Índice

[Fundamentos de C# con NET Core 2](#_Toc28876844)

[Descarga e instalación de .NET Core 2](#_Toc28876845)

[Mi primer programa C# 3](#_Toc28876846)

[Instalación de Visual Studio Code 6](#_Toc28876847)

[Configuración IDE 6](#_Toc28876848)

[Breve Historia de C# 6](#_Toc28876849)

[Principios de C# 7](#_Toc28876850)

[Versiones de C# ¿De dónde viene su nombre? 8](#_Toc28876851)

[Clases, objetos, métodos, acciones y atributos 9](#_Toc28876852)

[¿Qué puede ser descrito como un objeto? 9](#_Toc28876853)

[Clases y objetos 10](#_Toc28876854)

[Un objeto NO es una clase 10](#_Toc28876855)

[Introducción a la Programación Orientada a Objetos 11](#_Toc28876856)

[Etapa 1 – Creación de la estructura básica de la escuela 12](#_Toc28876857)

[Uso de numeraciones y el método ToString 13](#_Toc28876858)

[Creación de parámetros 13](#_Toc28876859)

[Etapa 2 – Creando otros objetos para nuestra escuela 14](#_Toc28876860)

[Crea un arreglo de objetos 14](#_Toc28876861)

[Formas de recorrer arreglos 15](#_Toc28876862)

[Condiciones 16](#_Toc28876863)

[Operadores en C# 16](#_Toc28876864)

[Algunos ejemplos de operadores en C# 17](#_Toc28876865)

[Etapa 3 – Colecciones vs arreglos 17](#_Toc28876866)

[Tipos de colecciones 18](#_Toc28876867)

[Diferencias entre Arreglos y Colecciones 19](#_Toc28876868)

[Removiendo y buscando objetos en colecciones 19](#_Toc28876869)

[Expresiones lambda y delegados 20](#_Toc28876870)

[Etapa 4 – Refactorizar a escuela engine 20](#_Toc28876871)

[Clases estáticas 21](#_Toc28876872)

[Generación de datos aleatorios 21](#_Toc28876873)

# Fundamentos de C# con NET Core

Bienvenidos al Curso de C# con .Net Core en este curso el profesor Juan Carlos Ruiz, que ha trabajado por 20 años en el ámbito de la programación y actualmente trabajando en Microsoft, te profundizara en el lenguaje C# en temas como excepciones, herencia, polimorfismo y lenguaje Linq.

# Descarga e instalación de .NET Core

Ingresamos al siguiente link <https://dotnet.microsoft.com/download> o ingresamos a la página oficial de Microsoft si por alguna razón el link no funcionara o simplemente googleamos .NET Core , una vez allí nos detectará el SO instalado y procedemos a descargar los paquetes.

