación del enunciado)

- Usar la función Substring() vista para obtener el primer y último carácter alfabético.

Mejora propuesta 2 (no obligatoria):

- Cuando se vean en el curso las expresiones regulares (ER), volver a este programa y tratar de crear una ER para eliminar TODOS los caracteres que no sean alfabéticos.

- En lugar de quitar manualmente, uno a uno, los caracteres indeseados (lo cual es largo, tedioso, y no asegura que hayamos considerado todos los caracteres posibles), la ER considerará solo los caracteres que no se desean eliminar, y se eliminarán todos los demás, sean cuales sean. Así sí es seguro que se obtendrá el comportamiento deseado.

Tipos primitivos en C#: int

Tipo de dato para guardar números enteros (negativo, cero o positivo): int

- Ejemplos de constante → -100, 0, 100
- Declaración → int numero1;
- Declaración + inicialización → int numero1 = -38;
- Valor por defecto → 0
- Leer valor de variable → int otraVariableEntera = numero1;
- Escribir valor de variable → numero1 = 20;

Tipos primitivos en C#: int

Función de .NET Framework para comprobar si lo que el usuario escribe en la consola es o no un número entero:

- bool **int.TryParse**(string candidato, out int numero);
  - si el string candidato corresponde a un entero, devuelve true y asigna a la variable numero el valor numérico de candidato
  - En caso contrario, devuelve false y 'numero' contendrá el valor por defecto para int: 0

```
string numStr = System.Console.ReadLine();
int num;
bool isNumInt = int.TryParse(numStr, out num);
```

O de forma más compacta:

```
bool isNumInt = int.TryParse(System.Console.ReadLine(), out int num);
```

Tipos primitivos en C#: int

Operadores y funciones para trabajar con variables de tipo int:

- Suma de dos literales

```
int dos = 1 + 1; // aquí dos tendrá valor 2
```

- Suma de dos variables

```
int numero1 = 3;
int numero2 = 7;
int suma = numero1 + numero2; // aquí suma tendrá valor 10
```

- Suma de una variable y un literal

```
int numero1 = 2;
int suma = numero1 + 6; // aquí suma tendrá valor 8
```

Tipos primitivos en C#: int

Operadores y funciones para trabajar con variables de tipo int:

- Suma de una variable y un literal, guardando el valor en la propia variable

```
int numero = 2; // aquí numero tendrá valor 2
numero = numero + 3; // aquí numero tendrá valor 5
```

- Como la operación anterior es muy común, existe una sintaxis alternativa para realizarla:

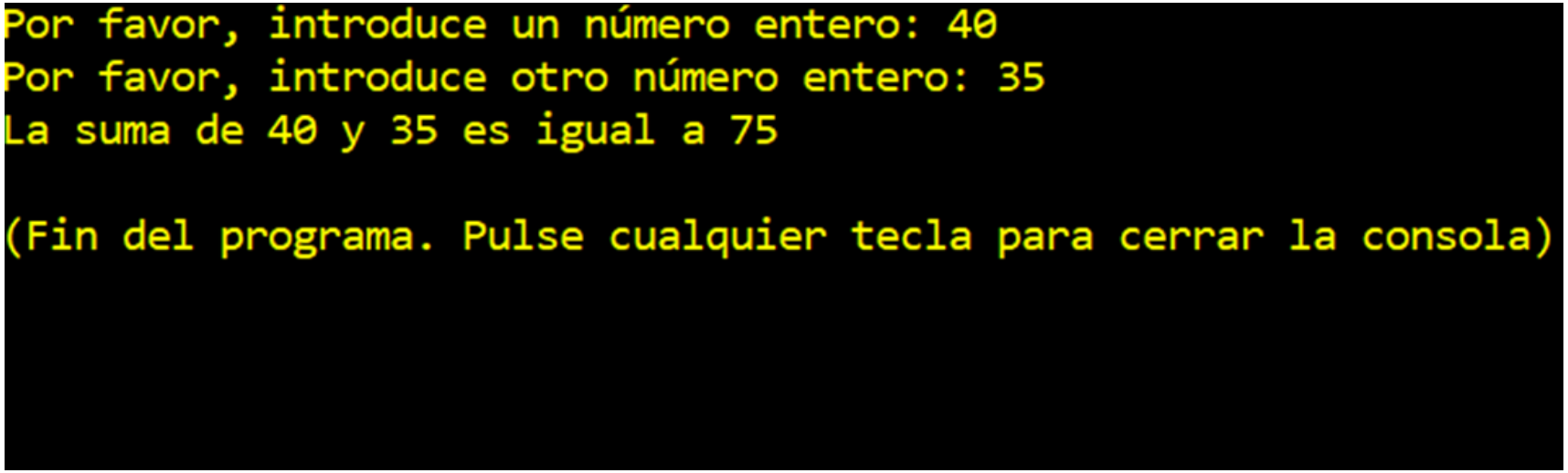
```
int numero = 2; // aquí numero tendrá valor 2
numero += 3; // aquí numero tendrá valor 5
```

