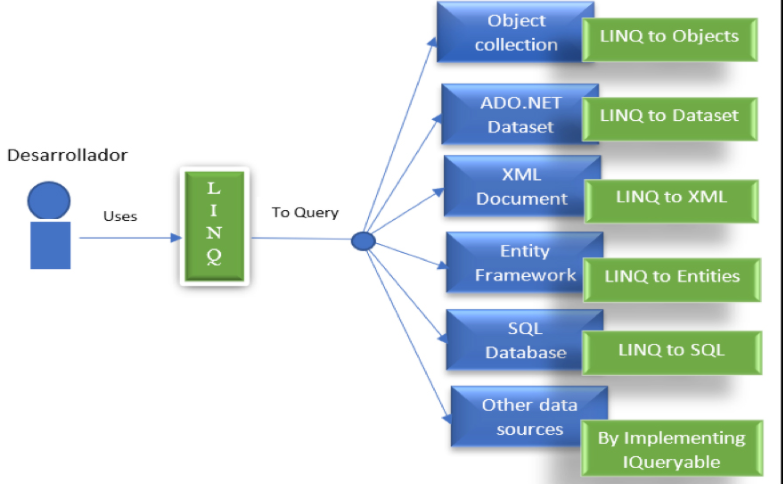
**¿Qué es LINQ?**

LINQ (Language Integrated Query) es una sintaxis de consulta uniforme en C # y VB.NET utilizada para guardar y recuperar datos de diferentes fuentes. Está integrado en C # o VB, lo que elimina la falta de correspondencia entre los lenguajes de programación y las bases de datos, y proporciona una interfaz de consulta única para diferentes tipos de fuentes de datos.  
Por ejemplo, SQL es un lenguaje de consulta estructurado que se usa para guardar y recuperar datos de una base de datos. De la misma manera, LINQ es una sintaxis de consulta estructurada construida en C # y VB.NET utilizada para guardar y recuperar datos de diferentes tipos de fuentes de datos, como una colección de objetos, una base de datos SQL server, XML, MySQL o un servicio web entre otros.

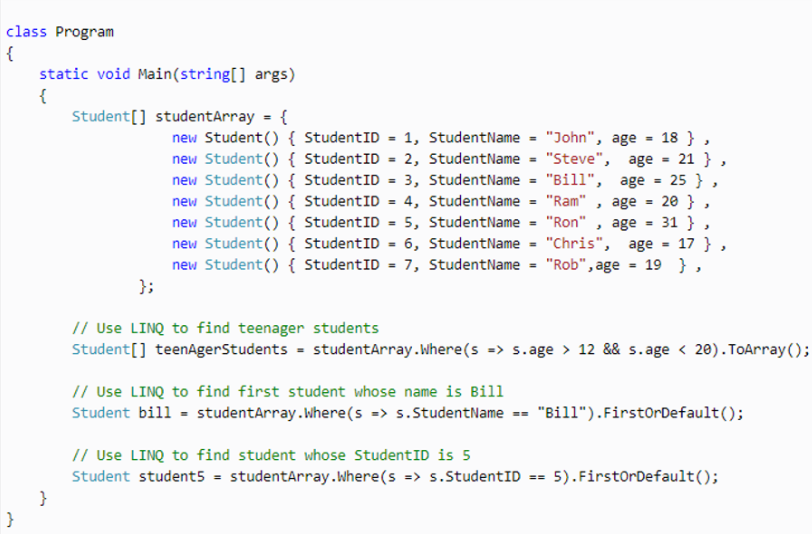
LINQ siempre trabaja con objetos para que pueda usar los mismos patrones básicos de codificación para consultar y transformar los datos.



**¿Por qué LINQ?**

Para entender por qué deberíamos usar LINQ, veamos un ejemplo. Supongamos que quiere encontrar una lista de estudiantes adolescentes de una variedad de objetos de Estudiante.  
Antes de C# 2.0, teníamos que usar un bucle ‘foreach’ o ‘for’ para recorrer la colección y encontrar un objeto en particular. El uso del bucle for es engorroso, no se puede mantener y leer adecuadamente, por eso el equipo de C# sintió que necesitaban hacer el código más compacto y legible, entonces introdujeron el método de extensión, la expresión lambda, el árbol de expresiones, el tipo anónimo y la expresión de consulta, naciendo LINQ en la versión de C# 3.0.

El siguiente ejemplo muestra cómo puede usar la consulta LINQ con la expresión lambda para encontrar un alumno (s) en particular de la colección de estudiantes.



Como puede ver en el ejemplo anterior, especificamos diferentes criterios utilizando el operador LINQ y la expresión lambda en una sola instrucción. Por lo tanto, LINQ hace que el código sea más compacto y legible, y también se puede usar para consultar diferentes fuentes de datos. Por ejemplo, si tiene una tabla de estudiantes en una base de datos en lugar de una matriz de objetos estudiantes como se indicó anteriormente.

**Ventajas de LINQ:**

**– Lenguaje familiar:** los desarrolladores no tienen que aprender un nuevo lenguaje de consulta para cada tipo de fuente de datos o formato de datos.  
**– Menos codificación:** reduce la cantidad de código que se debe escribir en comparación con un enfoque más tradicional.  
**– Código legible:** LINQ hace que el código sea más legible para que otros desarrolladores puedan entenderlo y mantenerlo fácilmente.  
**– Forma estandarizada de consultar múltiples fuentes de datos:** la misma sintaxis LINQ se puede utilizar para consultar múltiples fuentes de datos.  
**– Compilar la seguridad del tiempo de las consultas:** proporciona verificación de tipos de objetos en tiempo de compilación.  
**– Soporte IntelliSense**: LINQ proporciona IntelliSense para colecciones genéricas.  
**– Dar forma a los datos:** puede recuperar datos en diferentes formas.

**API DE LINQ**

LINQ no es más que la colección de métodos de extensión para las clases que implementan la interfaz IEnumerable y IQueryable. El espacio de nombres System.Linq incluye las clases y las interfaces necesarias para LINQ. Enumerable y Queryable son dos clases estáticas principales de LINQ API que contienen métodos de extensión.  
El espacio de nombres System.Linq se incluye por defecto cuando agrega una nueva clase en Visual Studio, de modo que puede usar LINQ de forma predeterminada.

