Curso C# Nivel 1 (Desde Cero)

Software Necesario

Introducción

¡Hola!

Leete todo atentamente, pero tené en cuenta que toda esta info también la tenés explicada en videos, así que si algo lo ves medio confuso, no lo hagas y andá a ver el video de cada tema y si hay dudas, contame en el [Foro de dudas](https://campusmaxiprograma.com/mod/forum/view.php?id=714).

En este curso vamos a programar con un lenguaje de programación llamado **C#** (C sharp). El lenguaje fue desarrollado por **Microsoft** y tiene una forma de ser escrito bastante parecida a lo que es **C++** o **Java**.

Visual Studio Code

Para poder programar en **C#** necesitamos contar con toda la suite de desarrollo de **Microsoft** que se instala automáticamente al descargar e instalar un **IDE** que se llama **Visual Studio Community**. El tema es que este programa es súper pesado y no tiene mucho sentido, por ahora.

Así que lo que vamos a usar es su hermano menor, que no es menos importante y de hecho es uno de los más famosos hoy en día. Se llama **Visual Studio Code**. No confundan el nombre, es muy parecido, pero son dos cosas bastante distintas.

El tema es que el **Visual Studio Code** no te instala todo automáticamente. Pero no pasa nada, es instalar una sola cosa aparte.

Entonces, lo primero que tenés que hacer es descargar e instalar:

Descargar: Visual Studio Code

SDK .Net

Luego de que hayas instalado el VS Code, vamos a descargar e instalar el **Software Development Kit** de **Microsoft** **.Net**. Ufff... cuánto nombre... es básicamente un kit de desarrollo que tiene todo lo que necesitamos, no te asustes.

SDK .Net

**Nota:**para poder instalar todo esto necesitás contar con Windows. Si no usás Windows, contame que lo podemos adaptar para el sistema que uses, de hecho en el video en el que explico la instalacion del SDK te cuento qué descargar para otros Sistemas Operativos. Cualquier duda, contame.

Crear nuevo proyecto

De esto también hay un video en el que te lo explico paso a paso, pero si querés ir intentando, te dejo los pasos a seguir para poder crear un nuevo proyecto.

Lo primero que necesitamos hacer es abrir la terminal, o **consola de comandos**. En **Windows,** para ello podés presionar la combinación de teclas **Windows + R** y en el cuadro de diálogo escribir "**cmd**" y presionar **Enter**. Otra forma es buscando la consola en el buscador del menú de **Windows**.

**Nota:** si tenés MacOS o Linux, tendrás que abrir la Terminal, el resto es igual.

Una vez allí lo que tendrás que hacer es posicionarte en una carpeta. O sea entrar a una carpeta en la que vamos a crear el proyecto. Para ello te recomiendo que vayas al escritorio o a **Documents** y te crees una carpeta que se llame "Práctica Programación" o algo por el estilo. Allí dentro crearás luego todos tus proyectos.

Supongamos que en **Documents** creaste la carpeta "Ejercicios", entonces en la consola deberías escribir:

**cd C:\Users\msarfernandez\Documents\Ejercicios**

O sea, la clave "**cd**" seguida por **un** **espacio** y luego la ruta completa de la carpeta. En el ejemplo es la ruta completa en **MI** máquina, vos tenés que poner la ruta completa en la tuya.

Una vez que logres eso vas a ver algo así:

Eso quiere decir que ya estás dentro de la carpeta.

Ahora lo que tenés que poner es lo siguiente:

**dotnet new console -o nombreproyecto**

Esto va a crear una nueeeva carpeta adentro de tu carpeta Ejercicios con el nombre "nombreproyecto" o el nombre que le pongas.

Ahora abrís Visual Studio Code y lo que hacés es simplemente arrastrar la nueva carpeta al Visual Studio Code. Se te va a abrir el proyecto y deberías ver algo así:

Bueno, falta poquito. Parece largo pero vas a ver que luego de hacerlo dos o tres veces, ya está.

Volvemos a la consola y escribimos:

**cd nombreproyecto**

Esto nos deja DENTRO de la carpeta del proyecto que corresponda, acordate que en mi ejemplo se llama así... qué ejemplo, no?...

Bueno, una vez allí, ahora escribimos:

**dotnet restore**

y presionamos enter. Y luego:

**dotnet run**

Tardará unos segundos y vas a ver en pantalla la frase "Hello World".

Uff, listo, ejecutaste tu primer programa.

La versión de .NET

¡Hola!

Si estás acá es porque probablemente sigas con dudas sobre ***cuál versión instalar***.

¿Y por qué tendrías esa duda si Maxi dice que instalen **NET 5** y listo?

La cuestión es que cuando vos querés instalar **NET 5**, la web de **Microsoft** te dice que ya no tiene más soporte. Entonces es lógico que a raíz de este mensaje en rojo que te pone la web, te surjan dudas...

"pero qué pasa si no tengo más soporte de **Microsoft**"?

***La realidad es que no pasa nada...***

En este nivel lo que vamos a hacer es **aprender a programar**, los fundamentos, las bases; y da igual que lo hagamos con NET 5, 1, 7, Net Core, o lo que sea. No tenemos que hacernos problema por eso por ahora, es algo que ya charlaremos más adelante (en el Nivel 2) para comprender cuáles son las diferencias.