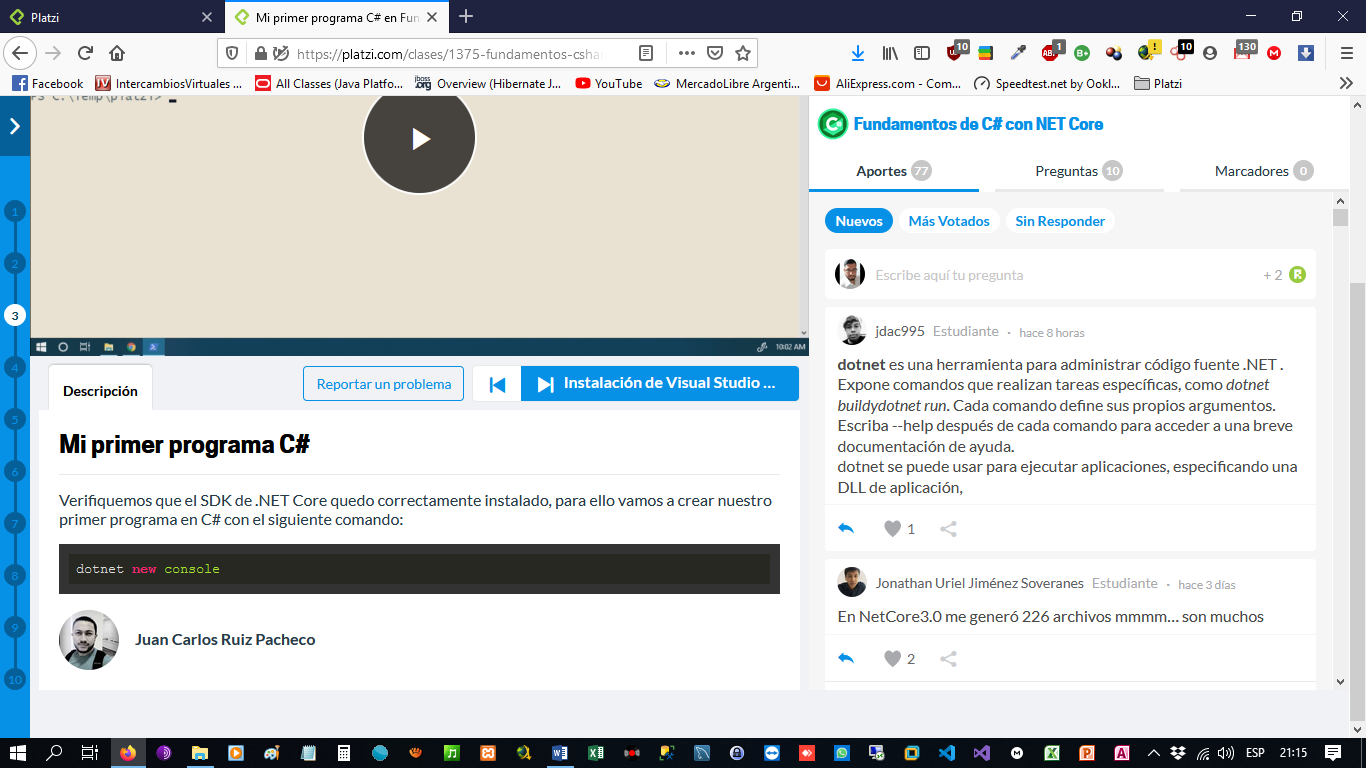
Hay tres tipos de descargas, las cuales son:

* Run Apps: Paquetes necesarios para correr aplicaciones con .NET Core.
* Build Apps: Paquetes necesarios para desarrollar aplicaciones con .NET Core.
* Advanced: Versiones anteriores, binarios, sumas de verificación y más.