**Enumerable:**  
La clase enumerable incluye métodos de extensión para las clases que implementan la interfaz IEnumerable<T>, esto incluye todos los tipos de colección como List<T>, Dictionary<T>, SortedList<T>, Queue<T>, HashSet<T>, LinkedList<T> etc.

**Queryable:**  
La clase Queryable incluye métodos de extensión para las clases que implementan la interfaz IQueryable<T>. IQueryable<T> se utiliza para proporcionar capacidades de consulta contra una fuente de datos específica donde se conoce el tipo de datos. Por ejemplo, Entity Framework implementa la interfaz IQueryable<T> para admitir consultas LINQ con una base de datos subyacente como SQL Server.  
Además, hay API disponibles para acceder a datos de terceros; por ejemplo, LINQ to Amazon ofrece la posibilidad de utilizar LINQ con los servicios web de Amazon para buscar libros y otros elementos mediante la implementación de la interfaz IQueryable.

**Vamos a hacer un pequeño resumen de lo visto hasta ahora**

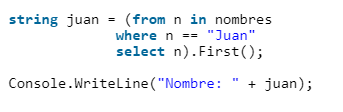
1. Use el espacio de nombres System.Linq para usar LINQ.  
2. La API de LINQ incluye dos clases estáticas principales Enumerable y Queryable, para el almacenamiento de la información que tratamos con LinQ.  
3. La clase Enumerable incluye métodos de extensión para las clases que implementan la interfaz IEnumerable<T>.  
4. El tipo de colecciones IEnumerable<T> son colecciones en la memoria como List, Dictionary, SortedList, Queue, HashSet, LinkedList.  
5. La clase Queryable estática incluye métodos de extensión para las clases que implementan la interfaz IQueryable<T>.  
6. El proveedor de consultas remotas implementa IQueryable<T>, por ejemplo Linq-to-SQL, LINQ-to-Amazon, etc.

**¿Qué es LINQ? 2**

Muchas veces he encontrado desarrolladores manejando arreglos o listas con iteraciones y no evito pensar, ¿porqué no usan LINQ? Quiero pensar que es por desconocimiento porque con LINQ puedes manejar arreglos, listas, en fin, objetos de manera rápida, fácil y eficiente. Pero antes, ¿qué es LINQ?

LINQ es un componente del framework .Net que permite hacer queries contra objetos. Esto incluye arreglos, listas, datatables, etc.. La mejor manera de demostrar el poder que LINQ te otorga es con varios ejemplos.

Unas de las situaciones con la que frecuentemente nos topamos es con buscar un valor en específico o los elemento que cumplan con ciertas condiciones.

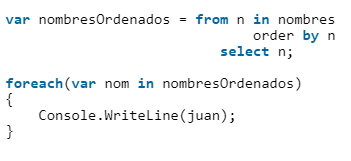


Resultado:  
Nombre: Juan

Como vemos, con varias lineas de código hacemos, de una manera limpia y elegante, lo que antes haciamos con iteraciones. Se preguntarán cómo funciona y verán lo simple que es.

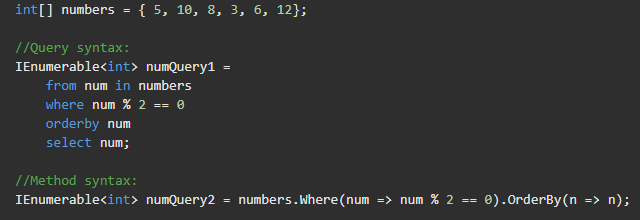
La variable n representa cada elemento en el arreglo nombres. El select nos devuelve una lista de string así que tenemos que añadir el método First() para obtener el primer valor de la lista que es el que cumple con nuestra condición de que sea Juan.

De la misma manera podemos ordenar la lista

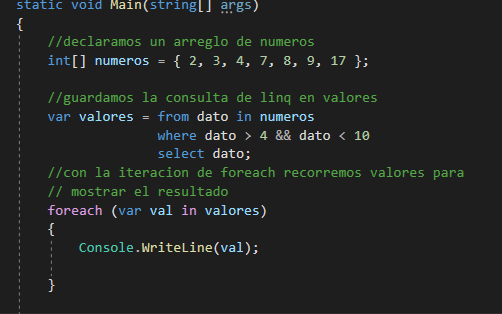


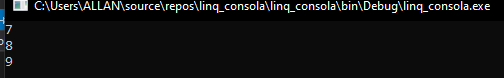
Resultado:  
Juan  
Pedro  
Roberto

Sintaxis de consulta y sintaxis semántica



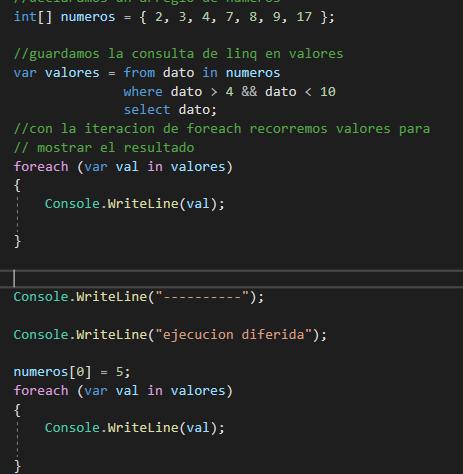
1. Ejemplos de linq en c#

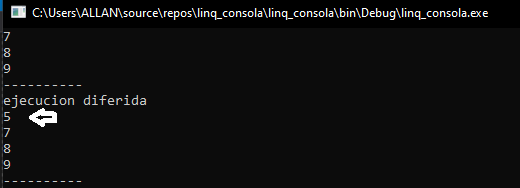




1. Ejecución diferida

Ejecución diferida es cuando hemos realizado la consulta de linq en una estructura de datos si se cambiara un valor de esa estructura y volvemos a iterar nos aparecerá los cambios que fueron realizados.