Sin embargo... acá seguimos con algo por resolver, y es el tema de que se vea igual la estructura de los programas que ustedes crean a la que crea Maxi, y eso es importante.

Entonces, ¿Qué instalo?

1. Si instalás **NET 5** más allá del mensaje de advertencia, ya estaríamos.

2. Si querés, luego de instalar**NET 5** podés actualizar e instalar la 6, y luego la 7, y así (sale una cada año). Pero acá lo que va a suceder es que tendrás que usar otra línea de comando para crear los programas. En realidad es la misma, pero tenés que agregarle la petición para que lo haga con la plantilla (la estructura) de **NET 5**. Y eso se hace como te expliqué en la clase de "Mi código se ve diferente", pero te dejo acá la línea:

**dotnet new console -o tu-programa --framework net5.0**

3. Hay una alternativa y es que instales directamente NET 6 o NET 7 (o superior); en ese caso, cuando quieras usar la línea anterior para crear el proyecto te va a fallar, pues no tenés instalado NET 5. La solución?

a. Instalar **NET 5**. Esto te permitirá invocarlo para crear los proyectos.  
b. Usar la siguiente línea para la creación del proyecto:

**dotnet new console -o tu-proyecto --use-program-main**

De esta forma podemos usar cualquier versión de la **SDK** y al agregar el parámetro "**--use-program-main**" trabajará con la versión larga de la plantilla; la que vemos en todas las clases.

--

Espero que este pequeño artículo te sea de utilidad.

Cualquier duda no dejes de consultar.

Saludos!

1. Etapa inicial

Introducción

La programación es una disciplina (de tantas hoy en día) que pertenece a la informática, ciencia que se dedica al análisis y transformación de datos e información a partir del uso de computadoras.

En dicha disciplina se desarrollan algoritmos en un determinado lenguaje de programación, los cuales serán implementados en una computadora para de esa manera realizar los llamados programas.

Algoritmo

Un **algoritmo** es una secuencia finita y ordenada de **instrucciones** que deben ser seguidas en pos de resolver un problema.

Programa

Un **programa** es, técnicamente hablando, uno o más (al menos uno) **algoritmos** de programación escrito en un lenguaje que puede ser interpretado y ejecutado por una computadora. En otras palabras, decimos que un **programa** es la solución a un problema, ya que cada vez que se requiere desde sumar números hasta realizar traducciones, grandes cálculos o enviar mensajes de texto, se recurre a un programa (o aplicación, o web, lo que corresponda).

Existen distintos tipos de programas o aplicaciones (podemos decir que son sinónimos), por ejemplo:

-       **Programas de escritorio:** Se ejecutan en windows, por ejemplo Word, Excel, el buscaminas.

-       **Programas web:** Facebook, Mercado Libre, son aplicaciones o sistemas web.

-       **Programas de celular:** también llamados apps, o aplicaciones, por ejemplo Whatsapp, Instagram, Google Maps.

Todos estos son parte de un gran grupo dentro de la informática que se denomina **Software**.

Sistema Informático

El sistema informático está compuesto necesariamente por tres partes:

-       Hardware

-       Software

-       Humano

Ante la ausencia de alguna de éstas partes, no podríamos completar el sistema y este no tendría sentido dado que:

**Software** es todo lo que creamos para que una computadora funcione, pero requiere justamente una computadora donde hacerlo, ahí entra el Hardware.

**Hardware** es “todo lo que se puede tocar” en una computadora: el monitor, el teclado, incluso todos los componentes internos dentro de la misma.

Sin embargo, sin programas para utilizar, todo ese poder de procesamiento, no nos sería de utilidad.

**Humano.**Se completa el ciclo con el humano que es quien hace uso de ese Software gracias a ese Hardware, y quizá pueda sonar a una obviedad, pero es clave tener en cuenta este círculo.

Como ya dijimos, el Software es lo que le da vida a la computadora. Son los programas que nos permiten utilizarlas, y existen distintas categorizaciones dentro del software. Las más comunes son:

-       **Software de base:** esto corresponde a las cuestiones básicas y de sistema operativo que se requiere para poder encender y usar una computadora (Windows, Linux, Mac OS, Android, iOS).

-       **Software utilitarios:** son aplicaciones complementarias para el correcto funcionamiento y/o administración de los sistemas. Podemos hablar del desfragmentador de disco, o utilidades de análisis de errores en memoria, entre otros.

-       **Software de aplicación:** los programas, sistemas o aplicaciones que utilizamos a diario.

Existen más categorías y sub-categorías y además ciertas características adicionales como si son libres, o no, o si son open source, o multiplataforma, pero eso es para indagar en otro contexto.

También podemos mencionar los “***programas para programar***”. Desde ya que hoy en día para poder desarrollar programas, vamos a necesitar una computadora, pero además vamos a necesitar uno o más (según corresponda) programas que nos sirvan para ello. Hay aplicaciones específicamente preparadas para, justamente, desarrollar otras aplicaciones, en definitiva, para programar. Estas aplicaciones se suelen conocer como **IDE**, de sus siglas en inglés "***Entorno de Desarrollo*Integrado**".