- Cuando la suma es de 1, se conoce como incremento, y tiene otra sintaxis especial:

```
Int numero = 2; // aquí numero tendrá valor 2
numero++; // aquí numero tendrá valor 3
```

Tipos primitivos en C#: int

Programa 4: Suma. Diálogo en el que el programa pide al usuario que introduzca dos números enteros. Después, el programa escribe por pantalla una frase con su suma



Tipos primitivos en C#: int

Más operadores y funciones para trabajar con variables de tipo int:

- Resta

```
int dos = 3 - 1; // aquí dos tendrá valor 2
```

- Multiplicación

```
int diez = 2 * 5; // aquí diez tendrá valor 10
```

- División ENTERA

```
int dos = 5 / 2; // aquí dos tendrá valor 2 (¡NO 2.5!)
```

¡CUIDADO con dividir entre 0! → `int dos = 5/0; // ERROR`

- Módulo (resto de la división). Si a % b = 0 significa que a es divisible por b (Ej: a%2=0 -> a es par)

```
int dos = 7 % 5; // aquí dos tendrá valor 2
```

¡CUIDADO con obtener el módulo sobre 0! → `int dos = 7%0; // ERROR`

Tipos primitivos en C#: int

Varios de los operadores anteriores poseen sus sintaxis cortas, igual que para la suma:

- Resta

```
int numero = 3; // aquí numero tendrá valor 3
numero -= 2; // aquí numero tendrá valor 1
numero--; // aquí numero tendrá valor 0
```

- Multiplicación

```
int numero = 2; // aquí numero tendrá valor 2
numero *= 5; // aquí numero tendrá valor 10
```

- División ENTERA

```
int numero = 5; // aquí numero tendrá valor 5
numero /= 2; // aquí numero tendrá valor 2 (¡NO 2.5!)
```

Tipos primitivos en C#: int

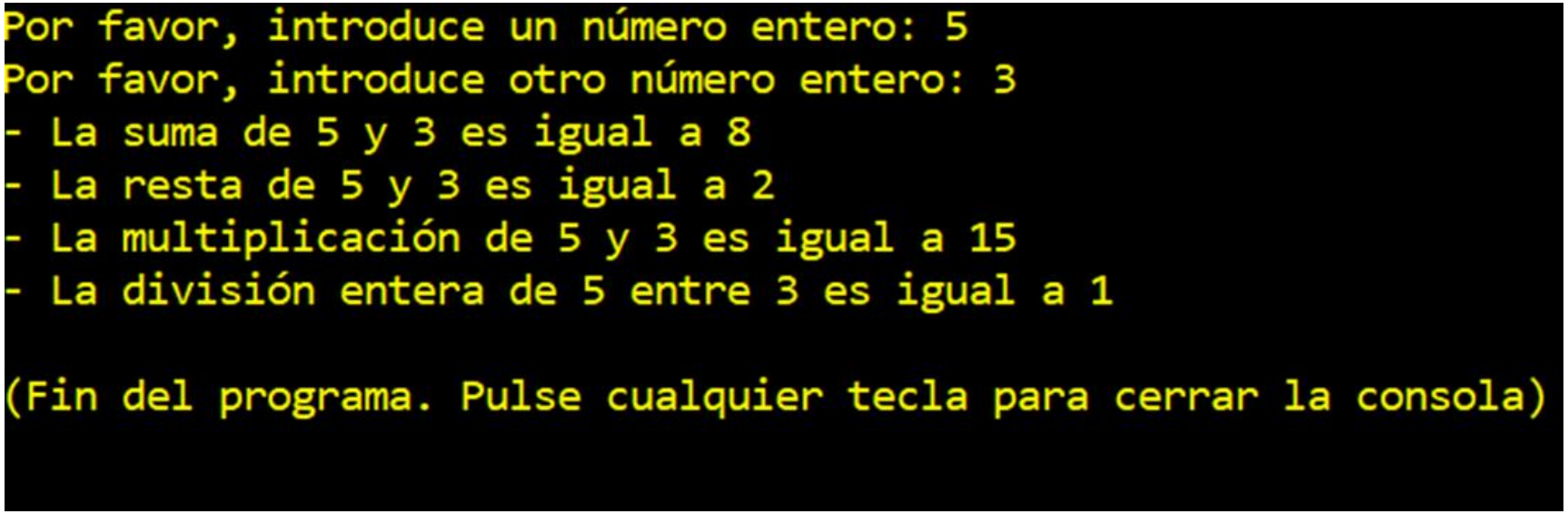
Más operadores y funciones para trabajar con variables de tipo int:

Documentación oficial en español:
https://docs.microsoft.com/es-es/dotnet/api/system.int32?view=net-5.0

