Apuntes básicos de programación en C#

> Autor: Miguel Medina Ballesteros (Maximetinu)
> Contacto: maximetinu (at) gmail (dot) com

Antes que nada, advertir que los apuntes que yo recopilo en esta guía no especifican la totalidad del lenguaje de programación C# ni lo hacen con tanta profundidad como sí que cabe esperar de su documentación oficial, que referencio a continuación:

- Documentación oficial de C#
- Guía de programación C#

En esas dos páginas, que en realidad son la misma, ya que la segunda es una sección de la primera; y navegando por el resto de secciones, están recopiladas absolutamente todas las características del lenguaje de programación C#, que es el que utilizaremos en la asignatura porque es el que utiliza el motor gráfico Unity.

La razón por la que he hecho este documento es para abstraeros de todas las características avanzadas del lenguaje, ya que si intentáis aprender directamente desde 0 de esa web, os perderéis entre muchos conceptos avanzados y palabras que no entendéis. Aún así, os recomiendo usarlas para resolver dudas o ampliar conocimientos, intentando ignorar los conceptos avanzados y buscando específicamente sobre las cosas que ya hemos visto en clase.

Entorno de desarrollo

El primer día, estuvimos programando en el siguiente editor web: dotnetfiddle.net. Es una buena opción para empezar, ya que cualquier persona con acceso a internet puede utilizarla.

Tarde o temprano acabaramos utilizando también Visual Studio, ya que será el que utilicemos al programar en Unity.

Código base

El código que se nos presenta al abrir el editor es el siguiente:

```
using System;

public class Program
{
    public static void Main()
    {
        // Escribe aquí tu código
    }
}
```

Hay que tener cuidado de no romper la estructura base. De momento **tenéis prohibido modificar la estructura básica**, ya que no entenderéis lo que estáis haciendo. Más adelante veremos conceptos más avanzados que nos permitirán entender qué está pasando en esas líneas.

> **Para los curiosos**:
> - En la primera línea estamos importando el espacio de nombres `System` para poder utilizar las clases estáticas `Console` y `Convert`, sin las cuales no podríamos mostrar ni introducir datos al programa.
> - Necesitamos una clase pública cualquiera que contenga a la función principal del programa `Main()`. Eso es lo que hace la línea `public class Program`
> - Necesitamos una función llamada obligatoriamente `Main()` que le dirá al programa por donde tiene que empezar a ejecutarse.

Tipos de datos

String

String significa "cadena" en inglés. Como su propio nombre indica, un string es una cadena de caracteres, es decir, de letras. Vamos, un texto al uso, por eso digo en clase que es una variable de tipo texto.

```
string nombreProfe = "Miguel";
string apodoProfe = "Metinu";
```

Los textos también pueden encadenarse entre sí con el operador suma:

```
string nombreProfe = "Miguel";
string primerApellidoProfe = "Medina";
string segundoApellidoProfe = "Ballesteros";
string nombreCompletoProfe;

nombreCompletoProfe = nombreProfe + primerApellidoProfe + segund
```

Ahora la variable nombreCompletoProfe sería igual a "MiguelMedinaBallesteros"

Para meter espacios entre palabras, directamente encadenamos un espacio en blanco entre palabra y palabra, tal que así:

```
nombreCompletoProfe = nombreProfe + " " + primerApellidoProfe +
```

Ahora la variable nombreCompletoProfe sería igual a "Miguel Medina Ballesteros"

Int

Int viene de la palabra inglesa *Integer*, que significa entero, y se refiere a número entero (es decir, sin decimales).

Se usa para representar valores que siempre vayan a ser números enteros, como por ejemplo el nivel de un personaje, el número de veces que has muerto, un puesto en el ránking, un número de goles, puntos de vida y de magia (si decidimos medir el daño en números enteros, claro), etc.

```
int numeroCaramelos = 10;
int numeroNinos = 5;
int caramelosPorNiño;

caramelosPorNiño = numeroCaramelos / numeroNiños;
```

Ahora la variable caramelosPorNiño sería igual a 2.

> Cuidado, porque en el código de verdad no se admiten eñes, ya que es una letra especial del español. Es por eso que se programa en inglés.

Si en lugar de 5 niños, fuesen 3 niños, al hacer la división se ignoraría la parte decimal.

```
int numeroCaramelos = 10;
int numeroNiños = 3;
int caramelosPorNiño;

caramelosPorNiño = numeroCaramelos / numeroNiños;
```

Ahora la variable caramelosPorNiño sería igual a 3. La división 10/3 realmente es igual a 3.333, pero Csharp ignora la parte decimal y se queda en 3.