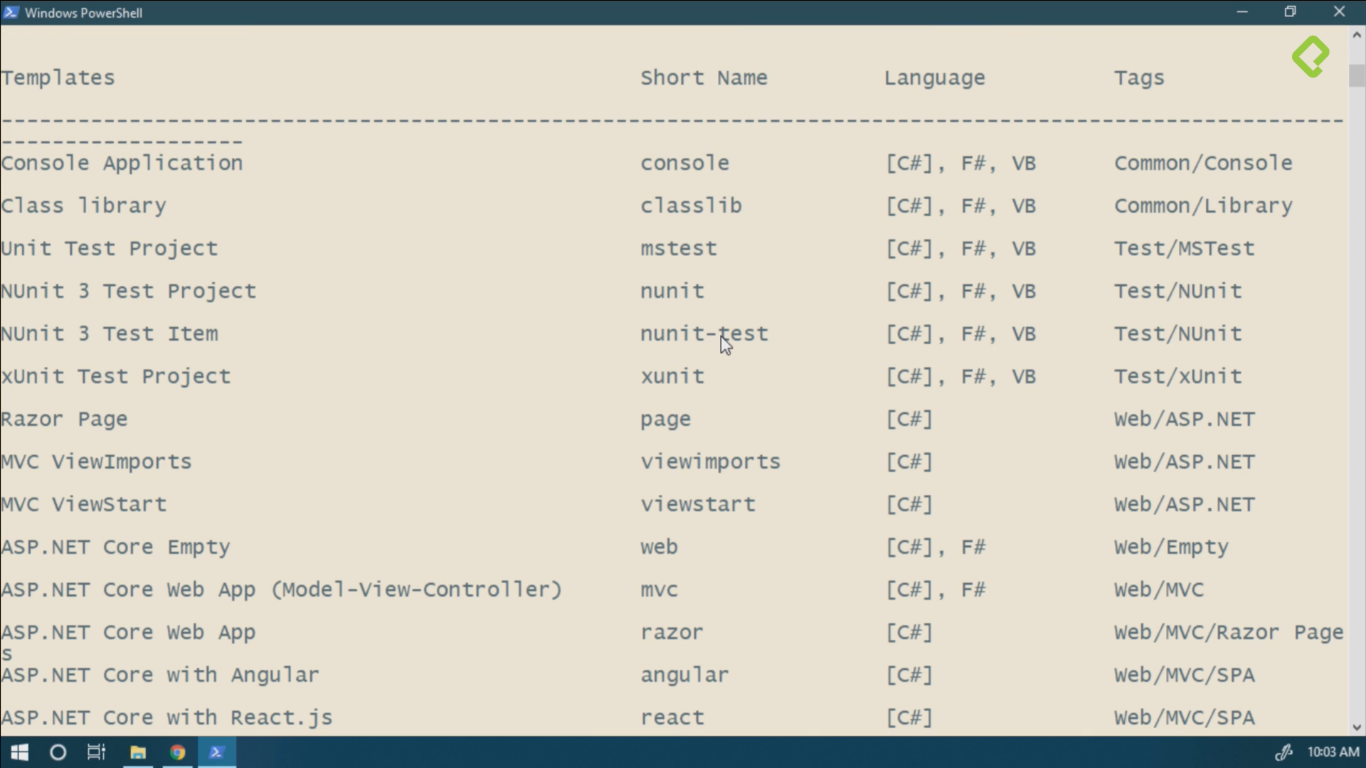
Nosotros en este curso necesitaremos descargar **Build Apps** para poder desarrollar aplicaciones.