Documentación oficial en inglés (sin errores debidos a mala traducción):
https://docs.microsoft.com/en-us/dotnet/api/system.int32?view=net-5.0

Tipos primitivos en C#: int

Programa 5: Aritmética. Diálogo en el que el programa pide al usuario que introduzca dos números enteros. Después, el programa escribe por pantalla su suma, su resta, su multiplicación y su división entera). Si el segundo número es 0, saltará una excepción de división por cero (no se controlará el caso de error, de momento…)



Tipos primitivos en C#: decimal

Tipo de dato para guardar números con decimales exactos*: decimal

- Ejemplos de constante → -100.221m, 0.45M, 100m, 100
- Declaración → decimal numConDec1;
- Declaración + inicialización → decimal numConDec1 = -38m;
- Valor por defecto → 0.0m
- Leer valor de variable → decimal numConDec2 = numConDec1;
- Escribir valor de variable → numConDec1 = 20.3m;

* NOTA: aunque existen los tipos float y double, para este curso se usará decimal, por guardar valor exacto

Tipos primitivos en C#: decimal

Función de .NET Framework para comprobar si lo que el usuario escribe en la consola es o no un número decimal:

- bool **decimal.TryParse**(string candidato, out decimal numero);
  - si el string candidato corresponde a un decimal, devuelve true y asigna a la variable numero el valor numérico de candidato
  - En caso contrario, devuelve false y 'numero' contendrá el valor por defecto para decimal: 0.0

```
string numStr = System.Console.ReadLine();
decimal num;
bool isNumDecimal = decimal.TryParse(numStr, out num);
```

O de forma más compacta:

```
bool isNumDecimal = decimal.TryParse(System.Console.ReadLine(), out decimal num);
```

Tipos primitivos en C#: decimal

¡CUIDADO!

- Los literales (valores constantes) de tipo decimal que se escriban en el código usarán el punto ‘.’ como carácter separador entre parte entera y parte decimal, pero…

- … si un programa de consola configurado para idioma español necesita pedir por pantalla al usuario un valor de tipo decimal, en vez de punto ‘.’, el usuario deberá escribir coma ‘,’ como separador decimal para que la instrucción decimal.TryParse() no dé error (no me creáis, probadlo…)

Tipos primitivos en C#: decimal

Operadores y funciones para trabajar con variables de tipo decimal:

- Suma de dos literales

```
decimal dec = 1.1m + 3.9m; // aquí dec tendrá valor 5m
```

- Suma de dos variables

```
decimal numero1 = 3; // se asigna el valor 3 de tipo int. Se aplica un casting implícito a decimal
decimal numero2 = 7; // se asigna el valor 7 de tipo int. Se aplica un casting implícito a decimal
decimal suma = numero1 + numero2; // aquí suma tendrá valor 10m
```

- Suma de una variable y un literal

```
decimal numero1 = 2.3m;
decimal suma = numero1 + 1.4m; // aquí suma tendrá valor 3.7m
```

Tipos primitivos en C#: decimal

Más operadores y funciones para trabajar con variables de tipo decimal:

- Resta

```
decimal dec = 3.2m - 1.1m; // aquí dec tendrá valor 2.1m
```

- Multiplicación

```
decimal diez = 2 * 5; // aquí diez tendrá valor 10m
```

- División con decimales

```
decimal dosymedio = 5 / 2m; // aquí dosymedio tendrá valor 2.5 (sin la m, valdrá 2)
```

¡CUIDADO con dividir entre 0! → `decimal hecatombe = 5/0; // ERROR`

- Módulo (también existe como operador de decimales, pero no tiene tanto sentido como con los enteros, y no se va a ver en el curso)

Tipos primitivos en C#: decimal

Más operadores y funciones para trabajar con variables de tipo decimal:

Documentación oficial en español:
https://docs.microsoft.com/es-es/dotnet/api/system.decimal?view=net-5.0

Documentación oficial en inglés (sin errores debidos a mala traducción):
https://docs.microsoft.com/en-us/dotnet/api/system.decimal?view=net-5.0

Tipos primitivos en C#: decimal

Programa 5 versión 2: Aritmética con decimales. Se trata de repetir el último ejercicio, pero esta vez con números decimales en vez de números enteros

```
Por favor, introduce un número decimal: 5,7
Por favor, introduce otro número decimal (distinto de 0): 3,2
La suma de 5,7 y 3,2 es igual a 8,9
La resta de 5,7 y 3,2 es igual a 2,5
El producto de 5,7 y 3,2 es igual a 18,24
La división entre 5,7 y 3,2 es igual a 1,781

(pulse cualquier tecla para terminar)
```

Tipos primitivos en C#: char

Tipo de dato para guardar caracteres individuales: char

- Ejemplos de constante → ‘a’, ‘1’, ‘:’, ‘\u03B1’ // ‘α’ codificada en UTF-16
- Declaración → char caracter1;
- Declaración + inicialización → char caracter1 = ‘!’;
- Valor por defecto → ‘\0’ (carácter de final de string)
- Leer valor de variable → char letra = caracter1;
- Escribir valor de variable → caracter1 = ‘?’;