Pero, SI Csharp ignora la parte decimal, ¿qué pasaría si la división diera 2.5, como en el caso de 10 caramelos y 2 niños; o si la división diera un número decimal que es casi 3, como en el caso de 11 caramelos y 2 niños, que sería 2.75? **Csharp no redondea, por lo que en esos casos daría 2**.

Float

Float es un número con decimales, y el nombre del tipo viene de "coma flotante", que es como se llama a la representación de este tipo de números en binario.

Se usa para representar valores que sean valores que puedan tener decimales, como por ejemplo una distancia, una velocidad, un tiempo, una posición en el mapa, la rotación en ángulos de un objeto, etc.

```
float spellCooldown = 3.225f;
```

> *Cooldown* significa enfriamiento y *spell* significa hechizo. La variable spellCooldown se refiere al tiempo de enfriamiento de un hechizo, que es el tiempo que tarda en recargarse para poder volver a usarlo. En el ejemplo son 3 segundos y 225 milisegundos.

La letra F que se pone junto al número es para especificarle al compilador que se trata de un float, ya que si el cooldown fuesen 3 segundos e introdujeramos un 3 solamente, no tendría forma de saberlo. La mayoría de las veces no pasa nada si se nos olvida, ya que convierte el entero a float

ignorando la parte decimal, que en ese caso sería ninguna. Pero a veces puede dar problemas, por eso lo ideal es ponerlo.

```
// Si el hechizo tarda en recargarse 3 segundos:
float spellCooldown;

spellCooldown = 3.0f;   // CORRECTO
spellCooldown = 3f;     // CORRECTO
spellCooldown = 3;      // INCORRECTO
```

Bool

Bool es una variable lógica, es decir, que sólo puede tomar el valor de verdadero o falso. Su nombre viene del matemático inglés George Boole, que sentó las bases de la revolución que supuso la informática.

Se usa para representar valores que pueden estar solo en dos estados, por ejemplo encendido o apagado, activado o desactivado, vivo o muerto, equipado o desequipado, etc.

```
bool estosApuntesMolan = true;
bool modoDiosActivado = false;
```

Su uso no tiene sentido sin las instrucciones condicionales if/else, que se explican más adelante. Por ello, usaré if en los ejemplos.

```
// Si tiene equipado el anillo de velocidad equipado, corre un 2
// (supongamos que las variables ya han sido declaradas e inicia

if (anilloVelocidadEquipado == true)
{
    velocidadMovimiento = velocidadMovimiento * 1.8f;
}
```

> Multiplicar por 1.2 es lo mismo que aumentar la velocidad en un 20%. Seguid mi lógica. Si multiplicar por 2 es el doble, y multiplicar por 1 es dejarlo igual, multiplicar por 1.2 sería aumentar el valor en un 120%. Si el 100% del valor es el valor original, el 120% es el valor aumentado en un 20%.
> - Si quisiéramos que corriera el doble, es decir, el 200%,

> multiplicaríamos por 2.
> - Si en lugar de un anillo mágico tuviera una maldición de lentitud que redujese su velocidad de movimiento un 20%, multiplicaríamos por 0.8, que sería igual que quedarnos solo con el 80% del valor original.

Operadores aritméticos

Suma +

En valores numéricos, su uso es evidente:

```
int enteroA = 2;
int enteroB = 3;
float decimalA = 2.2f;
float decimalB = 0.4f;
float resultado;

resultado = enteroA + enteroB;      // resultado es igual a 5.0f
    // enteroA + enteroB es igual a 5, pero se
    // convierte en float al hacer la asignación

resultado = decimalA + decimalB;    // resultado es igual a 2.6f
resultado = enteroA + decimalB;     // resultado es igual a 2.4f
```

> La variable resultado es un float porque hemos previsto que la utilizaríamos en una operación con un float. Ante la posibilidad de operaciones con floats, mejor elegir float.

Encadenación de strings

En variables de tipo string, el operador + une el texto:

```
string nombre = "Miguel";
string apellidos = "Medina Ballesteros";

Console.WriteLine("Mi nombre es " + nombre);
// Imprime: Mi nombre es Miguel

Console.WriteLine("Mi nombre y apellidos es " + nombre + " " + a
// Imprime: Mi nombre y apellidos es Miguel Medina Ballesteros
```

El operador + es el único que puede usarse con strings. Resta, multiplicación, división, etc; no tienen sentido usarse en strings.