Dependiendo de lo que quisiéramos programar, será la aplicación que deberíamos utilizar, y esto aplica del mismo modo para la tecnología o **lenguaje de programación** a utilizar.

Tipos de datos

Como veremos más adelante, al programar no solo hay que escribir en un lenguaje determinado, sino que vamos a tener que utilizar muchas herramientas distintas, dentro de la propia programación, que iremos adquiriendo a medida que vayamos avanzando. Una de esas herramientas y de la que más vamos a hablar y utilizar de entrada (y para siempre si seguís con la programación) es la variable.

Una variable, técnicamente hablando, un espacio en la memoria de la computadora destinado a ser utilizado por nuestro programa para poder guardar un dato. La variable tiene ciertas características como que debe contar con un nombre, que puede guardar un solo dato a la vez (el mismo puede ir cambiando, de allí su nombre) y que debe tener un tipo. Este tipo se refiere a qué es lo que precisamente va a guardar esa variable. Los primeros tipos que aprendemos son los más basicos y elementales pero que nos acompañarán, junto a las variables, en toda la vida dentro de la programación. Estos tipos son:

-       Int para guardar números enteros.

-       Float o double (dependiendo del lenguaje, hay otras variantes) para guardar números con coma.

-       Char para guardar letras.

-       Bool para guardar verdadero o falso (que es lo mismo a 1 o 0 respectivamente.

Que le asignemos un tipo de dato a una variable afecta y tiene una relevancia puntual. Si estoy creando una variable de tipo entera, la misma solamente podrá guardar valores enteros. Qué sucederá si quiero guardarle un número con coma? Bueno, dependiendo del momento, el lenguaje o la operación, podrá dar error o bien podrá redondear el valor quitando los decimales y quedándose solo con la parte entera. Ninguno de los escenarios serían pertinentes en nuestro programa. Lo ideal sería que si la variable solo admite valores enteros, pues que se la use con valores enteros.

2.Primer programa

En este módulo comenzamos a manejar conceptos más precisos y técnicos, puntuales y necesarios para la confección de nuestros primeros programas.

En la unidad anterior comprendimos las herramientas a utilizar y llevamos a cabo la creación del primer proyecto para corroborar que todo estaba en orden.

Lo primero que tenemos que comenzar a comprender en cuanto a los programas es que existe un principio básico de todo algoritmo, de todo programa, de toda aplicación; el cual se basa en el esquema que se muestra a continuación:

Todo programa, para poder operar, necesita contar con una entrada de datos con los cuales realizará un posterior procesamiento, una transformación de esos datos, a partir de los cuales terminará obteniendo información para finalmente brindar una salida.

Nuestro ejemplo más sencillo parte de la necesidad de sumar dos números. Para poder sumar dos números, el programa deberá solicitarlos (o recibirlos), es decir, contar con una entrada de datos que serán esos dos números. Luego aplicará un procesamiento, que en este caso será sumar esos dos valores que fueron ingresados; para finalmente obtener el resultado de la suma y mostrarlo en pantalla. Esta última sería la salida de información.

¿Dato o Información?

Por qué se hace hincapié en la diferencia de que lo que entra son Datos y lo que sale es Información? Podríamos tomarlos como sinónimos?

La definición de datos dice que: un dato es un conjunto de símbolos sin significado alguno, mientras que por otro lado, la definición de información afirma que: la información está compuesta por símbolos a los que se le ha atribuido algún significado.

Yo siempre suelo dar un ejemplo un tanto fantástico, pero simple y que, a mí entender, funciona para comprender la idea.

Se trata de una persona que, caminando por la calle, se encontró con un pergamino tirado. Cuando lo abre se trata de un documento bastante elegante, con firmas y demás pero con un texto que esta persona no podía comprender. Pensó en hacer un cuadro, ya que era lindo, vistoso; terminó por regalárselo a un amigo hijo de japoneses. Este amigo resulta que manejaba la lengua nativa de Japón por sus padres y al ver el pergamino entendió de qué se trataba y no solo eso, pudo leerlo pues estaba escrito en un lenguaje que manejaba y comprendía. Se trataba de un documento correspondiente a un testamento el cual otorgaba una fortuna a quién lo encuentre y, desde ya, pueda reclamarlo por saber de qué se trataba.

A qué quiero llegar con todo esto? Para la primera persona ese pergamino no era más que un papel "lindo" que a lo sumo podría utilizar para decorar una sala. Un papel con un conjunto de símbolos sin sentido. Pero por otro lado, para su amigo, quien contaba con los elementos necesarios para comprender ese papel, lo que decía en él, par aplicar un procesamiento (a partir de traducir -aplicar un algoritmo-) el contenido del mismo, pasó a atribuirle un significado valiosísimo, en este caso monetario, transformando esos datos en información, mediante un procesamiento.

El fin de toda aplicación es siempre obtener información.

El proceso y las variables

Para poder realizar las transformaciones de las que hablamos necesitamos construir nuestros algoritmos con las instrucciones necesarias. Desde ya que tendremos, generalmente para comenzar, un ingreso de datos y para terminar una salida de información. Pero en el medio es donde vamos a tener que establecer las instrucciones necesarias para darle forma al proceso pertinente que realice la transformación, o "la magia".