Tipos primitivos en C#: char

Operadores y funciones para trabajar con variables de tipo char:

Documentación oficial en español:
https://docs.microsoft.com/es-es/dotnet/api/system.char?view=net-5.0

Documentación oficial en inglés (sin errores debidos a mala traducción):
https://docs.microsoft.com/en-us/dotnet/api/system.char?view=net-5.0

Conjuntos de tipos primitivos en C#: Array

Array: estructura de tamaño fijo para guardar varios datos del mismo tipo en una sola variable:

- Sintaxis de declaración de una variable de tipo Array de números enteros:

```
int[] numbers = new int[5];
// variable en la que poder guardar 5 enteros diferentes. Por defecto el valor de los 5 enteros es 0
```

- Sintaxis de declaración+inicialización de una variable de tipo Array de números enteros:

```
int[] numbers = new int[5] {10, 20, 30, 40, 50};
// variable en la que se pueden guardar 5 enteros diferentes con los valores iniciales definidos entre corchetes. Probad a inicializar menos de 5 elementos, y luego más de 5, a ver qué pasa
```

Conjuntos de tipos primitivos en C#: Array

Para acceder a los distintos valores de los arrays se usan **índices** (números enteros):

- Sintaxis de asignación de un entero guardado en una variable de tipo Array de números enteros:

```
int number = numbers[3];
// probad a cambiar el índice a -2, -1, 0, 1, 2, 3, 4, 5, 6 y 7, a ver qué pasa…
```

¡¡IMPORTANTE!! Los índices de un array empiezan en 0, es decir, para acceder al 1º elemento del array, hay que usar el índice 0, NO el índice 1.

Como truco, el índice de un elemento en un array ES IGUAL a su posición menos uno:

- Ej: El último elemento de un array de tamaño 10 tendrá índice 9, y así con todos…

Conjuntos de tipos primitivos en C#: Array

Igual que se ha visto para variables de tipo Array de int, se pueden definir y utilizar Arrays de string, Arrays de char, Arrays de decimal o Arrays de bool. El ejemplo, para string:

- string[] meses = new string[3]; // variable en la que se pueden guardar 3 strings diferentes
  // por defecto el valor de los 3 strings es null

- string[] meses = new string[3] {“enero”, “febrero”, “marzo”};
  // variable en la que se pueden guardar 3 strings diferentes con los valores iniciales definidos entre corchetes

Conjuntos de tipos primitivos en C#: Array

- string[] meses = new string[3] {“enero”, “febrero”, “marzo”};
  string mes1 = meses[0]; // mes1 tiene valor “enero”
  string mes2 = meses[1]; // mes2 tiene valor “febrero”
  string mes3 = meses[2]; // mes3 tiene valor “marzo”

Conjuntos de tipos primitivos en C#: Array

Más operaciones y funciones con arrays:

- string[] meses = new string[3] {“enero”, “febrero”, “marzo”};

- Obtener el nº de elementos de un Array:

meses.Length; // devuelve el nº de elementos del array. Aquí: 3

- Convertir un Array en una lista (ver más adelante):

List<string> ListMeses = meses.ToList(); // crea una lista del tamaño del nº de elementos del array, con los elementos de la lista en el mismo orden que tenían en el array

Conjuntos de tipos primitivos en C#: Array

Más operaciones y funciones con arrays:

string cadena = “HOLA MUNDO”;

- añadimos una nueva operación para el tipo string que devuelve un array de strings: Split

```
string[] palabras = cadena.Split(“ “);
// aquí palabras tendrá valor {“HOLA”, “MUNDO”}
// palabras[0] devolverá “HOLA”
// palabras[1] devolverá “MUNDO”
```

Conjuntos de tipos primitivos en C#: Array

Operadores y funciones para trabajar con variables de tipo Array:

Documentación oficial en español:
https://docs.microsoft.com/es-es/dotnet/api/system.array?view=net-5.0

Documentación oficial en inglés (sin errores debidos a mala traducción):
https://docs.microsoft.com/en-us/dotnet/api/system.array?view=net-5.0

Conjuntos de tipos primitivos en C#: Array

Programa 6: Posición en array. Programa declara un array de cinco strings y da valor distinto a cada uno. El programa pide al usuario que escriba un entero, usuario lo escribe, programa lo lee, usa el entero leído como posición en el array, y escribe por pantalla el string guardado en esa posición (RECORDAD, la posición y el índice NO VALEN LO MISMO)

```
Por favor, introduce un entero entre 1 y 5: 4
El 4º string almacenado en el array es CUATRO

(Fin del programa. Pulse cualquier tecla para cerrar la consola)
```