Resta -

Su uso es la inversa de la suma, como cabría esperar:

```
int enteroA = 2;
int enteroB = 3;
float decimalA = 2.2f;
float decimalB = 0.4f;
float resultado;

resultado = enteroA - enteroB;      // resultado es igual a -1.0
    // enteroA - enteroB es igual a -1, pero se
    // convierte en float al hacer la asignación
resultado = decimalA - decimalB;    // resultado es igual a 1.8f
resultado = enteroA - decimalB;     // resultado es igual a 1.6f
```

Incremento ++

Es una operación para sumar uno. Un +1.

```
int goles = 3;
bool golMetido = true;

if (golMetido == true)
{
    goles++;
}
```

Ahora la variable goles es igual a 4.

Es equivalente a hacer:

```
goles = goles + 1;
```

Decremento - -

Es una operación para restar uno. Un -1. Lo contrario al incremento.

```
int goles = 3;
bool eraFueraDeJuego = true;

if (eraFueraDeJuego == true)
{
    goles--;
}
```

Ahora la variable goles es igual a 2.

Es equivalente a hacer:

```
goles = goles - 1;
```

Multiplicación *

Su uso es evidente:

```
vidaTotal = vidaInicial * nivel;
```

Cuando hay sumas y restas de por medio, se resuelve primero la multiplicación:

```
vidaTotal = vidaInicial * nivel - dañoRecibido;
```

Si queremos cambiar el orden de las operaciones será necesario usar paréntesis.

```
// imaginemos una maldición que te resta 3 niveles:
if (estoyMaldito == true)
{
    vidaTotal = vidaInicial * (nivel - 3);
}
```

Si hiciéramos directamente `vidaTotal = vidaInicial * nivel - 3` estaríamos restando solo 3 puntos de vida al total, en lugar de calculando la vida de 3 niveles atrás.

División /

Es la inversa de la multiplicación, pero cuidado, porque **no se puede dividir por 0**. Además de que es matemáticamente imposible, el programa daría error de ejecución (es decir, fallaría mientras se está ejecutando).

```
vidaInicial = vidaTotal / nivel;
```

Resto %

El operador resto, también llamado operador módulo, devuelve el resto de hacer la división entre dos números enteros. Por ejemplo:

```
int caramelos = 10;
int niños = 3;
int caramelosSobrantes;

caramelosSobrantes = caramelos % niños;
```

Ahora la variable caramelosSobrantes sería igual a 1.

Se puede utilizar, por ejemplo, para determinar si un número es par o impar:

```
bool esPar;
bool miNumero = 457;

if (miNumero % 2 == 0)
{
    esPar = true;
}

if (miNumero % 2 == 1)
{
```

```
    esPar = false;
}
```

Asignación =

Aunque su uso resulte evidente, el símbolo `=` también es un operador. El operador de asignación, que asigna lo que haya a la derecha a la variable de la izquierda.

```
int miVariable = 5;
```

Lo más interesante son las abreviaturas siguientes:

Asignación y operación (abreviaturas)

Asignación y suma +=

Los siguientes fragmentos de código son equivalentes:

```
int puntosDeVida = 500;

// Curarse 100 puntos de vida:
puntosDeVida = puntosDeVida + 100;

// Ahora puntosDeVida vale 600
```

```
int puntosDeVida = 500;

// Curarse 100 puntos de vida:
puntosDeVida += 100;

// Ahora puntosDeVida vale 600
```

Asignación y resta -=

Los siguientes fragmentos de código son equivalentes:

```
int puntosDeVida = 500;

// Sufrir 100 puntos de daño:
```

```
puntosDeVida = puntosDeVida - 100;

// Ahora puntosDeVida vale 400
```

```
int puntosDeVida = 500;

// Sufrir 100 puntos de daño:
puntosDeVida -= 100;

// Ahora puntosDeVida vale 400
```

Asignación y multiplicación *=

Los siguientes fragmentos de código son equivalentes:

```
int puntosDeVida = 500;

// Multiplicar la vida por 2:
puntosDeVida = puntosDeVida * 2;

// Ahora puntosDeVida vale 1000
```

```
int puntosDeVida = 500;

// Multiplicar la vida por 2
puntosDeVida *= 2;

// Ahora puntosDeVida vale 1000
```

Asignación y división /=

Los siguientes fragmentos de código son equivalentes:

```
int puntosDeVida = 500;

// Dividir la vida entre 2:
puntosDeVida = puntosDeVida / 2;

// Ahora puntosDeVida vale 250
```

```
int puntosDeVida = 500;

// Multiplicar la vida entre 2
puntosDeVida /= 2;

// Ahora puntosDeVida vale 250
```

Asignación y resto %=

Funciona igual que los otros:

```
a = a % b;
```

```
a %= b;
```

> No lo he usado en mi vida, así que no le veo utilidad.