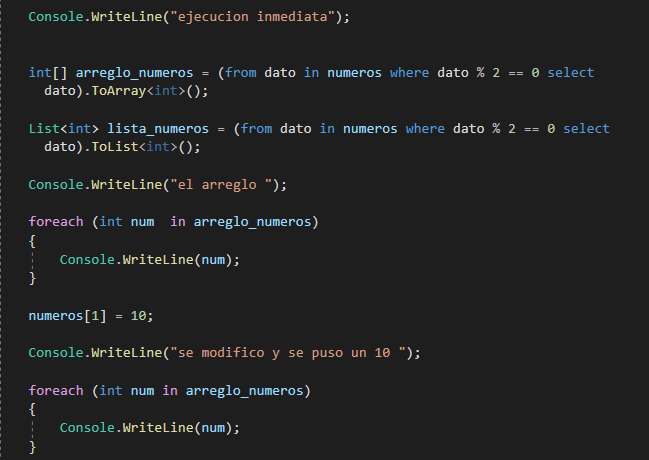


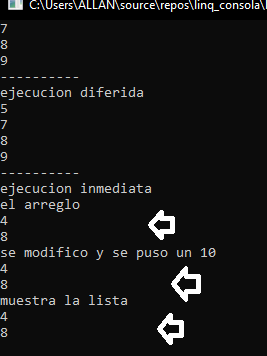


Apareció el cambio que se realizo en la estructura sin necesidad de volver a realizar la consulta de lin q

1. Ejecución inmediata

La ejecución inmediata es cuando se realiza la consulta linq en la declaración de la estructura y si se realizara un cambio en esa estructura no se muestran en si se vuelve a iterar la estructura

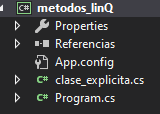




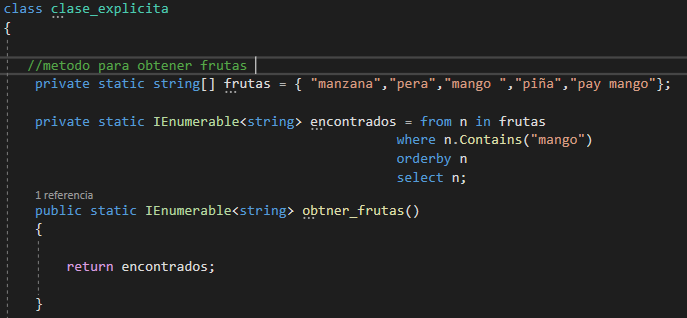
No se ve aplicado el cambio del valor 10 en la posición 0, esto solo muestra los valores cuando fue declarado la estructura

1. Query desde métodos

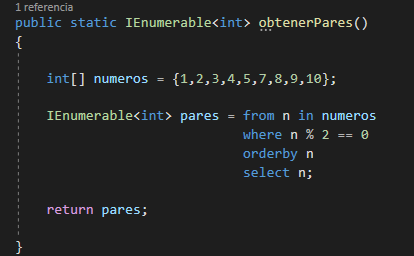
Se crea un main llamado program y una clase llamada claseimplicita



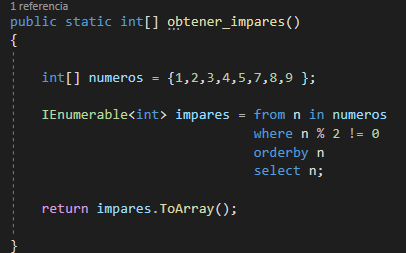
En clase implícita creamos los métodos



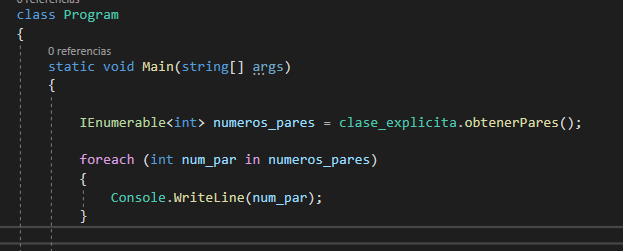
Métodos obtener pares

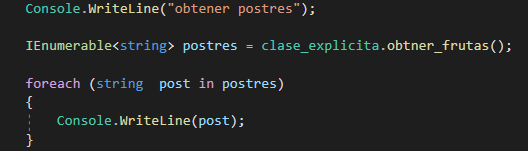


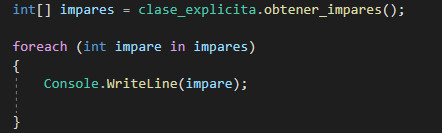
Método obtener impares



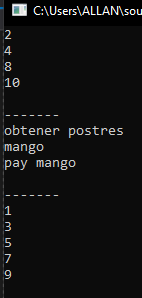
Clase program llamamos los métodos de claseimplicita





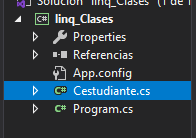


Resultados de los 3 metodos



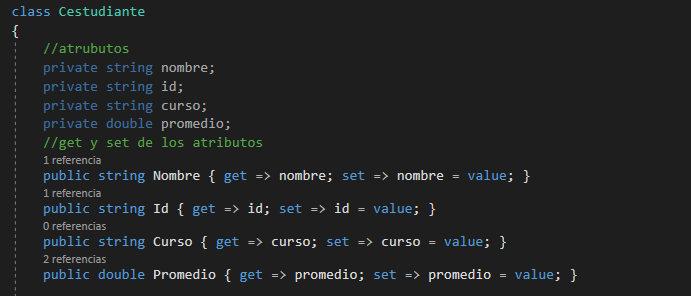
1. Query con clases.