El array del programa tiene los valores: “UNO”, “DOS”, “TRES”, “CUATRO” y “CINCO”

Conjuntos de tipos primitivos en C#: List

List: estructura de tamaño indeterminado para guardar varios datos del mismo tipo en una sola variable

- Sintaxis de declaración + inicialización de una variable de tipo List de números enteros:

```
List<int> bingo = new List<int>(); // se crea una lista vacía que puede contener solo valores enteros
```

- Sintaxis para añadir un nuevo número entero al final de la lista de números enteros:

```
List<int> bingo = new List<int>(); // aquí bingo = {} (vacía)
bingo.Add(1); // aquí bingo = {1} (un elemento)
bingo.Add(-23); // aquí bingo = {1, -23} (dos elementos)
```

- !!! Cuidado con declarar sin inicializar una variable de tipo List, porque su valor por defecto es null, y ejecutar cualquier función (Add, u otra) sobre null genera ERROR DE EJECUCIÓN

Conjuntos de tipos primitivos en C#: List

Igual que se ha visto para variables de tipo List de int, se pueden definir y utilizar Listas de string, Listas de char, Listas de decimal o Listas de bool. El ejemplo, para string:

- List<string> palabras = new List<string>();
  palabras.Add(“Hola”);
  palabras.Add(“Mundo”);

Conjuntos de tipos primitivos en C#: List

Existe una sintaxis especial solo para declaración + inicialización de variables List:

- List<string> palabras = new List<string> { “Hola”, “Mundo” };

Para tener cada elemento de la lista en una línea, se puede escribir del siguiente modo:

- List<string> palabras = new List<string>
  {
  “Hola”,
  “Mundo”
  };

Conjuntos de tipos primitivos en C#: List

Más operaciones y funciones con listas:

- Vaciar la lista:

```
bingo.Clear();
```

- Obtener el nº de elementos de una lista:

```
int listSize = bingo.Count;
```

- Convertir la lista en Array:

```
int[] arrayInt = bingo.ToArray(); // crea un array del tamaño del nº de elementos de la lista, con los elementos del array en el mismo orden que tenían en la lista
```

Conjuntos de tipos primitivos en C#: List

Operadores y funciones para trabajar con variables de tipo List:

Documentación oficial en español:
https://docs.microsoft.com/es-es/dotnet/api/system.collections.generic.list-1?view=net-5.0

Documentación oficial en inglés (sin errores debidos a mala traducción):
https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1?view=net-5.0

Tipos primitivos en C#: bool

Tipo de datos para guardar valores de verdadero/falso: bool

- Ejemplos (únicos) de constante → true, false
- Declaración → bool check1;
- Declaración + inicialización → bool check1 = false;
- Valor por defecto → false
- Leer valor de variable → bool esCierto = check1;
- Escribir valor de variable → check1 = true;

Tipos primitivos en C#: bool

Operadores y funciones para trabajar con variables de tipo bool:

Documentación oficial en español:
https://docs.microsoft.com/es-es/dotnet/api/system.boolean?view=net-5.0

Documentación oficial en inglés (sin errores debidos a mala traducción):
https://docs.microsoft.com/en-us/dotnet/api/system.boolean?view=net-5.0

Tipos primitivos en C#: cambio de tipo (casting)

Cambios de tipo explícitos (casting de variables):

{nombre de variable a asignar} = ({tipo de dato deseado}) {nombre de variable a convertir}

- No cualquier variable de un tipo se puede convertir a otra variable de cualquier otro tipo. Hay conversiones que darán error de compilación.

```
int x = 89;
double z = (double)x;
```

Tipos primitivos en C#: cambio de tipo (casting)

Cambios de tipo (explícitos e implícitos):

Documentación oficial en español:
https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/types/casting-and-type-conversions

Documentación oficial en inglés (sin errores debidos a mala traducción)
https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/types/casting-and-type-conversions

tecnara
Cluster TIC de Aragón / Aragón IT Cluster

05

Lógica condicional e iterativa en C#

integra

Lógica condicional e iterativa: if

Bloque if

```
if (1 <= num && num <= 100)
{
    System.Console.WriteLine("El número que ha introducido es " + num);
    finish = true;
}
```

- Se evalúa una expresión booleana (solo puede valer true o false)
  Si es true → se ejecuta el código del interior del bloque (lo escrito entre { y })
  Si es false → NO se ejecuta el código del interior del bloque
  En ambos casos, se sigue con la ejecución del código que haya después del bloque if

Lógica condicional e iterativa: if-else

Bloque if-else

```
if (1 <= num && num <= 100)
{
    System.Console.WriteLine("El número que ha introducido es " + num);
    finish = true;
}
else
{
    System.Console.WriteLine("El número que ha introducido no está dentro del rango [1-100]");
}
```