# Mi primer programa C#

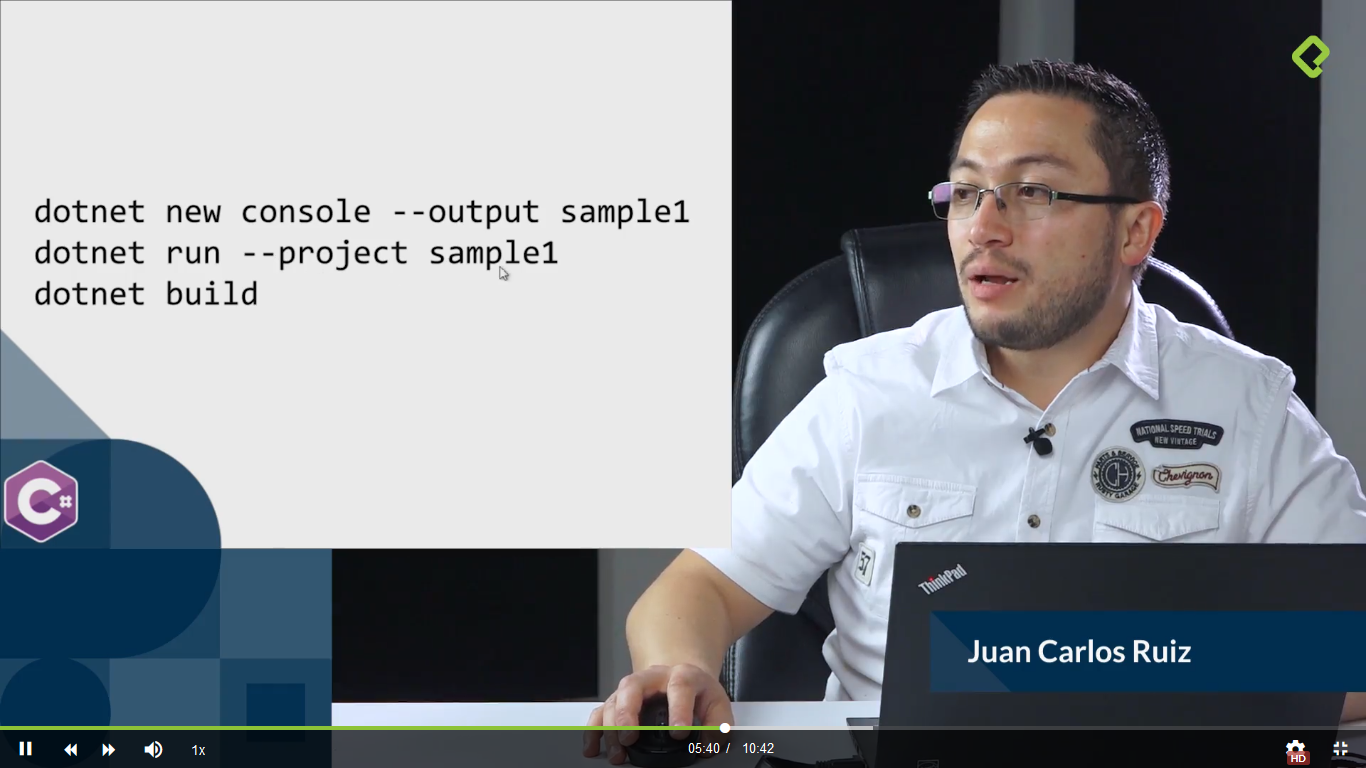
Si estamos dentro de windows es recomendable hacerlo desde una ventana de comando de **PowerShell** porque tiene mayor herramienta para desarrolladores y es la más moderna de windows hasta la actualidad. Verifiquemos que el SDK de .NET Core quedo correctamente instalado, para ello vamos a crear nuestro primer programa en C# con el siguiente comando:



El código anterior nos creará un proyecto de tipo **console**. Para ver las distintas opciones de creación de proyectos digitamos **dotnet new** y presionamos enter,nos mostrará lo siguiente:

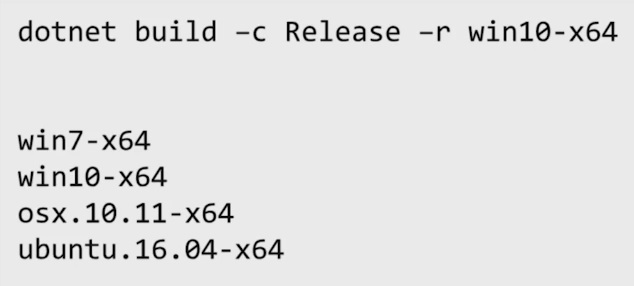


Existen una serie de parámetros que podemos utilizar, entre ellos se encuentran



* **dotnet new console - - output nombre\_carpeta**: Con el parámetro **- - output** podemos especificar en qué carpeta queremos crear nuestro proyecto.
* **dotnet run - -project nombre\_proyecto**: Podemos ejecutar nuestro proyecto o incluso podemos ejecutar el nombre de un proyecto en específico si tenemos proyectos muy grandes con el parámetro **- -project**.
* **dotnet build**: Nos permite compilar nuestro proyecto de consola.

Existe otra forma más especializada de crear nuestros ejecutables y es con el comando



Sabemos que el ejecutable por defecto en windows es un **.exe** el formato ejecutable de .NET Core es una **dll** y no es una **dll** de windows, es una **dll** propia de .NET Core. Por lo tanto, es un ejecutable “nativo” que corre en cualquier plataforma.

El comando que tenemos en la imagen anterior **dotnet build - c Release – r win10-x64** nos genera un ejecutable de windows, pero optimizado para cada plataforma que le indiquemos como se muestra en la imagen.

