Índice

[**Operadores aritméticos** 1](#_Toc144665111)

[**Operadores lógicos** 2](#_Toc144665112)

[**Operadores relacionales** 3](#_Toc144665113)

[**Strings en C#** 4](#_Toc144665114)

[**Cadenas y literales de cadena** 4](#_Toc144665115)

[**Cadena frente System.String** 4](#_Toc144665116)

[**Condicional IF** 4](#_Toc144665117)

[**Condicional Switch** 6](#_Toc144665118)

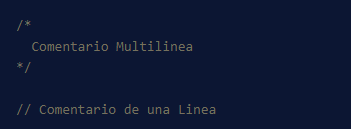
[**Ciclos While y Do While** 8](#_Toc144665119)

[**Ciclos For** 8](#_Toc144665120)

# **Operadores aritméticos**

Podemos comentar en el editor con **//**, lo que permite ingresar anotaciones para nosotros y otros colaboradores con los que estemos trabajando. Dichos comentarios no afectan a la ejecución del código y también pueden servir para sacar alguna porción de nuestro trabajo para que no se ejecute, pero sin necesidad de borrarlo

El doble slash sirve para comentar UNA línea, pero no si queremos hacer saltos de línea. En este último caso lo haremos con **/\* \*/**



Originalmente las computadoras se usaban para realizar cálculos muy bestiales por lo que tenemos a disposición multitud de signos que podemos usar para nuestros cálculos:

* Suma: +
* Resta: -
* Multiplicación: \*
* División: /
* Módulo o Resto (lo que te quedaba de una división): %
* Incremento (sería un +1): ++
* Decremento (-1): –

# **Operadores lógicos**

Los operadores lógicos en C# son utilizados para comparar y combinar valores booleanos. Los operadores lógicos disponibles en C# incluyen:

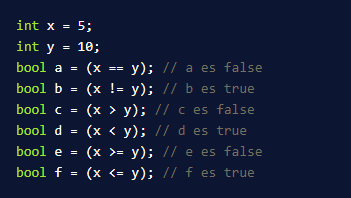
* **&&** (and): Devuelve verdadero si ambos operandos son verdaderos.
* **||** (or): Devuelve verdadero si al menos uno de los operandos es verdadero.
* **!** (not): Invierte el valor booleano del operando.
* **^** (xor): Devuelve verdadero si sólo uno de los operandos es verdadero.



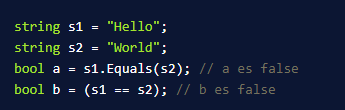
# **Operadores relacionales**

Los operadores relacionales en C# se utilizan para comparar valores y determinar si una relación es verdadera o falsa. Los operadores relacionales disponibles en C# incluyen:

* **==** (igual a): Devuelve verdadero si los operandos son iguales.
* **!=** (no igual a): Devuelve verdadero si los operandos son diferentes.
  + (mayor que): Devuelve verdadero si el operando de la izquierda es mayor que el de la derecha.
* **<** (menor que): Devuelve verdadero si el operando de la izquierda es menor que el de la derecha.
* **>=** (mayor o igual que): Devuelve verdadero si el operando de la izquierda es mayor o igual que el de la derecha.
* **<=** (menor o igual que): Devuelve verdadero si el operando de la izquierda es menor o igual que el de la derecha.



Es importante recordar que para comparar strings se utiliza el método **.Equals()** o el operador **==** y **≠**



# **Strings en C#**

## **Cadenas y literales de cadena**

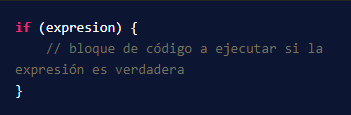
Una cadena es un objeto de tipo **String** cuyo valor es texto. Internamente, el texto se almacena como una colección secuencial de solo lectura de objetos **Char**. No hay ningún carácter que finalice en **NULL** al final de una cadena de C#, por lo tanto, la cadena de C# puede contener cualquier número de caracteres nulos insertados ("**\0**"). La propiedad **Length** de una cadena representa el número de objetos **Char** que contiene, no el número de caracteres Unicode. Para obtener acceso a los puntos de código Unicode individuales de una cadena, use el objeto **StringInfo**.