- Se evalúa una expresión booleana (solo puede valer true o false)
  Si es true → se ejecuta el código del interior del bloque if y NO se ejecuta el bloque else
  Si es false → NO se ejecuta el bloque if y se ejecuta el código del interior del bloque else
  En ambos casos, se sigue con la ejecución del código que haya después del bloque else

Lógica condicional e iterativa: if-else

Bloque if-else usando operador ternario (para abreviar)

```
string respuesta = Console.ReadLine();
int terminar;
if(respuesta == "S")
{
    terminar = 1;
}
else
{
    terminar = 2;
}
Console.WriteLine(terminar);
```

```
string respuesta = Console.ReadLine();
int terminar = respuesta == "S" ? 1 : 2;
Console.WriteLine(terminar);
```

O, si se entiende mejor con paréntesis…

```
string respuesta = Console.ReadLine();
int terminar = (respuesta == "S" ? 1 : 2);
Console.WriteLine(terminar);
```

Lógica condicional e iterativa: if-else

Bloque if-else usando operador ternario (para abreviar)

```
string respuesta = Console.ReadLine();
bool terminar;
if(respuesta == "S")
{
    terminar = true;
}
else
{
    terminar = false;
}
Console.WriteLine(terminar);
```

- Se evalúa una expresión booleana (solo puede valer true o false)
  Si es true → se ejecuta el código del interior del bloque if y NO se ejecuta el bloque else
  Si es false → NO se ejecuta el bloque if y se ejecuta el código del interior del bloque else
  En ambos casos, se sigue con la ejecución del código que haya después del bloque else

Lógica condicional e iterativa: operadores condicionales

Operadores condicionales (para usar dentro de las condiciones de los bloques if, u otros)

Operador	Finalidad
==	Comprobar igualdad.
!=	Comprobar desigualdad.
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que

Lógica condicional e iterativa: operadores condicionales

Operadores condicionales: ejemplos

```
int Edad = 56;
//Es igual a 56
if (Edad == 56)
{
    Console.WriteLine("Tiene 56 años");
}
//Es distinto
if (Edad != 18)
{
    Console.WriteLine("No tiene 18 años");
}
//Es distinto
if (Edad >= 18)
{
    Console.WriteLine("Es mayor de edad");
}
//Es distinto
if (Edad < 18)
{
    Console.WriteLine("Es menor de edad");
}
```

Lógica condicional e iterativa: operadores condicionales

Expresiones que comprueban varias condiciones de vez
(salvo el NOT, que niega la condición sobre la que se aplique)

Operador	Finalidad
&&	AND condicional.
||	OR condicional.
!	NOT condicional.

- Si la condición (a == 1) devolviera true, la condición !(a == 1) devolvería false
- Si la condición (a == 1) devolviera false, la condición !(a == 1) devolvería true

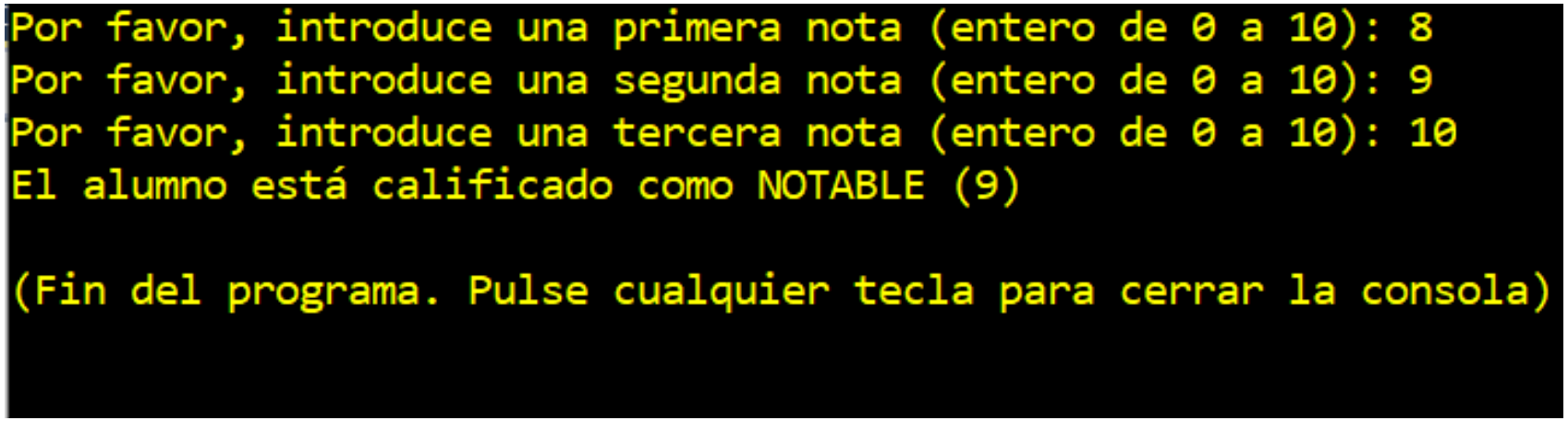
Lógica condicional e iterativa: operadores condicionales