Input y Output por consola

Cuando estemos programando en consola y sin Unity interactuaremos con el programa mediante solo mediante texto.

Introducir texto por consola (Input)

Se hace mediante el método Console.ReadLine(), de la clase estática Console.

> Que no cunda el pánico. Qué es un método, qué una clase y qué es clase estática son conceptos que veremos más adelante.

Ejemplo:

```
string miNombre;

miNombre = Console.ReadLine();
```

Ahora la variable miNombre tendrá el valor de lo que hayamos introducido por consola.

Cuidado con:
- `Console.ReadLine()` no tiene argumentos, es decir, no recibe nada por parámetro, por eso **no se puede escribir nada entre los paréntesis**.
- `Console.ReadLine()` devuelve una variable de tipo string cargada con lo que el usuario ha introducido por teclado. **Si queremos leer un valor numérico, tendremos que convertirlo primero** usando `Convert.ToInt32(textoAConvertir)`, como veremos más adelante.

Cuando el código ejecuta `Console.ReadLine()`, el programa se detiene y no continúa hasta que introducimos un texto en consola y pulsemos enter.

Mientras el programa espera a que introduzcamos el texto, la consola muestra el símbolo `>`, para indicar que está esperando una entrada de texto.

> De ahí es de donde viene el famoso símbolo de la terminal (consola y terminal son sinónimos):
> 
>

Imprimir texto en la consola (Output)

> En informática, imprimir (*print*) se refiere a mostrar texto por pantalla, normalmente en la consola.

Se hace mediante el método Console.WriteLine(...), de la clase estática Console.

> De nuevo, que no cunda el pánico. Qué es un método, qué una clase y qué es clase estática son conceptos que veremos más adelante.

Ejemplo:

```
Console.WriteLine("HelloWorld!");
```

Al pasarle variables de tipo texto como argumento, también podemos unirlas con texto fijo para mejorar la presentación, así como usar varias variables:

```
string nombre = "Al'thor";
string clase = "Paladín";
int nivel = 80;

Console.WriteLine("Mi nombre es " + nombre + ", y soy un " + cla
```

Se imprimirá por consola el texto: `"Mi nombre es Al'thor, y soy un Paladín de nivel 80."`, con el punto final incluido.

Afortunadamente, el método WriteLine(...) convierte automáticamente la variable que le pasamos a texto, por tanto no hace falta que la convirtamos nosotros, como con el método ReadLine().

También se puede utilizar `Console.WriteLine(...)` para embellecer la interacción con el programa y que así quede mucho más clara su utilización por consola sin necesidad de leer el código. En combinación con `Console.ReadLine()` se puede utilizar así:

```
string nombre;
string clase;
int nivel;

Console.WriteLine("Introduce el nombre de tu personaje: ");
nombre = Console.ReadLine();

Console.WriteLine("Introduce la clase de tu personaje: ");
clase = Console.ReadLine();

Console.WriteLine("Introduce el nivel de tu personaje: ");
nivel = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Mi nombre es " + nombre + ", y soy un " + cla
```

Cómo convertir texto leído en variables de otro tipo

Para leer de teclado variables de tipo string, no hace falta realizar ninguna conversión:

```
string miNombre;

miNombre = Console.ReadLine();
```

Leer *ints* de teclado (Convertir de texto a número entero)

Se hace mediante el método `Convert.ToInt32(textoAConvertir)` de la clase estática `Convert`.

```
int miEdad;
string miEdadInput;

miEdadInput = Console.ReadLine();
miEdad = Convert.ToInt32(miEdadInput);
```

Leer *floats* de teclado (Convertir de texto a número decimal)

Se hace mediante el método `Convert.ToSingle(textoAConvertir)` de la clase estática `Convert`.

```
int miAltura;
string miAlturaInput;

miAlturaInput = Console.ReadLine();
miAltura = Convert.ToSingle(miAlturaInput);
```