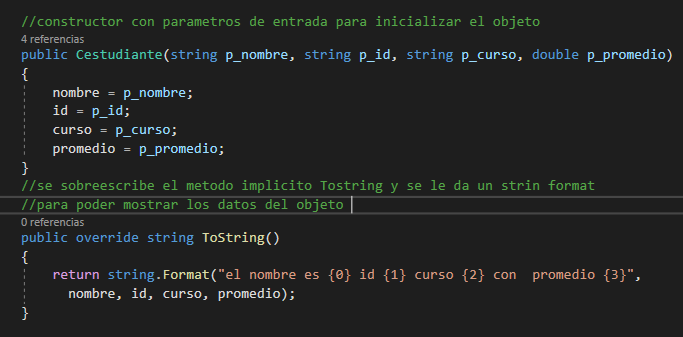
Se crea un programa donde esta la clase estudiante y se guadan objetos de tipo estudiantes un una lista el cual se realizan consultas en linq para poderlas mostrar



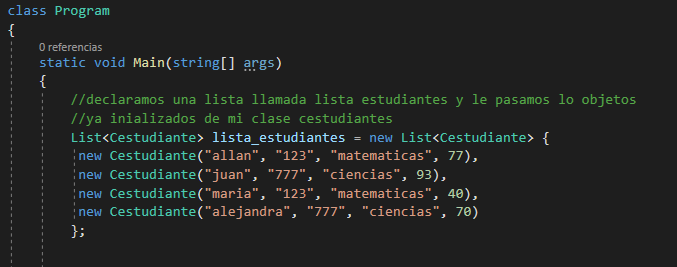
En la clase estudiante se ponen los atributos de la clase y se agregan los set y get

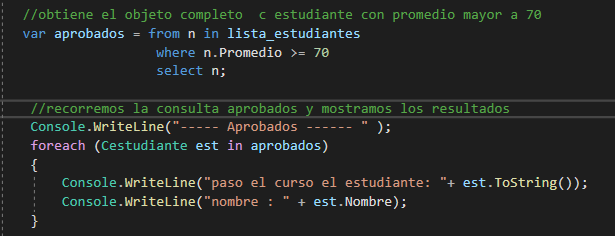
Además se sobrecarga el método Tostring(); para poderle dar formata y asi poder mostrar un formato de string y poder mostrar los datos del objeto c estudiante

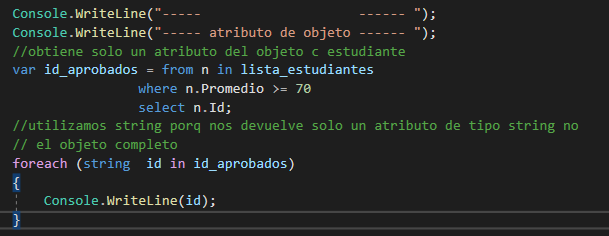




En la clase program declaramos lo siguiente

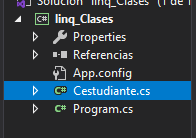






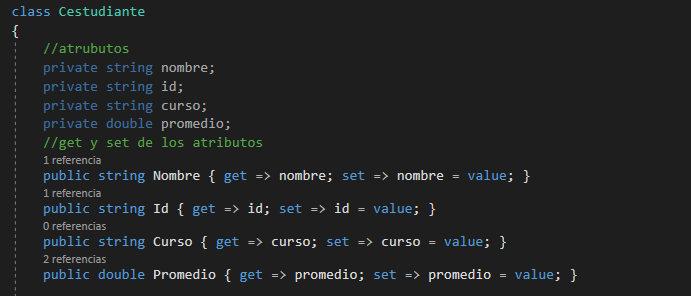
1. Arrayslist con linQ.

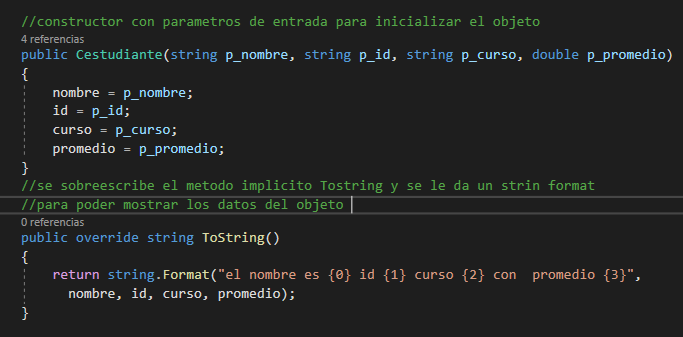
**Arraylist** no implementa a **lenumerable** por eso hay que usar OfType<tipo\_de\_dato>(); para que me muestre un solo tipo de datos del arraylist



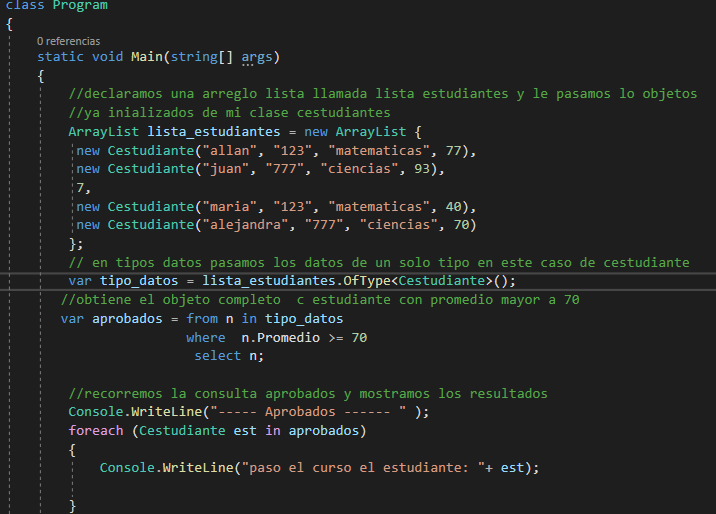
En la clase estudiante se ponen los atributos de la clase y se agregan los set y get

Además se sobrecarga el método Tostring(); para poderle dar formata y asi poder mostrar un formato de string y poder mostrar los datos del objeto c estudiante