Volviendo a nuestro ejemplo de la suma, el proceso sería justamente sumar, valga la redundancia, pero hay varias cosas que se encuentran implícitas en esa operación.

Para poder sumar dos valores necesitamos previamente contar con ellos, y para eso no alcanza con pedirlos. Una vez que los pedimos, tenemos que guardarlos. Para poder guardar esos datos vamos a necesitar hacer uso de un elemento fundamental en la programación: la variable. Una **variable** es un espacio en memoria que permite alojar un valor. Es un medio que utilizaremos en nuestros programas para ir guardando todas las cosas que necesitamos dentro y a lo largo de todo **algoritmo**.

Imaginen la memoria **RAM** (si no conocen la memoria RAM repasen el video de la PC del módulo 1) como una hoja excel, la cual cuenta con columnas representadas por letras y filas representadas por números y en cada intersección encontramos una celda.

Ok. Si yo digo **C5**, estoy haciendo referencia a la **columna C** en la **fila 5**. Podemos decir que la memoria RAM trabaja parecido a esto. C5 sería la **dirección de memoria** de esa celda. Sin embargo, al querer usar esas "***celdas" (variables)*** en nuestros programas no vamos a estar todo el tiempo usando direcciones de memoria, para ello, lo que haremos es ponerle nombre a las variables. De ese modo, podremos hacer referencia a esas celdas mediante un nombre significativo y fácil de recordar. Significativo porque deberíamos nombrar nuestras variables teniendo en cuenta qué es lo que contienen dentro.

Volviendo una vez más a nuestro ejemplo de la suma, para guardar los dos valores ingresados podemos decir que lo haremos en las variables **numero1** y **numero2**. Los nombres de las variables pueden contener cualquier letra, varias palabras y también números, aunque hay ciertas restricciones:

* No se puede comenzar el nombre de una variable con números.
* El nombre no puede estar compuesto solo por números.
* El nombre no puede contener símbolos especiales (por ejemplo \* o +).

El nombre no puede ser igual a una palabra reservada. Dependiendo del lenguaje de programación que usemos, hay palabras que son específicas para hacer algo puntual, con lo cual no podemos usarlas como nombres de variables. Por ejemplo:

Console

Palabra que usamos para invocar a la muestra de un mensaje en pantalla como vimos en el primer proyecto de ejemplo:

Console.WriteLine("Hola Mundo");

Operadores Aritméticos

Son los primeros operadores que comenzaremos a utilizar. Estos son:

* + para sumar
* - para restar
* \* para multiplicar
* / para dividir

Estas son las operaciones básicas que el procesador ya conoce y que tenemos disponibles para utilizar. De allí para arriba, es decir, un cuadrado, un logaritmo, etc., etc., son operaciones que tendremos que construir a mano porque el procesador no las sabe. Es verdad que en la actualidad existen muchas cosas ya hechas las cuales podemos utilizar (se llaman librerías, las veremos más adelante), pero generalmente cuando se está comenzando en la programación, una de las primeras cosas se trata de recrear algunas de esas funcionalidades para entender cómo trabajan.

Asignación

La asignación es una acción mediante la cual se da un valor a una variable. Por ejemplo

jota = 10;

Lo que estamos haciendo en esa instrucción es tomar el valor 10 y asignarlo, o sea guardarlo, en la variable llamada jota. Hay varios orígenes de datos para realizar una asignación. La que acabamos de ver directamente asigna un valor. También se puede asignar el resultado de una operación matemática o simplemente asignar lo que se pide que una persona ingrese por teclado:

resultado = jota + 7;  
variable = Console.ReadLine();

El primer programa

Ya sabemos que tenemos que pedir y guardar, y que tenemos que procesar y mostrar. Y tenemos casi todas las herramientas. Hagamos nuestro primer programa en código y luego terminamos de analizar.

int numero1;  
int numero2;  
int resultado;  
numero1 = int.Parse(Console.ReadLine());  
numero2 = int.Parse(Console.ReadLine());  
resultado = numero1 + numero2;  
Console.WriteLine(resultado);

En las primeras tres líneas lo que hacemos es declarar las variables y darles un tipo. Trabajaremos con números enteros (recordemos que los tipos que manejaremos por ahora son int, float, char y bool).

Luego pedimos datos por teclado con la función **ReadLine**. La misma devuelve un texto, es por eso que con **int.Parse** lo que hacemos es transformarlo a número, que es lo que queremos. Tranqui, hay cosas que de a poco vamos a ir terminando de entender.

Luego de pedir y guardar los datos en nuestras variables, volvemos a usar éstas últimas ahora para leer el contenido que antes guardamos y, haciendo uso del operador **aritmético para sumar**, realizamos la suma. El resultado lo guardamos con el **operador =** (que en programación no es igual, sino **asignar**).

Finalmente, con **WriteLine** volvemos a leer, esta vez el valor que quedó guardado en ***resultado***, para mostrarlo en pantalla.

Y listo, ese es nuestro primer programa. Todos estos pasos que tenemos detallados aquí, los tenés súper explicados en el video "**Primer Programa**" que te sugiero que veas.