> ¡Cuidado! Al introducir el valor en la consola, hay que **usar coma en vez de punto para expresar los decimales, al revés que en el código.**

Leer *bools* de teclado (Convertir de texto a variables booleanas)

Se hace mediante el método `Convert.ToBoolean(textoAConvertir)` de la clase estática `Convert`. Solo funciona si el texto a convertir es `false` o `true`; tal cual. No podemos introducir sí o no, ni verdadero o falso.

```
bool usuarioEsMayorDeEdad;
string usuarioEsMayorDeEdadInput;

usuarioEsMayorDeEdadInput = Console.ReadLine();
usuarioEsMayorDeEdad= Convert.ToBoolean(usuarioEsMayorDeEdadInpu
```

Para leer bools de teclado, existe otra opción: especificando al usuario que introduzca literalmente `si` o `no` y comparando el valor introducido por teclado en un condicional if/else:

```
bool usuarioEsMayorDeEdad;
string usuarioEsMayorDeEdadInput;

Console.WriteLine("¿Es usted mayor de edad? si/no");
usuarioEsMayorDeEdadInput = Console.ReadLine();

if (usuarioEsMayorDeEdadInput == "si")
{
    usuarioEsMayorDeEdad = true;
}
else if (usuarioEsMayorDeEdadInput == "no")
{
    usuarioEsMayorDeEdad = false;
}
else
{
    Console.WriteLine("Error al leer la respuesta. Se tomará el
    usuarioEsMayorDeEdad = false;
}
```

Condicionales if/else

Como no tiene sentido explicar if sin usar operadores de comparación, se utilizarán en los ejemplos. Los operadores de comparación se explican en el próximo apartado.

If

Si la condición que se evalúa en los paréntesis del if es verdadera, entra dentro de las llaves y ejecuta el código de su interior. Si no, se salta el código de dentro de las llaves.

```
if (condicion)
{
    // Do something
}
```

Por ejemplo:

```
if (miEdad >= 18)
{
    Console.WriteLine("¡Bien! ¡Ya puedo ir a la cárcel!);
}
```

Else

Después de un if, puede especificarse una instrucción else, que se ejecutará en caso de que la condición de los paréntesis del if sea evaluada como falsa.

```
if (miEdad >= 18)
{
    puedoEntrarALaDiscoteca = true;
}
else
{
    puedoEntrarALaDiscoteca = false;
}
```

Else If

También es posible encadenar el else con otro if. Por ejemplo:

```
if (miEdad >= 18)
{
    puedoEntrarAlConcierto = true;
}
else if (voyAcompañadoDeUnAdulto == true)
{
    puedoEntrarAlConcierto = true;
}
else
{
```

```
    puedoEntrarAlConcierto = false;
}
```

If anidados

También podemos poner un if dentro de otro, y otro, y otro... ¡Y tantos como queramos!

```
if (miEdad >= 18)
{
    Console.WriteLine("¡Bien! ¡Ya puedo ir a la cárcel!");

    if (tengoCarnetDeConducir == true)
    {
        Console.WriteLine("¡Y además ya puedo conducir!");
    }
    else
    {
        Console.WriteLine("Ahora solo me falta sacarme el carnet
    }
}
```

Operadores de comparación

Dentro de los paréntesis del if, podemos comprobar multitud de cosas. Si un número es igual a otro, o mayor, o menor, o si algo es verdadero o falso, o si un texto es igual a otro, o si lo que acabo de tocar es una bala o un botiquín para curarme... ¡Tenéis que tomar soltura con esto!

¡Esos sí! **Las variables a la izquierda y a la derecha del operador tienen que ser comparables**. No se puede comparar un string con un int, por ejemplo, el programa daría error de compilación y no podría ni ejecutarse.

Igualdad ==

Compara si las dos variables son iguales. Se lee como "es igual a".

```
if (miEdad == 64)
{
```

```
    Console.WriteLine("Ya mismo me jubilo.");
}
```

> Lo dicho, la variable miEdad tiene que ser de tipo entero, como el 64 de la derecha. También vale si es un float porque lo convierte automáticamente, pero sería un código feo, y eso es peor incluso que un error.