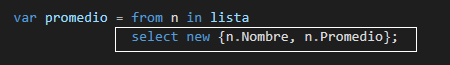


En la clase program ponemos lo siguiente

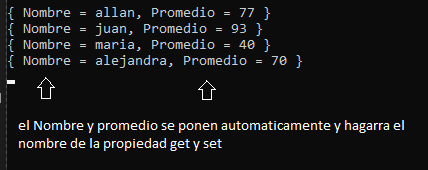


La proyección

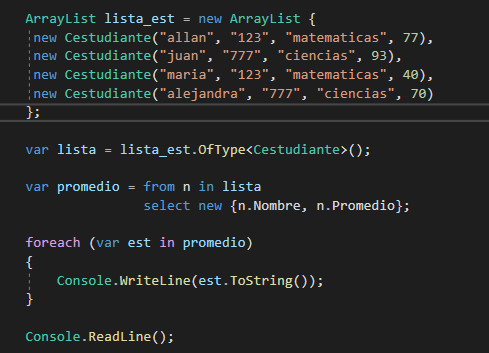
Se realiza después del select ejemplo

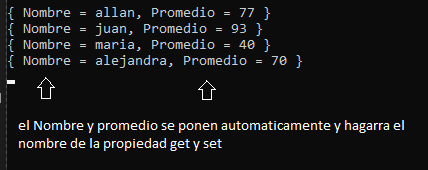


Esto nos mostrara solo el nombre y promedio del arraylist y quedaría asi



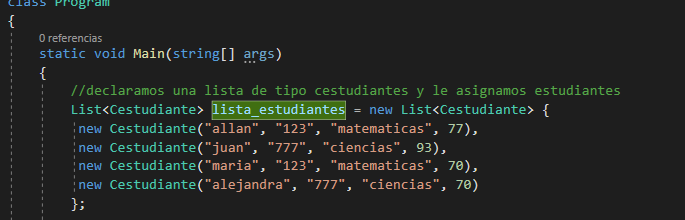
Ejemplo completo



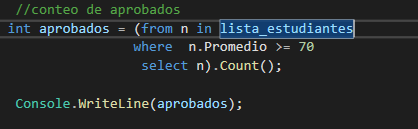


1. Operaciones básicas linq.

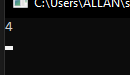
Utilizamos la clase c estudiante de los ejemplos pasados y en program ponemos lo siguiente



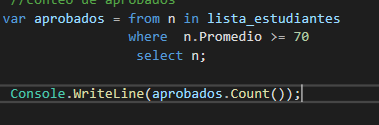
**CONTEO**

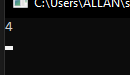
****

Resultado

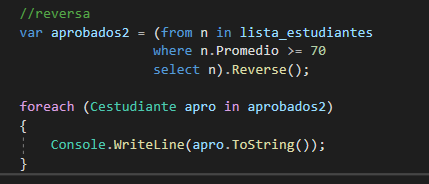
****

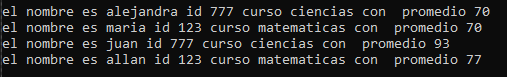
**O de esta otra manera**

****

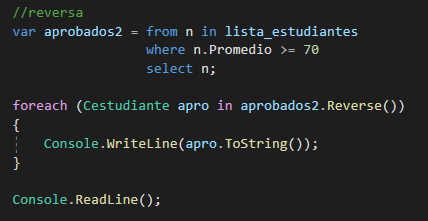
****

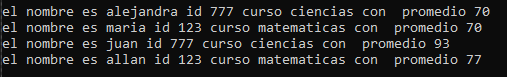
**REVERSA**

****

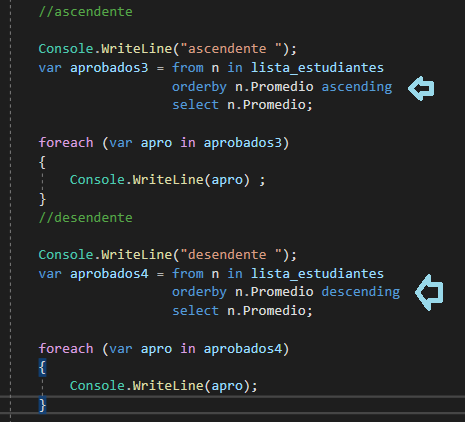


O también así

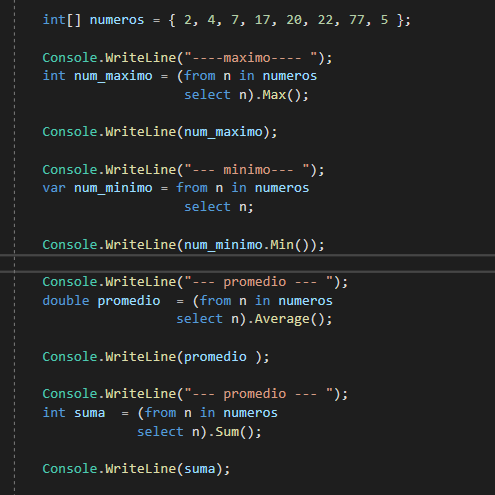


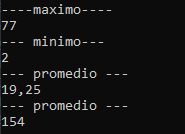


**Ascendente y descendente**

****

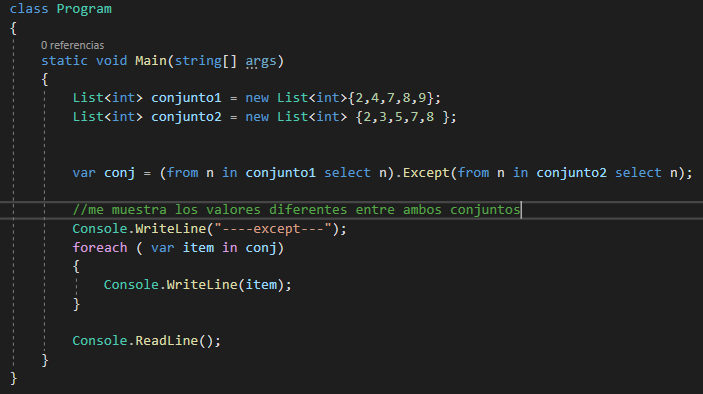
**Operaciones varias**

****

****

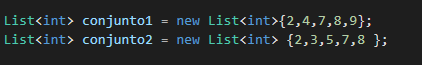
1. Operaciones relacionas con conjuntos.