Luego de todo esto, ya estamos en condiciones para seguir avanzando en la creación de otros programas de este tipo.

Tenés algunos ejercicios para practicar.

[Tipos HYPERLINK "https://campusmaxiprograma.com/mod/url/view.php?id=746" HYPERLINK "https://campusmaxiprograma.com/mod/url/view.php?id=746"de dato INT](https://campusmaxiprograma.com/mod/url/view.php?id=746)

<https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/integral-numeric-types>

Tipos de dato FLOAT

<https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/floating-point-numeric-types>

3. Condicionales

Condicionales

A ésta altura ya hemos desarrollado nuestros primeros programas. Los mismos se encargan de realizar operaciones matemáticas sencillas solicitando datos a un usuario y luego mostrando un resultado en pantalla. Las operaciones no tienen por qué ser sencillas, vimos que podemos combinar los operadores aritméticos y hacer uso de paréntesis (como en matemática) para separar en términos y demás con el fin de llegar a armar operaciones un poco más complejas que una simple suma.

Sin embargo, todos estos programas siguen siendo lineales, tengan tres líneas de código o cien. Quiere decir que comienzan por la primera instrucción y recorre absolutamente cada línea ejecutando todo lo que se encuentre en su camino hasta concluir.

Existen momentos en los que vamos a querer ejecutar instrucciones bajo determinados escenarios; en determinadas circunstancias o en situaciones específicas exceptuando de ese modo un escenario alterno. Por ejemplo: calcular y mostrar el resultado de una suma si el primer número ingresado es mayor a 10.

Ya hicimos programas que piden y suman dos números, pero ahora solamente hay que sumarlos si el primer ingreso es mayor a 10, de otro modo, no hay que hacerlo.

Para esto, y muchas otras cosas, existe una herramienta en programación denominada "**If**", del idioma inglés "**si**". Se trata de una herramienta que nos permitirá confeccionar condiciones (realizar preguntas) que tendrán como resultado posible **SI** o **NO**y dependiendo de cuál sea la respuesta podremos programar distintos sets de instrucciones.

Una vez que se establece una condición **If**, lo que sucede es que se bifurca, se divide el flujo de ejecución del programa, creando, por así decirlo, dos "dimensiones" o caminos alternativos. El programa deberá elegir uno de esos caminos a seguir, no ejecutará los dos al mismo tiempo ni primero uno y luego el otro. No. Se evalúa la **condición**, y si el resultado es **SI** (recordemos que los posibles son **SI** o **NO**, o **true** o **false**, o **1** o **0**), entonces se ejecutará el set de instrucciones programado en ese camino; pero si la respuesta a la pregunta es **NO**, entonces se ejecutará el otro set de instrucciones (en caso de haberlo, el **NO** puede no tener nada para ejecutar).

Ejemplo en código

Por otro lado, podemos ver el código del ejemplo antes mencionado;

int a, b, c;  
Console.WriteLine("ingrese un nro");  
a = int.Parse(Console.ReadLine());  
Console.WriteLine("ingrese un nro");  
b = int.Parse(Console.ReadLine());  
If(a > b) {  
   c = a + b;  
   Console.WriteLine("El resultado es: " + c);  
}  
Console.ReadLine();

Explicado paso a paso:

* Declaramos las variables.
* Mostramos un cartel para pedir el primer número.
* Leemos y guardamos el primer número.
* Mostramos otro cartel para pedir el siguiente número.
* Leemos y guardamos el segundo número.
* Establecemos la condición IF. En la misma preguntamos si el contenido de la variable "a" es mayor a 10.
* Si es mayor a 10, se ejecutará el contenido siguiente encerrado entre llaves, sino no entra a ese set y sigue de largo.

Algunas consideraciones a tener en cuenta:

**Formato de las condiciones**

Las condiciones están formadas, inicialmente, por tres partes:

* Un valor.
* Un operador de comparación.
* Otro valor a ser comparado.

En este caso el primer valor corresponde al contenido de la variable "a", mientras que el operador de comparación es ">" (mayor) y finalmente el segundo valor es el literal "10".

**Operadores relacionales**

Los operadores disponibles para construir condiciones lógicas se denominan relacionales, y son los siguientes:

* > mayor
* < menor
* >= mayor o igual
* <= menor o igual
* == igual (nótese que no es = a secas, son dos símbolos juntos para la igualdad)
* != distinto

Por ahora, solo podemos poner una condición por If. Es decir, una sola "**valor condición valor**"; con lo cual, deberías tener solamente dos valores y UN operador relacional, para que se entienda que es correcta la condición.

Ejemplos

**Ejemplos correctos**

* A > 10
* B < 12
* A != B
* C == 20

**Ejemplos incorrectos**

* A > B < 10 en este caso tenemos dos operadores y tres valores. No es correcto.
* 10 < 15 en este caso tenemos dos literales, no tiene sentido la condición pues siempre va a dar verdadero.

Si se te ocurren otros ejemplos y no sabés si son correctos, podés postearlos en el [Foro de dudas](https://campusmaxiprograma.com/mod/forum/view.php?id=714) y los vemos.

¿Verdadero o Falso?