Desigualdad !=

Compara si las dos variables son diferentes. Se lee como "no es igual a" o "distinto de".

```
if (miNombre != "Miguel")
{
    Console.WriteLine("Vaya :( No me llamo como el profe");
}
```

> **Truco para saber que se escribe != y no al revés ~~=!~~**
> Porque, como veremos más adelante, el símbolo `!` indica negación (es decir, NO).
> Por tanto, `!=` se lee como "NO =", o, directamente "NO IGUAL" (por tanto, distinto de).

Menor <

Compara si la variable de la izquierda es estrictamente menor que la de la derecha (no vale que sean iguales, tiene que ser menor).

```
// En este ejemplo miEdad es siempre 18...
int miEdad = 18;

if (miEdad < 18)
{
    // ... el programa NO entrará en este if
    // porque 18 no es menor que 18 (es igual)
}

if (miEdad < 19)
{
    // ... el programa SÍ entrará en este if
```

```
    // porque 18 sí que es menor que 19
}
```

Mayor >

Compara si la variable de la izquierda es estrictamente mayor que la de la derecha (no vale que sean iguales, tiene que ser menor).

```
// En este ejemplo miEdad es siempre 18...
int miEdad = 18;

if (miEdad > 18)
{
    // ... el programa NO entrará en este if
    // porque 18 no es mayor que 18 (es igual)
}

if (miEdad > 19)
{
    // ... el programa NO entrará en este if
    // porque 18 no es mayor que 17
}
```

Menor o igual <=

Compara si la variable de la izquierda es menor o igual que la de la derecha.

```
// En este ejemplo miEdad es siempre 18...
int miEdad = 18;

if (miEdad <= 18)
{
    // ... el programa SÍ entrará en este if
    // porque 18 sí que es menor o igual que 18 (igual)
}

if (miEdad <= 19)
{
    // ... el programa TAMBIÉN entrará en este if
    // porque 18 sí que es menor o igual que 19
}
```

Mayor o igual >=

Compara si la variable de la izquierda es mayor o igual que la de la derecha.

```
// En este ejemplo miEdad es siempre 18...
int miEdad = 18;

if (miEdad >= 18)
{
    // ... el programa SÍ entrará en este if
    // porque 18 sí que es menor o igual que 18 (igual)
}

if (miEdad >= 19)
{
    // ... el programa NO entrará en este if
    // porque 18 no es menor ni igual que 19
}
```

> **Truco para el mayor o igual y el menor o igual.**
> Se escribe en el mismo orden en que se lee.
> - Mayor o igual //// > ó = //// >=
> - Menor o igual //// < ó = //// <=
>
> Y también un **truco para diferenciar cuál símbolo es el de mayor y cual el de menor**:
> - `A>B` A es mayor que B porque la parte grande del ángulo es la de la A
> - Y al contrario, `B<A` B es menor que A porque la parte pequeña del ángulo es la de B.

Operadores lógicos

Pero antes de continuar veamos las **abreviaciones booleanas**.

```
bool anilloEquipado = true;

if (anilloEquipado == true)
{
```

```
    // Do something
}
```

El código de arriba es equivalente a:

```
bool anilloEquipado = true;

if (anilloEquipado)
{
    // Do something
}
```

Negación !

El símbolo `!` se escribe antes de cualquier expresión o variable booleana para darle la vuelta a su valor. Si es `false`, la vuelve `true`; y si es `true`, la vuelve `false`. Se lee como "NO". Por ejemplo `!mayorDeEdad` se leería como "NO mayor de edad".

La tabla de la verdad es la siguiente:
A	!A
true	false
false	true

Por ejemplo

```
if (miEdad >= 18)
{
    soyMayorDeEdad = true;
}
else
{
    soyMayorDeEdad = false;
}

if (!soyMayorDeEdad )
{
    Console.WriteLine("¡Aún no soy mayor de edad!");
}
```

También puede usarse para darle la vuelta a una variable booleana en una sola línea. Por ejemplo, los siguientes códigos son equivalentes:

```
// Al pulsar el botón de agacharse,
// se levanta si estaba agachado
// o se agacha si estaba levantado.

if (botonAgacharsePulsado == true)
{
    if (agacharse == true)
    {
        agacharse = false;
    }
    else
    {
        agacharse = true;
    }
}
```

```
// Al pulsar el botón de agacharse,
// se levanta si estaba agachado
// o se agacha si estaba levantado.

if (botonAgacharsePulsado == true)
{
    agacharse = !agacharse;
}
```

Como puedes ver, el código anterior es mucho más corto, pero ambos hacen lo mismo.