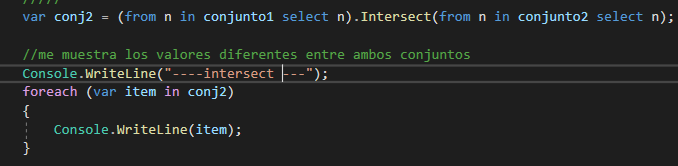
Except: muestras los valores diferentes que tiene el conjunto 1 con respecto al conjunto 2

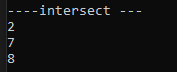




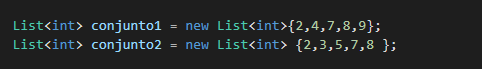
Intersect: muestra los valores iguales que tiene conjunto1 con respecto a conjunto2

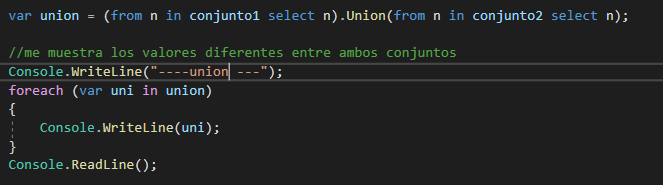


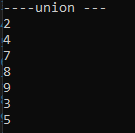




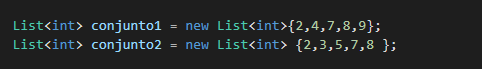
Union: muestra todos los valores que tienen conjunto 1 y conjunto 2 sin repetir datos

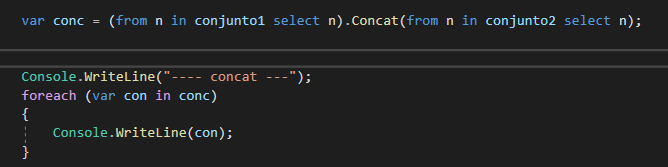


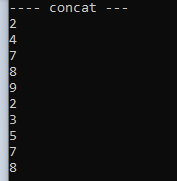




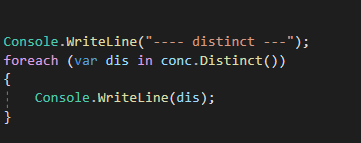
Concat: uno tanto los valores de conjunto 1 como los de conjunto2

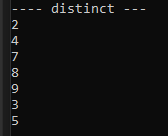






Distinct : remueve los duplicados y trabaja solo con un operando o conjunto.