* El parámetro **– c** nos indica que vamos a generar:
  + Código de producción.
  + Código de depuración.
* El parámetro **– r** nos indica el SO que queremos compilar.

Sí nos dirigimos a la carpeta que nos genera este comando veremos que tiene varios archivos, entre ellos

* nombre\_proyecto.exe
* nombre\_projecto.dll

El cual **.dll** nos permitirá ejecutar nuestro proyecto en cualquier plataforma y el **.exe** sólo nos permitirá ejecutar en entorno windows.

# Instalación de Visual Studio Code

A medida que necesitamos hacer programas más serios necesitamos mejores herramientas, en este curso vamos a utilizar Visual Studio Code como nuestro editor de código.

Visual Studio Code es el editor de código preferido para proyectos de C#, pero cuando estamos realizando software más complejo es preferible usar **Visual Studio Community**.

# Configuración IDE

Cuando abramos nuestro primero proyecto con **Visual Studio Code** nos daremos cuenta de que este no entiende nada del lenguaje de C#, para ello nos va a sugerir instalar ciertas extensiones para el lenguaje.

A medida que usemos Visual Studio Code nos ira sugiriendo diversas extensiones que nos ayudaran con lo que estemos trabajando.

# Breve Historia de C#

Creado por Anders Hejilsberg (canoso) y su equipo.

C# es un lenguaje multiparadigma

* Fuertemente tipado: Debemos elegir los tipos de datos con los que voy a trabajar, por ej, si trabajamos con números enteros debemos usar

**int nombre\_variable = 5;**

* Imperativo: Se basa en sentencia de lo que hay que hacer, es decir, doy una instrucción y esa instrucción se tiene que ejecutar y yo establezco un flujo para esas operaciones.
* Declarativo: Donde yo puedo tener no un flujo de programación determinado sino una especificación y esa especificación se convierte luego en un conjunto de instrucciones para el lenguaje, pero desde el punto de vista del programador, hace una especificación de pasos.
* Funcional: Es un paradigma de programación declarativa basado en el uso de funciones matemáticas, en contraste con la programación imperativa, que enfatiza los cambios de estado mediante la mutación de variables​.
* Genérico: Es como se estructuran los objetos o la información en momería para manipularla de una manera más fácil como programador, pero también dependiendo de cómo se implemente para permitir que el SO o Framework que ejecuta un programa sea mucho más eficiente.
* Orientado a objetos (basado en clases): También es basado en objetos y basado en clases.
* Orientado a componentes: Es decir, permitir las diferentes piezas de software se puedan conectar una con otras de manera desacoplada.

# Principios de C#

* Simple, moderno, de propósito general, orientado a objetos.
* Robusto, durable, productividad para el programador.
* Ambientes distribuidos.
* Portabilidad.
* Soporte para internacionalización.
* Uso en servidores y entornos embebidos.
* Bajo uso de procesador y memoria.

# Versiones de C# ¿De dónde viene su nombre?

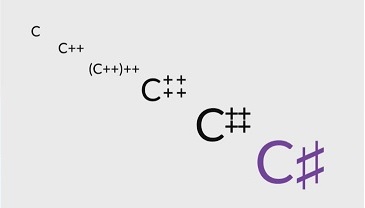
En un comienzo las versiones de C# iban agregando funcionalidades básicas o funcionalidades al Core, luego llegaron versiones creando grandes bloques de funcionalidad y con el tiempo fue madurando el framework, entonces las nuevas versiones de C# lo que buscaban eran nuevas funcionalidades para escribir más rápido y amigable el código.

La versión 1.0 de C# era muy parecida a Java. A partir de la versión 2.0 empezaron a implementar nuevas ideas como:

* Los tipos parciales.
* Métodos anónimos.
* Iteradores.
* Tipos nullable.
* Getter and Setter.
* Entre otras cosas.

Para la versión 7.0 se incrementó la velocidad de lanzamiento de una nueva versión, entonces empezaron a sacar C# 7.1, 7.2.