AND &&

Para que sea `true`, ambas condiciones a la izquierda y a la derecha del símbolo `&&` tienen que ser verdaderas.

- Si cualquiera de ellas es falsa, la expresión es evaluada como falsa.
- Solo es verdadera cuando ambas son verdaderas a la vez.

La tabla de la verdad es la siguiente:

A	B	A && B
true	true	true
true	false	false
false	true	false
false	false	false

Por ejemplo:

```
if (soyMayorDeEdad && tengoCarnetDeConducir)
{
    puedoConducir = true;
}
else
{
    puedoConducir = false;
}
```

> **¿Entendéis ahora por qué son la utilidad de las abreviaciones booleanas?**
> Es mucho más fácil escribir
> `soyMayorDeEdad && tengoCarnetDeConducir`
> que
> `soyMayorDeEdad == true && tengoCarnetDeConducir == true`.

OR ||

Para que sea `true`, basta con que una de las condiciones sea verdadera, da igual cual sea. Puede ser la de la izquierda o la de la derecha del símbolo `||`.
- Si cualquiera de ellas es verdadera, la expresión es evaluada como verdadera
- Solo es falsa si ambas son falsas al mismo tiempo.

La tabla de la verdad es la siguiente:

A	B	A || B
true	true	true

A	B	A || B
true	false	true
false	true	true
false	false	false

Por ejemplo:

```
if (soyMayorDeEdad || voyAcompañadoDeUnAdulto)
{
    puedoEntrarAlConcierto = true;
}
else
{
    puedoEntrarAlConcierto  = false;
}
```

Bucles

Hay cuatro tipos de bucles principales: While, Do While, For y Foreach (ordenador de menos a más difícil, respectivamente). En los siguientes apartados estudiaremos los tres primeros, ya que para entender Foreach primero hay que entender conceptos más avanzados.

While

Mientras que una condición sea evaluada como verdadera, el código de dentro se repite. La notación es la siguiente:

```
while (condicion)
{
    // Do something
}
```

> ¡Cuidado! Si la condición no cambia nunca, el programa entra en un **bucle infinito** y se queda pillado.

Por ejemplo, queremos un programa que imprima un mensaje por pantalla desde que tenemos 4 años hasta que tenemos 18:

```
int miEdad = 4;

while (miEdad <= 18)
{
    Console.WriteLine("¡Tengo " + miEdad + " años!");
    miEdad++;
}
```

- Recuerda que si utilizamos `<=`, el último mensaje será `"¡Tengo 18 años!"`, mientras que si utilizamos `<`, el último mensaje será `"¡Tengo 17 años!"`.
- Recuerda también que dentro de los bucles puede haber ifs anidados también. ¡Y dentro de un bucle también puede haber otro bucle!

Do While

Es como el bucle While, pero la condición se evalúa después de haberse ejecutado una vez, **por tanto siempre se ejecutará como mínimo una vez**.

Su notación es:

```
do
{
    // Do something
} while (condicion);
```

Un ejemplo muy útil, y probablemente de las únicas utilidades a nivel básico, sería para obligar al usuario a introducir un dato correcto:

```
int miEdad;

do
{
```

```
    miEdad = Convert.ToInt32(Console.ReadLine());
} while (miEdad < 0);
```

For

Es equivalente al bucle While, pero su notación es un poco más compleja. Por eso, para principiantes, es recomendable empezar utilizando While.

Su notación es:

```
for (inicializar contador; condicion de parada; incrementar cont
{
    // Do Something
}
```

- **Inicializar contador**: aquí se inicializa la variable entera contador, que es el número de veces que el bucle se va a repetir. A esta variable, casi siempre se le llama `i`.
- **Condición de parada**: se hace esta comprobación cada vez que se repite el bucle, incluida la primera vez, para ver si se ha llegado al final. Casi siempre, esta comprobación es `¿Hemos llegado al final?`, que traducida en código sería `i < numeroDeRepeticiones`.
- **Incrementar contador**: en esta sección se le suma 1 a la variable contador, por lo que casi siempre escribiremos `i++`, para así pasar a la siguiente repetición. Pero, por ejemplo, si quisiéramos ir de cinco en cinco, escribiríamos `i+=5`.

Por ejemplo:

```
// Programa que cuenta hasta 10
for (int i = 0; i < 10; i++)
{
    Console.WriteLine("Vamos por la repetición número " + i);
}
```