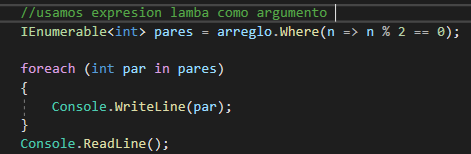


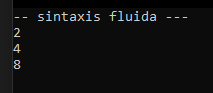


1. Sintaxis fluida.

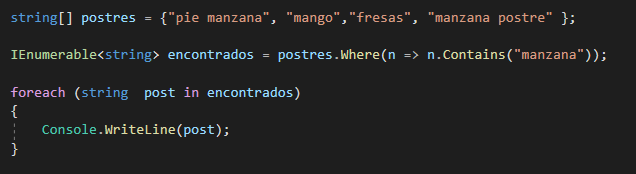
Utilizamos expresión lamba

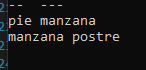




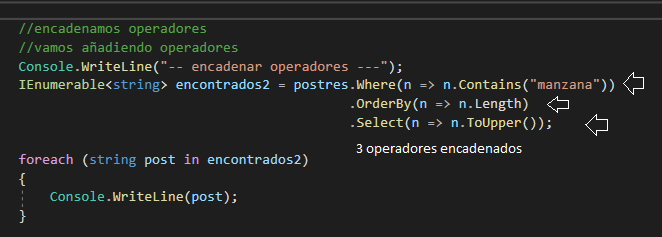


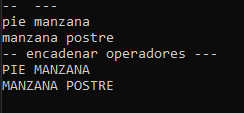
Ejemplo con un string





Encadenado de operadores

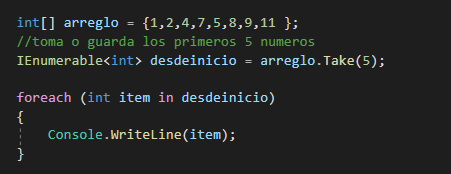


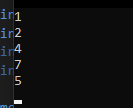


1. Otros operadores

Hay cerca de 40 operadores se van a ver en este ejemplo unos cuantos

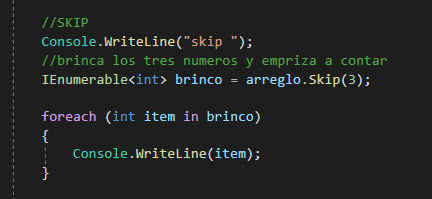
TAKE

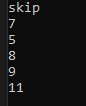




SKIP

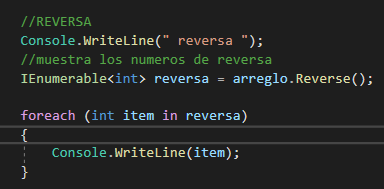


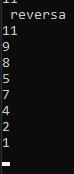




REVERSE

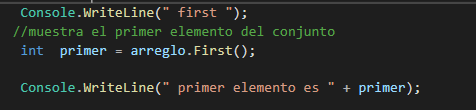






FIRST

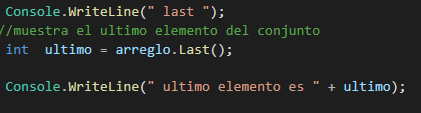






LAST

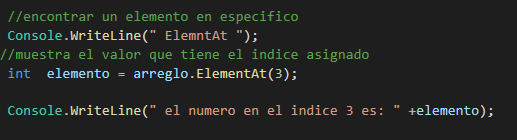






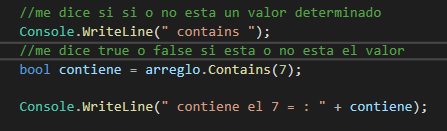
ElementAt







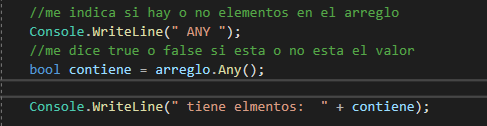
CONTAINS





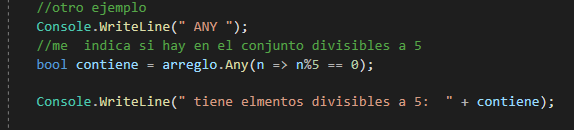
ANY

Devuelve true o false





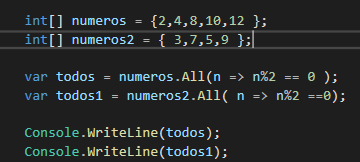
Otro ejemplo





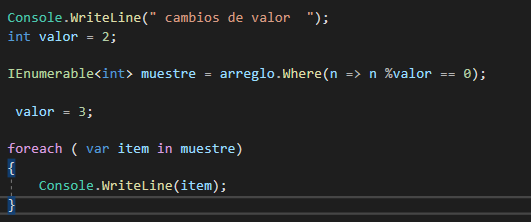
ALL

Me dice si todos cumplen con la condición o es true o es false



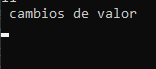


CAMBIOS DE VALOR

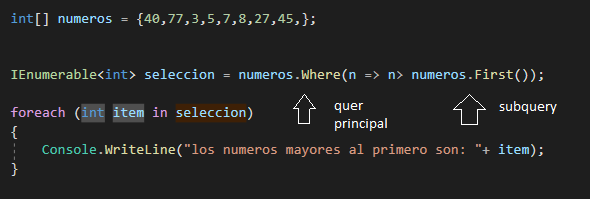


Se va a realizar el cambio de valor y si se va a tomar en cuenta por q el query se carga cuando iteramos muestre con el foreach

Resultado

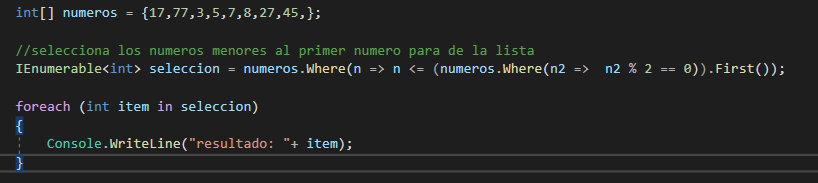


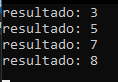
1. Subquerys





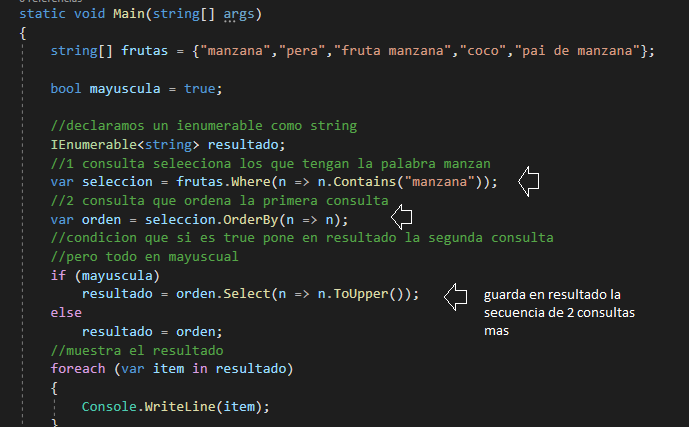
Ejemplo 2





1. Query progresivos

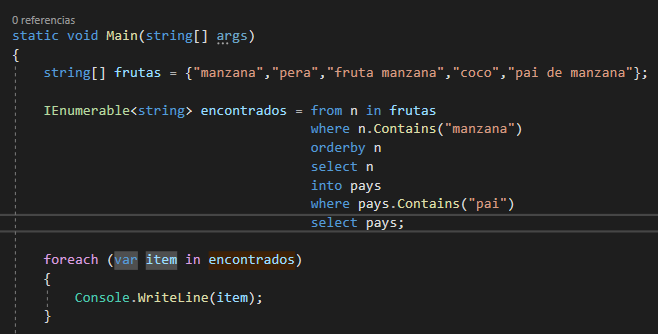
Query progresivo nos permite guardar cambios en una variable de resultados de diferentes querys



INTO

Into solo se puede usar después de select o gruop

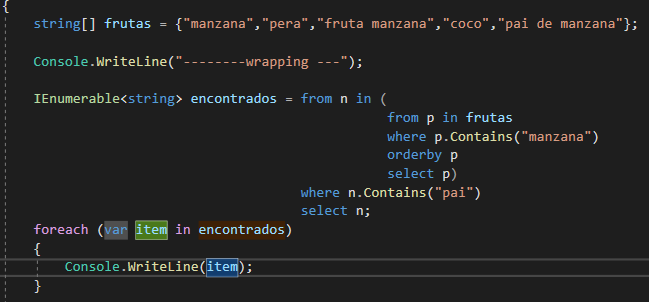
Nos ayuda a reiniciar el query para poder usar otravez el where, orderby y select





Into pays = resultado de la primera consulta y lo continua

ENVOLVER QUERY O WRAPPING





1. operadores 1

hay 3 categorias para los operadores de query

* Secuencia a secuencia (secuencia de entrada secuencia salida)
* Secuencia de entrada y nos da un elemento sencillo o escalar
* Nada de entrada y se obtiene una secuencia de salida

**Secuencia a secuencia:**

**Filtro**: where, startwith, endwith, take, takewhile, skip, skipwhile, distinct

**Proyeccion:** select, selectmany

**Union:** join, groupjoin, zip

**Ordenamiento:** orderby, thenBy, reverse

**Agrupamiento:** groupby

**Operadores de conjunto:** concat, union, intersect, except

**Conversión import:** ofType, cast

**Conversión export:** toarray, todicionary, asEnumerable, asQueryable

**Secuencia a elemento escalar:**

**Operadores de elemento:** first, last, single, element

**Agregacion**: aggregate, average, count, longcount, sum, max, min

**Cuantificador:** all, any, contains, sequenceEqual

**Nada de entrada secuencia de salida**

**Generación:** empty, range, repeat

FILTRO:

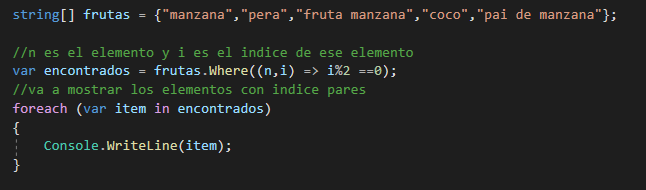
**Where:** regresa un subconjunto de elementos que dependen de la condición.