C# se pensó como una mejora al lenguaje C++, este lenguaje tiene su nombre por ser una mejora del lenguaje C, entonces llamar (C++)++ a C# hubiera quedado muy mal. Por ello se optó en llamarlo **C Sharp** que **Sharp** significa inteligente, pero el símbolo de Sharp no es fácil de sacar con el teclado por ello se usa el signo **#**.



# Clases, objetos, métodos, acciones y atributos

## **¿Qué puede ser descrito como un objeto?**

Todo puede ser descrito como un objeto, con ciertas características y métodos, por ejemplo:

Toda escuela tiene un nombre, una ubicación, un tipo, esas serían sus características. Mientras que sus métodos pueden ser iniciar clases, terminar clases, sonar el timbre, etc.



## **Clases y objetos**

* Un objeto tiene atributos (color, texto) y comportamientos (métodos/acciones)
* La clase es la descripción de un objeto.
* Por defecto los atributos de las clases tienen un nivel de acceso no público, por ende, debemos editarlo y hacerlo público para acceder a ellos.
* Con la palabra reservada **void** estamos diciendo que esa acción hace algo, pero no retorna ningún valor.

## **Un objeto NO es una clase**

La clase es la descripción de un objeto, pero no el objeto, más bien **es una plantilla** para crear objetos. Por ej: ¿El plano de una casa es una casa? No, pero nos sirve para construir una casa.

# Introducción a la Programación Orientada a Objetos

Cuando creamos un nuevo proyecto lo primero que nos encontraremos es una clase con el método Main, por convención este es el primer método que se ejecuta en una aplicación por consola.

Para definir una clase tenemos que usar la palabra reservada “class” antes del nombre de la clase a definir, todos sus métodos y atributos se encontraran dentro de las llaves o curly brackets.

Para crear una nueva instancia de una clase debemos crear una variable y asignarle la palabra reservada “new” seguido del nombre de la clase.

Cada cosa definida dentro de una clase tiene un nivel de acceso, por defecto el nivel de acceso es privado o protegido y esto significa que no pueden modificar o acceder a ese atributo. Para poder acceder por fuera de la clase se debe asignar el nivel de acceso público con la palabra reservada “**public**”.

# Etapa 1 – Creación de la estructura básica de la escuela

Dentro de nuestro proyecto crearemos una carpeta llamada “**Entidades**” donde iremos guardando las entidades del proyecto como un Curso, una Escuela, etc.

Dentro de cada archivo **.cs** añadiremos un espacio de nombres o **namespace**, este nos sirve para facilitar el encontrar objetos en nuestro proyecto.

En la programación orientada a objetos los datos de una clase pueden ser protegidos del acceso directo o público, esto es el **Encapsulamiento**. Dentro de los métodos por los cuales va a acceder a la información se puede agregar cualquier tipo de lógica.

Para que se pueda crear un objeto se hace uso de la palabra reservada **new**, esto invoca al método constructor de la clase.

# Uso de numeraciones y el método ToString

Ya hemos creado la estructura básica de la escuela, ahora vamos a explorar como se muestra la información en pantalla, además crearemos una nueva entidad llamada “**TiposEscuela**” que será de tipo **enum**.

Cuando imprimimos en consola la variable de nuestro objeto escuela vemos que por defecto nos describe el nombre completo del objeto, en este caso es el namespace donde se encuentra más el nombre de la clase. Para modificar esto podemos sobrescribir el método “ToString” con la palabra clave override.

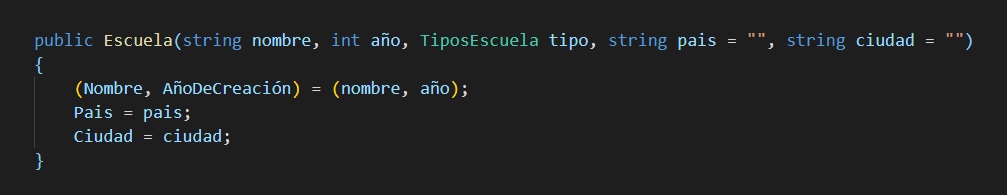
En C# todo es un objeto y por ello todo hereda de la clase **Object**, es así como todos los objetos tienen el método **ToString**.