## **Cadena frente System.String**

En C#, la palabra clave **string** es un alias de **String**. Por lo tanto, **String** y **string** son equivalentes, aunque se recomienda usar el alias proporcionado **string**, ya que funciona incluso sin **using System;**. La clase **String** proporciona muchos métodos para crear, manipular y comparar cadenas de forma segura. Además, el lenguaje C# sobrecarga algunos operadores para simplificar las operaciones de cadena comunes.

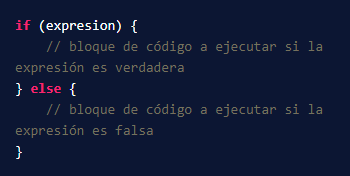
# **Condicional IF**

La sentencia **if** en C# se utiliza para evaluar una expresión lógica y ejecutar un bloque de código si la expresión se evalúa como **verdadera**. La sintaxis básica de la sentencia if es la siguiente:

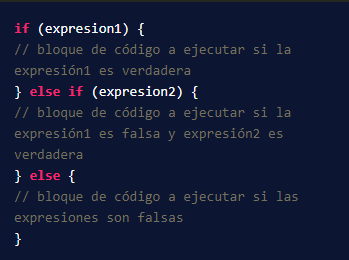


La condición puede ser cualquier expresión que evalúe a verdadero o falso, como una comparación de igualdad, una comparación de desigualdad, una comparación mayor o menor, etc.

También se puede agregar un bloque else opcional para ejecutar un bloque de código si la expresión se evalúa como falsa:

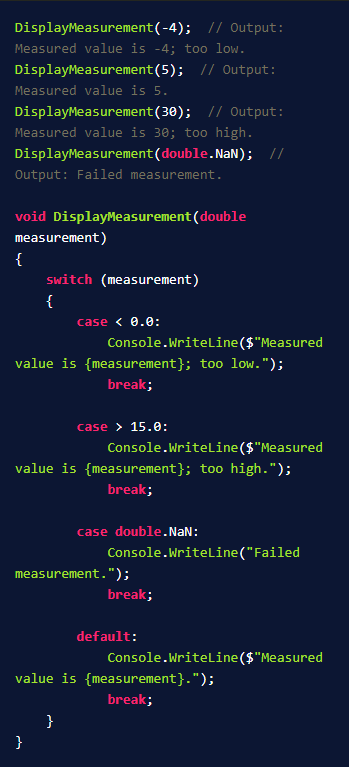


Además, se puede utilizar la sentencia else if para evaluar varias condiciones



# **Condicional Switch**

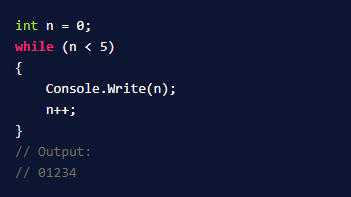
La instrucción **switch** selecciona una lista de instrucciones para ejecutarla en función de la coincidencia de un patrón con una expresión de coincidencia, como se muestra en el ejemplo siguiente:



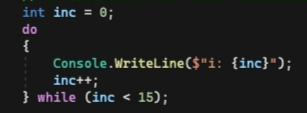
# **Ciclos While y Do While**

La instrucción **while** ejecuta una instrucción o un bloque de instrucciones **mientras** que una expresión booleana especificada se evalúa como **true**. Como esa expresión se evalúa antes de cada ejecución del bucle, un bucle while se ejecuta **cero** **o** **varias** **veces**. La while instrucción difiere de un bucle **do while**, que se ejecuta **una o varias veces**.

En el ejemplo siguiente se muestra el uso de la instrucción while:

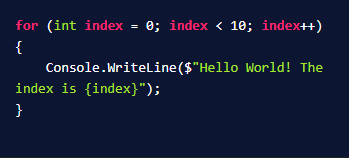


La estructura **do while** es otra estructura repetitiva, la cual ejecuta al menos una vez su bloque repetitivo, a diferencia del **while** o del **for** que podían no ejecutar el bloque.



# **Ciclos For**

La instrucción for consta de tres partes que controlan su funcionamiento.



* La primera parte es el inicializador de for: **int index = 0;** declara que index es la variable de bucle y establece su valor inicial en 0.
* La parte central es la condición de for: **index < 10** declara que este bucle for debe continuar ejecutándose mientras que el valor del contador sea menor que diez.
* La última parte es el iterador de for: **index++** especifica cómo modificar la variable de bucle después de ejecutar el bloque que sigue a la instrucción for. En este caso, especifica que index debe incrementarse en uno cada vez que el bloque se ejecuta.