Hasta ahora vimos como hacer una condición y trabajar si la misma da como resultado true o no hacer nada si la misma da como resultado false. Por ejemplo:

int edad = 21;  
if(edad > 18){  
   Console.WriteLine("Sos mayor");  
}

En este caso muestra un cartel en pantalla si edad tiene un valor mayor a 18 y en caso contrario, no hace nada. Pero qué pasa si sí quisiéramos hacerlo? En ese caso lo que podemos hacer es escribir el bloque else, que, como te habrás dado cuenta, es opcional, ya que hasta ahora no lo habíamos hecho.

De este modo, siguiendo el mismo ejemplo, podemos mostrar un cartel que diga que sos menor si el valor no es mayor a 18, o sea, si la condición fue falsa.

int edad = 17;  
if(edad > 18){  
   Console.WriteLine("Sos mayor");  
}  
else{  
   Console.WriteLine("Sos menor");  
}

En este caso, como ahora a edad le asignamos el valor 17, en vez de ir por el **verdadero** del **If**, va a ir por el **else**, o sea, el **falso**. Tené en cuenta que le asignamos 17 a la variable. Y qué pasa si le hubiésemos asignado 18? En ese caso qué resultado daría el **If**, **verdadero** o **falso**? Teniendo en cuenta *lo que se pregunta específicamente* en el **If** y no lo que sepamos de cuándo sos mayor de edad. Si tenés dudas sobre cuál es la respuesta correcta podés probar el código o consultarlo en el [Foro de dudas](https://campusmaxiprograma.com/mod/forum/view.php?id=714).

4. Condicionales ++

En este módulo vamos a ver varias herramientas complementarias a la hora de trabajar con condicionales.

Ya sabemos cómo trabaja el If y cuál es el modo correcto de construir condiciones lógicas a partir del uso de operadores relacionales.

Sin embargo, lo que vimos hasta ahora es que cada If puede tener una única condición y que si, por ejemplo, quisiéramos realizar dos preguntas, lo tenemos que hacer en dos Ifs separados.

Bueno, hay una herramienta que nos brinda la posibilidad de crear más de una condición en un mismo If, y esa herramienta se llama: operador lógico.

Operadores lógicos

Existen dos operadores lógicos, también conocidos como operadores de concatenación lógica.

* OR cuyo símbolo corresponde a || (doble pipe).
* AND cuyo símbolo corresponde a && (doble ampersand).

AND

El and es el operador "y". Al usar este operador para concatenar dos o más condiciones lógicas, el comportamiento será el siguiente:

si todas las condiciones dan verdadero, entonces el If dará como resultado verdadero.

si una condición da falso, entonces el If dará como resultado falso.

Esto se debe a que al usar el operador AND para concatenar las condiciones, lo que estamos indicando es que queremos que se consulta si se cumple la condición 1 Y se cumple la condición 2 Y se cumple la condición 3 Y.... etc. Estamos buscando saber si se cumplen todas; con lo cual, si se cumplen todas, dará verdadero PERO, con UNA condición que no se cumpla, entonces todo el If dará como resultado falso.

OR

Con el operador "o" pasa exactamente lo contrario. Lo que se busca es el verdadero si se cumple la condición 1 o se cumple la condición 2 o se cumple la condición 3 o... etc. En el caso del operador OR, el resultado será verdadero cuando cualquiera de las condiciones dentro del If de como resultado verdadero. Ahora, cuándo el If se irá hacia el falso? Cuando todas las condiciones den falso.

NOT

Existe un tercer operador lógico que es el operador de negación, cuyo símbolo es el "!" (signo de admiración). Este operador sirve para invertir (para negar) el resultado de una condición lógica. De este modo, si la condición diera verdadero, el Not haría que se vaya para el falso.

Por ejemplo:

int a = 10  
if(!a==10){  
   Console.WriteLine("no es igual a 10");  
}  
else{  
   Console.WriteLine("ES igual a 10");  
}

Contadores

Un contador no es más que una variable que vamos a usar para realizar un conteo de algo. Por ejemplo: cuántas personas son mayores de 18, o cuántas personas se sacaron un 10 en la materia, etc.

Todo conteo comienza desde un valor conocido, generalmente desde cero; es por eso que un de las prácticas a tener en cuenta a la hora de contar es inicializar la variable en cero. Inicializar significa dar un valor inicial. De ese modo nos garantizamos que el conteo arranca desde un valor determinado.

Contar es agregarle una unidad a una variable. Esto se puede hacer del siguiente modo:

int a = 0;  
a = a + 1;  
a = a + 1;  
a = a + 1;  
Console.WriteLine(a);  
Console.ReadLine();

Si se observa, en este caso el valor de **"a"** comienza en cero y luego se realiza tres veces una suma de una unidad sobre la misma. Esto realiza un efecto de conteo ya que se va sumando de a uno. Al final, al mostrar el contenido de **"a"**, el mismo será 3.

La idea de estos conteos es realizarlos, como se mencionó antes, de manera condicional, de modo tal que antes de contar se realizaría una comparación para identificar si hay que contar o no.

Por ejemplo: evaluar tres edades y mostrar por pantalla cuántas son mayores a 18.

int a = 16, b = 20, c = 33;  
int con = 0;  
if(a > 18){  
   con = con + 1;  
}  
if(b > 18){  
   con = con + 1;  
}  
if(c > 18){  
   con = con + 1;  
}  
Console.WriteLine("La cantidad de mayores es: " + con);  
Console.ReadLine();

En este caso, el resultado por pantalla sería 2.