Expresiones que prueban varias condiciones de vez (salvo el NOT)

```
int Edad = 56;
//OR
if (Edad == 56 || Edad == 18)
{
    Console.WriteLine("Tiene 56 años O 18 años");
}
//AND
if (Edad == 56 && Edad > 18)
{
    Console.WriteLine("Tiene 56 años Y 18 años");
}
//XOR
if (!Edad >= 18)
{
    Console.WriteLine("Es menor de edad");
}
```

Lógica condicional e iterativa: operadores condicionales

Programa 7: Nota media. Programa pide al usuario que escriba tres notas académicas (enteros entre 0 y 10), usuario los escribe, programa los lee, programa calcula la media aritmética de las notas, y muestra por pantalla la calificación final: SUSPENSO si media < 5, APROBADO si media entre 5 y 7, NOTABLE si media entre 7 y 9, SOBRESALIENTE si media > 9. Controlar que las notas introducidas sean válidas (entre 0 y 10). ¿Enteras o decimales?



Lógica condicional e iterativa: switch-case

Bloque switch-case: cuando el código que se debe ejecutar depende del valor que tenga una variable. Útil por ejemplo para programas con menú y selección de opción.

```
string Color = "Rojo";
switch (Color)
{
    case "Rojo":
        Console.WriteLine("Es Rojo");
        break;
    case "Verde":
        Console.WriteLine("Es verde");
        break;
    default:
        Console.WriteLine("Color no contemplado");
        break;
}
```

Lógica condicional e iterativa: switch-case

Programa 8: Calculadora. Programa pide al usuario que escriba dos enteros, usuario los escribe, programa los lee, después programa escribe un menú con opciones de distintas operaciones aritméticas a realizar con los dos números anteriores, y pide al usuario que elija una opción. Según la opción que elija, el programa mostrará el resultado de una operación u otra y terminará

```
Por favor, introduce un número entero: 5
Por favor, introduce otro número entero: 3

1 - Suma
2 - Resta
3 - Multiplicación
4 - División entera
Selecciona una opción (1, 2, 3 o 4): 3
- La multiplicación de 5 y 3 es igual a 15

(Fin del programa. Pulse cualquier tecla para cerrar la consola)
```

Lógica condicional e iterativa: foreach

Bloque foreach: cuando se desea recorrer un array o una lista para aplicar una misma operación a cada elemento, SIN necesitar tener en cuenta para nada los índices de los elementos (si los índices son importantes durante la operación, mejor usar un bloque for)

```
string[] ArrayCadenas = {"Hola", "Adios"};

foreach (string s in ArrayCadenas)
{
    System.Console.WriteLine(s);
}
```

Lógica condicional e iterativa: foreach

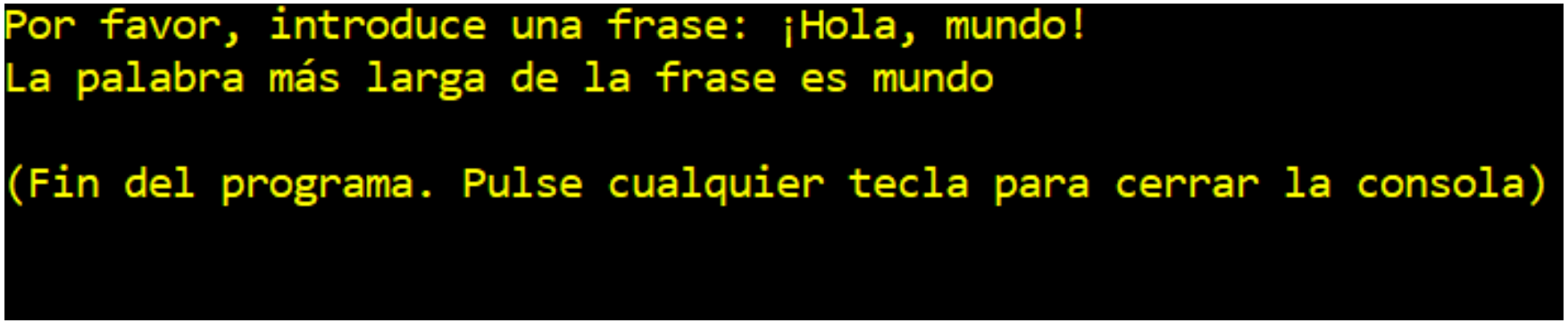
Programa 9: Contador de vocales. Programa pide al usuario que escriba una frase, usuario la escribe, programa la lee, programa pide al usuario que escriba una vocal, usuario la escribe, programa la lee, programa calcula el nº de veces que aparece la vocal en la frase y muestra por pantalla el resultado del cálculo. ¿Mayúsculas/minúsculas? ¿tildes? ¿diéresis?

```
Por favor, introduce una frase: Ábaco es un instrumento matemático interesante
Por favor, introduce una vocal (a, e, i, o, u): a
La frase introducida tiene 5 vocales 'a'

(Fin del programa. Pulse cualquier tecla para cerrar la consola)
```

Lógica condicional e iterativa: foreach

Programa 10: Palabra más larga. Programa pide al usuario que escriba una frase, usuario la escribe, programa la lee, programa calcula cuál es la palabra más larga de la frase y muestra por pantalla el resultado del cálculo. ¿Y si la frase tiene dos 'palabras más largas'?