# Creación de parámetros

Vamos a profundizar en cómo crear una lista de parámetros y la manera en que podemos hacer que esos parámetros sean opcionales.

Para los parámetros opcionales basta con asignarle un valor por defecto en el método. Si queremos acceder a la variable de una clase dentro de la misma podemos usar la palabra **this**.

Por ejemplo:



Parámetros opcionales

Asignación de tuplas

# Etapa 2 – Creando otros objetos para nuestra escuela

Nuestra escuela va a tener Cursos y estos cursos van a tener una Jornada, en esta clase vamos a crear las entidades de **Curso** y **TiposJornada**.

Cada curso va a tener un **UniqueId** que se va a generar automáticamente dentro del método constructor.

# Crea un arreglo de objetos

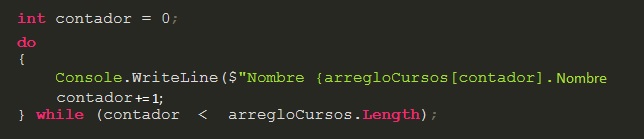
Un arreglo o array es una entidad donde podemos alinear varios objetos del mismo tipo en memoria. La posición en un arreglo arranca desde 0, entonces en un arreglo cuyo tamaño es 3 la última posición seria 2.

Existen muchas formas de recorrer un arreglo, entre ellas se encuentra la sentencia while que va a ejecutarse hasta que su condición se termine, en este caso se termina de ejecutar hasta que haya recorrido todo el arreglo.

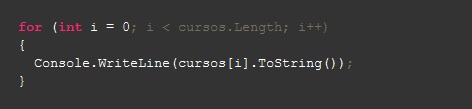
# Formas de recorrer arreglos

Además de recorrer un arreglo con la sentencia **while** podemos encontrar:

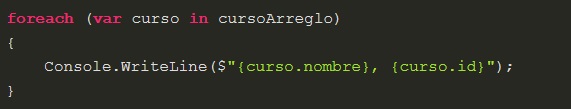
* La sentencia **do while** donde a diferencia de while se va a ejecutar primero el código y luego va a verificar si la condición es válida.



* La sentencia **for**.



* La sentencia **foreach**.



# Condiciones

El operador **if** nos indica si se cumple o no una condición, si la condición se cumple entonces ejecuta el bloque de código, en caso de no cumplirse no se ejecuta. La sentencia **else** se ejecuta cuando un if no se cumple y la sentencia **else if** lo que hace es en caso de no cumplirse un if, pregunta por una nueva condición.

Existen diferentes operadores que resultan útiles para verificar condicionales, algunos de ellos son:

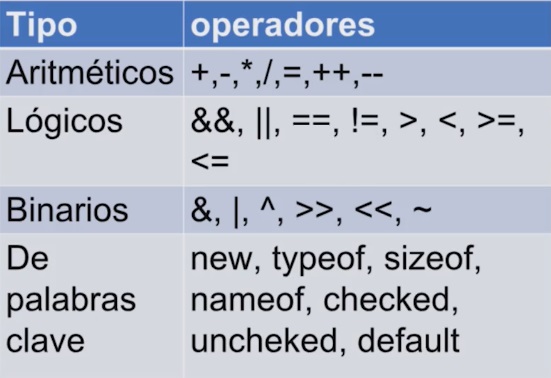
* || o or.
* && o and.
* > mayor que.
* < menor que.
* % operador MOD, retorna el residuo de una división.

# Operadores en C#

En el lenguaje de C# existen diferentes tipos de operadores, se encuentran los Aritméticos, Lógicos, Binarios y algunos por palabras clave.