Operador de incremento

Existe un operador que nos permite simplificar el conteo aplicado a una variable. Ese operador se denomina **operador de incremento**, cuyo símbolo se representa con un doble más: **++**

Con este operador, pasamos de tener

con = con + 1

a tener simplemente

con++

El **++** lo que hará es incrementar una unidad a cualquiera sea la variable y contenido que tenga a su izquierda. También existe su contraparte que corresponde a **"--"** (menos menos) que lo que hace es exactamente lo opuesto: disminuir una unidad de la variable a la que se asigne.

Nótese que en este caso no aplica el uso de la asignación (operador =), ya que la misma se desarrolla de manera implícita.

Acumuladores

Un acumulador funciona de manera similar a un contador. La diferencia es que en este caso no se suma de a una unidad sino que se suman distintos valores y lo que escriba, se irá agregando a lo ya existente. Por ejemplo:

int a = 0;  
a = a + 10;  
a = a + 3;  
a = a + 5;  
Console.WriteLine(a);  
Console.ReadLine();

En este caso el resultado en pantalla será 18, la suma de todos los valores.

De la misma manera que el conteo, los acumuladores se suelen usar condicionalmente; y también contamos con un operador especial para resumir la instrucción. El operador de acumulación: **+=.**

De este modo, pasamos de tener:

a = a + 10;

a tener

a+=10;

Son instrucciones que hace exactamente lo mismo, pero de una manera más compacta de escribir. También existe el operador para disminuir, y el mismo es el **"-=".**

El Switch

El switch es una herramienta de decisión que nos permite evaluar el contenido de una variable u operación y, a diferencia del If que nos permite solo una de dos opciones posibles (verdadero o falso), nos permite setear múltiples opciones posibles para el valor que se esté evaluando. Desde ya que una vez decidido el valor, se ejecutará un solo camino como también sucede con el If.

La diferencia principal, además de la cantidad de opciones, es que e switch no hace uso de los operadores relacionales. Simplemente se evalúa un valor y se elige un camino posible dependiendo de lo que se haya evaluado.

Por ejemplo: haremos un switch que reciba una variable y si la misma contiene un 1, mostrar Hola en pantalla, si tiene 2, mostrar Chau, si tiene 3 mostrar Blanco y si tiene 4 mostrar Negro.

int a = 3;  
Switch(a){  
case 1:   
Console.WriteLine("Hola");  
break; case 2: Console.WriteLine("Chau"); break; case 3: Console.WriteLine("Blanco"); break; case 4: Console.WriteLine("Negro"); break; default: Console.WriteLine("Opción incorrecta"); break;

} Console.ReadLine();

Este programa muestra por pantalla "Blanco", como pueden ver (también pueden copiar y probar el código en ejecución). Pueden ver dos particularidades: por un lado, contamos dentro de cada "case" el valor posible. En este caso cada "case" es cada brazo del Switch. Pero además, cada case al final cuenta con un break, que determina el final del mismo. Por otro lado, contamos con un último "case" denominado "default", que es por donde pasará la ejecución si el valor contenido en la variable que está siendo analizada no se corresponde con ninguna de las opciones seteadas.

5. Ciclos

Ciclos

Un ciclo es una herramienta de la programación que nos permite repetir la ejecución de cierta instrucción o cierto conjunto de instrucciones tantas veces como necesitemos.

Es decir, al día de hoy, si quisiéramos mostrar un cartel "Hola mundo" en la pantalla 5 veces, podemos, pero tendríamos que hacer lo siguiente:

Console.WriteLine("Hola mundo"); Console.WriteLine("Hola mundo"); Console.WriteLine("Hola mundo"); Console.WriteLine("Hola mundo"); Console.WriteLine("Hola mundo");

Es decir, escribir cinco veces la misma línea de código. Con un ciclo (también conocidos como bucles) puedo programar escribir esa instrucción una sola vez pero que se ejecute cinco veces. O las veces que necesite.

Ciclo For

Hay distintos tipos de ciclos, vamos a comenzar con el ciclo exacto: el ciclo FOR.

El FOR es un ciclo que nos permite ejecutar un conjunto de instrucciones una cantidad de veces establecida. Por ejemplo cinco. Por ejemplo diez. De allí su calificación de "ciclo exacto", porque dará exactamente la cantidad de vueltas que esté configurado para dar.

Para configurar un ciclo FOR vamos a necesitar contar con una variable, generalmente se usa como nombre para dicha variable las letras "x", "y" o "z" aunque también encontrarán muy frecuentemente que la variable del FOR se denomina "i" (de "iterar", que significa ciclar, o dar una vuelta al ciclo).

Una vez con la variable hay tres factores que se requiere configurar en un ciclo:

* Inicialización.
* Condición.
* Incremento.

Cada una de estas acciones ya son conocidas por separado, pero en este caso van a estar trabajando en conjunto para poder configurar lo que se denomina "reloj" del ciclo y que será lo que permita el funcionamiento del mecanismo del mismo.