Antes de dividir la frase en palabras, eliminad todos los caracteres NO alfabéticos. ¿y los números, los contamos como palabras?

Lógica condicional e iterativa: for

Bloque for: cuando se desea realizar una misma tarea un número conocido o acotado de veces. Útil para recorrer arrays y listas, aplicando una misma operación a cada elemento, cuando los índices se usan dentro del bloque para la operación. Útil también para generar y recorrer rangos numéricos.

```
string[] ArrayCadenas = {"Hola", "Adios"};

for (int i=0; i<2; i++)
{
    System.Console.WriteLine(ArrayCadenas[i]);
}
```

Lógica condicional e iterativa: for

Programa 11: Palabra al revés. Programa pide al usuario que escriba una frase, usuario la escribe, programa la lee, programa obtiene la misma frase pero con sus caracteres escritos en orden inverso y la muestra por pantalla

```
Por favor, introduce una frase: ¡Hola, mundo!
La frase introducida, escrita al revés, queda así:
"!odnum ,aloH¡"

(Fin del programa. Pulse cualquier tecla para cerrar la consola)
```

Lógica condicional e iterativa: while

Bloque while: cuando se desea realizar una misma tarea un número desconocido de veces (hasta que se cumple una condición). Útil para recorrer listas o arrays que PUEDE SER que estén vacías

```
string[] ArrayCadenas = {"Hola", "Adios"};
int y = 0;
while (y < 2)
{
    System.Console.WriteLine(ArrayCadenas[i]);
    y++;
}
```

Lógica condicional e iterativa: while

Programa 12: Lista números pares. Programa que calcula la lista de números pares que hay entre el 0 y el 100 y la muestra por pantalla. ¿y si en vez de 0 y 100 queremos otro rango?

```
Lista de pares entre 0 y 100:
"0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34,
36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68,
70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90, 92, 94, 96, 98, 100"

(Fin del programa. Pulse cualquier tecla para cerrar la consola)
```

Lógica condicional e iterativa: do-while

Bloque do-while: cuando se desea realizar una misma tarea un número desconocido de veces (hasta que se cumple una condición), y la primera vez es obligatoria. Útil para programas que repiten una petición de valor al usuario hasta que el valor introducido es correcto o es un valor especial de terminación

```
string[] ArrayCadenas = {"Hola", "Adios"};
int y = 0;
do
{
    System.Console.WriteLine(ArrayCadenas[y]);
    y++;
} while(y < 2);
```

Lógica condicional e iterativa

Programa 13: Gestión de cuenta bancaria

- El programa mostrará un menú con las siguientes opciones:
  1– Ingreso de efectivo
  2–Retirada de efectivo
  3- Lista de todos los movimientos
  4- Lista de todos los ingresos de efectivo
  5- Lista de todas las retiradas de efectivo
  6- Mostrar saldo actual
  7- Salir.
- Existirá una variable que guardará el saldo de un único cliente y sobre ese saldo se realizarán los ingresos (sumas de dinero al saldo actual) o retiradas (restas de dinero al saldo actual).
- Una vez finalizada cualquier operación, el programa preguntará si queremos realizar alguna otra operación:
  - Si usuario responde que sí, se volverá a mostrar el menú, esperando que el usuario elija otra opción
  - en caso contrario mostrará el valor actual de saldo y finalizará el programa.

Ejercicio evaluable

Programa 13 versión 2: Gestión de cuenta bancaria multiusuario

- Antes de mostrar ningún menú de opciones, el programa pedirá al cliente introducir un nº de cuenta y un pin
- Existirá una lista con nºs cuenta y otra con pines, de tal forma que el iésimo nº cuenta de la primera lista tendrá asociado el iésimo pin de la segunda lista
- La información del saldo y los movimientos del cliente del programa 13 versión 1, aquí estarán incluidos en listas que, igual que para el pin, contendrán en su elemento iésimo la información del cliente que en la lista de nºs cuenta tenga su nº cuenta en esa misma posición
- El programa comprobará si el nº cuenta y el pin introducidos por el cliente existen y están en la misma posición (llamémosla i) para las dos listas correspondientes, y solo en ese caso se permitirá seguir igual que la versión 1, considerando que la información de saldo y movimientos a consultar o modificar se obtendrán de la posición i de las listas correspondientes

MUCHAS GRACIAS

Pasión por la innovación

www.integratecnologia.es