Hay más de 40 operadores, la mayoría de ellos son combinaciones de tipos más básicos y puedes ver más sobre ellos en el siguiente [enlace](https://docs.microsoft.com/es-es/dotnet/csharp/language-reference/operators/).

## **Algunos ejemplos de operadores en C#**

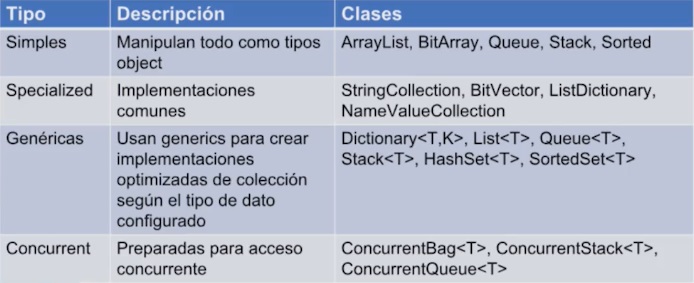


# Etapa 3 – Colecciones vs arreglos

"Los arreglos son mucho más rápidos para trabajar en memoria, consumen menos memoria, si se utilizan tipos nativos son mucho más eficientes y son ideales para trabajar con APIS de bajo nivel.

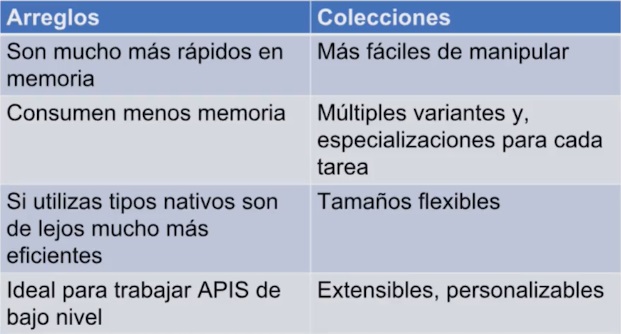
Por otro lado, las colecciones son más fáciles de manipular, tienen múltiples variantes y, especializaciones para cada tarea, los tamaños son flexibles y las colecciones son extensibles y personalizables.

## **Tipos de colecciones**



Si quieres ver más tipos de colecciones que hay en C# visita el siguiente [enlace](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/collections).

## **Diferencias entre Arreglos y Colecciones**



# Removiendo y buscando objetos en colecciones

Para eliminar un objeto especifico de una colección podemos hacerlo mediante referencia que es pasándole al método **Remove** la variable del objeto a eliminar, o también indicando el índice del objeto que queremos eliminar con el método **RemoveAt**.

El método **RemoveAll** pide como parámetro un **Predicate** el cual es una función que solamente va a regresar valores **Bool** y se va a ejecutar en cada elemento de la colección, de igual forma la función va a recibir como parámetro un elemento de la colección y en caso de que la condición regrese un valor de **true** entonces el objeto se va a borrar.

# Expresiones lambda y delegados

Una expresión lambda es una función anónima que puedes usar para crear delegados, son bastante útiles pues tu código queda de forma más compacta, pero solamente si esta función se utilizara en un solo lugar.

En C# puedes comentar de distintas formas:

* Con **//** para comentar una sola línea.
* Con **/\*** y **\*/** para comentar varias líneas.
* Con **///** para comentar lo que hace un método especifico.

# Etapa 4 – Refactorizar a escuela engine

Es importante refactorizar nuestro código para mantenerlo ordenado.

El constructor de una clase tiene que ser tan rápido como sea posible, tiene que estar desconectado de cualquier cosa que implique entrada y salida.

# Clases estáticas

Una clase estática no permite crear nuevas instancias, la clase por si misma funciona como un objeto.

# Generación de datos aleatorios

Si un método tiene más de 50 líneas significa que está mal diseñado, cada método debe tener una sola responsabilidad. Es por eso por lo que vamos a separar en distintos métodos la carga de datos de nuestro proyecto.