Inicialización

Se trata de dar un valor inicial a la variable del ciclo. Esta instrucción se ejecutará una sola vez en toda la vida del ciclo y es para darle un punto de partida al mismo. Se puede inicializar en cualquier valor, incluso en el valor de otra variable, aunque en una configuración regular, se suele incializar simplemente en cero.

Condición

La condición es la instrucción que determinará si se sigue iterando o no. Se ejecuta previo a comenzar cada vuelta del ciclo y si da verdadero, se ejecuta la vuelta. Cuando sea falso, se dará por terminado el ciclo. Aquí se puede asignar cualquier condición válida para un IF, ya que manejan exactamente el mismo formato, incluso se pueden agregar operadores lógicos. En una configuración regular se suele hacer una comparación del tipo "menor a" el valor cuya cantidad de vueltas se quiere dar. Por ejemplo "x < 5" si quiero dar cinco vueltas.

Incremento

El incremento es la instrucción que se encargará de modificar el valor de la variable del ciclo para que, eventualmente, la condición dé falso y el ciclo pueda concluir. De otro modo, la condición dará siempre verdadero y el ciclo no terminaría, lo que sería un problema (se conoce como ciclo que tiende a infinito). Generalmente se utiliza un incremento de a una unidad (x++) aunque también puede variar.

Funcionamiento

Cuando la ejecución de un programa se encuentra con un ciclo, lo primero que hace es reconocer cuál es la variable del mismo. Lo siguiente, y por única vez, es ejecutar la inicialización. Luego de eso ejecuta la condición y si la misma da verdadero, ingresa al ciclo y ejecuta todo lo que haya dentro. Dentro del ciclo pueden haber tantas instrucciones como necesitemos, incluso otros ciclos, pero esto lo veremos más adelante. Luego de haber ejecutado todas las instrucciones dentro del ciclo y haber llegado al final del mismo, el próximo paso es el incremento. Una vez realizado, se vuelve a evaluar la condición, y si sigue dando verdadero, sigue entrando al ciclo. Así hasta que la condición por fin dé falso.

Como se ha dicho antes, cada una de las configuraciones cuenta con lo que llamaremos "configuración estándar", sin embargo se puede establecer valores de cualquier tipo en cada una de ellas siempre y cuando se cumpla con la estructura respectiva. Hay que tener en cuenta que la cantidad de vueltas que dará el ciclo depende directamente de la configuración de su reloj, con lo cual tenemos que tener muy claro qué es lo que estamos configurando y por qué.

Configuración estándar

Si queremos que un ciclo de diez vueltas, la configuración más sencilla sería:

* x = 0
* x < 10
* x++

Donde "x" arranca en cero, y en cada vuelta se preguntará si "x" es menor a diez. Mientras sea, se ingresa y se da una vuelta. Al final de cada vuelta se incrementará el valor de "x" en uno.

Ejemplo en código

Hagamos en código el ejemplo de mostrar cinco veces el hola mundo pero escribiendo una sola vez dicha instrucción:

for(x = 0; x < 5; x++){  
   Console.WriteLine("Hola mundo");  
}

Ciclo While

Ya conocemos el Ciclo FOR. El mismo nos permite, a partir de una configuración previamente establecida, repetir un conjunto de instrucciones siempre y cuando conozcamos la cantidad de veces que necesitamos que el mismo se ejecute. Hay ocasiones en las que no se conocerá la cantidad de veces que un algoritmo deberá ser ejecutado; en dichos casos se utiliza otro tipo de ciclo, denominado ciclo inexacto, que repetirá el conjunto de instrucciones dado dependiendo de una condición establecida. Existen básicamente dos tipos de ciclos inexactos en la programación. Por un lado el While y por otro lado en Do While.

El ciclo While (del inglés “mientras”) comenzará a ciclar siempre y cuando la condición dada sea verdadera, y seguirá ciclando hasta tanto esa situación no cambie.

int N = 0;  
while(n<10){  
   n++;  
   Console.WriteLine(n);  
}

En el ejemplo se declara una variable inicializada en cero. Cuando el flujo se encuentra con el ciclo While, lo primero que se hace es evaluar la condición: “El contenido de la variable, es menor a diez?”. Naturalmente en este momento la condición tiene como resultado un valor “TRUE”. Recordemos que una condición de este tipo puede adoptar sólo uno de dos valores (true o false. 1 o 0, etc.). Una vez dentro del ciclo, se ejecutan las instrucciones. Se incrementa el valor de la variable y se muestra en pantalla. Una vez concluías estas dos instrucciones, se evaluará la condición nuevamente. El ciclo del ejemplo dejará de girar cuando la variable “N” sea igual a 10 (diez).

Ciclo Do While

Este ciclo funciona casi de la misma manera que el ciclo While tradicional. La diferencia fundamental es que la primer vuelta se ejecutará siempre, y recién para la segunda es que la condición será evaluada para determinar si continuar ciclando o concluir el bloque. Un ejemplo codificado sería:

int N = 0;  
do{  
   n++;  
}while(n<10)

7. Vectores

Arrays en C# (Docu Ofucial)

<https://docs.microsoft.com/en-us/dotnet/api/system.array?view=netcore-3.1>