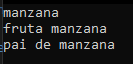
**Take**: regresa los primeros elementos n e ignora el resto

**Skip:** ignora los primeros elementos y regresa el resto

**Distict:** regresa una secuencia que excluye a los duplicados.

**Where:** otro uso de where:





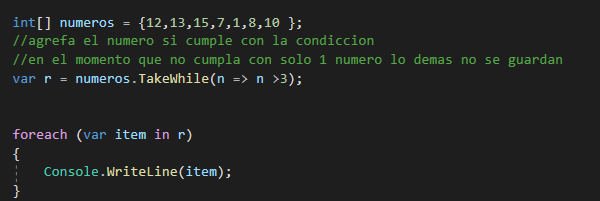
**Startwith:** comienza con



**Endwith**: termina con



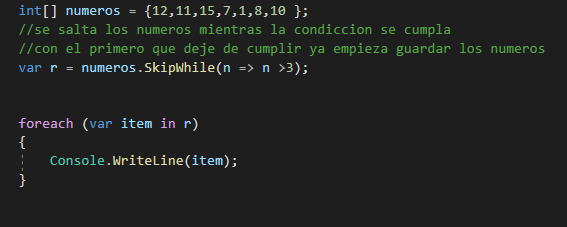
**Takewhile:** va guardando los primeros números hasta que la condición no se cumpla y deja de agregar a partir del primer número que no cumplió con la condición





**skipwhile: se salta los números mientras la condiccion se cumpla**

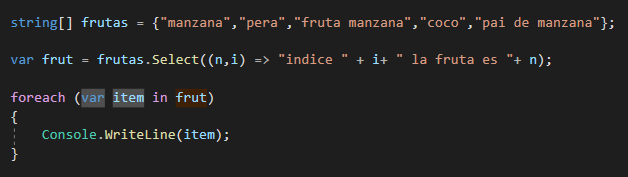
**cuando el primero se deja de cumplir ahí empieza a guardar los números**

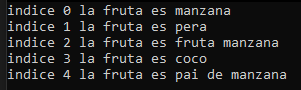




Proyeccion :

Select: otro ejemplo





SelectMany: **falta**

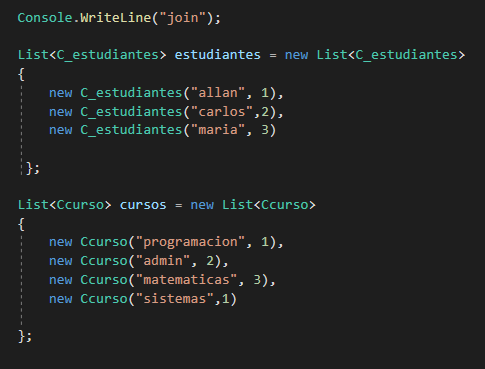
Cada elemento de entrada produce un elemento de salida

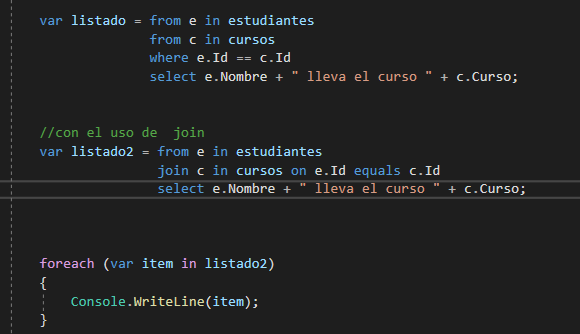
Selectmany produce de 0….. n elementos

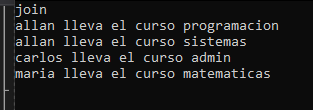
UNION:

**Join:** une los elementos de dos colecciones en un solo conjunto

Declaramos 2 clases c estudiante y c curso



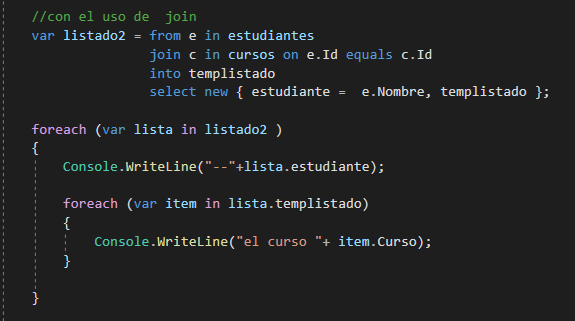


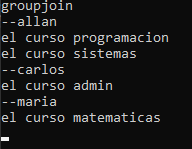


**Groupjoin:** es como join pero da un resultado jerárquico

Los resultados se obtienen de forma jerárquica

La sintaxis es la misma, pero se utiliza into

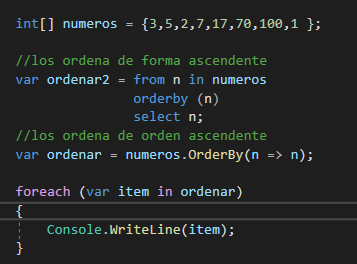


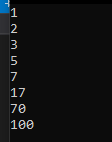


Zip: enumera 2 secuencias y aplica una función a cada par falta

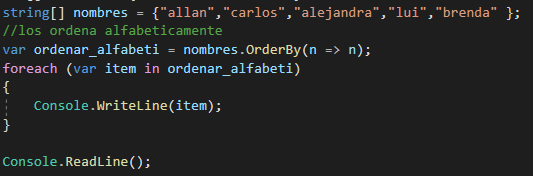
ORDENAMIENTO**:**

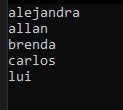
**Orderby:**

****

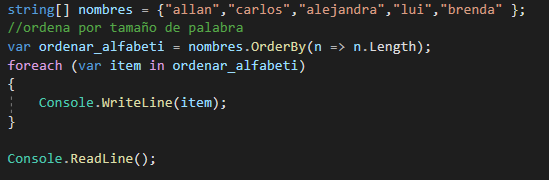
****

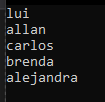
Ordena alfabeticamente

****

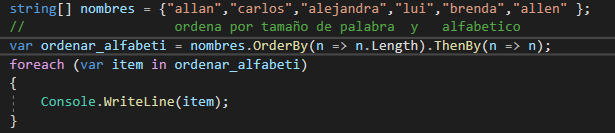
****

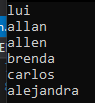
Ordena por tamaño de nombre



****

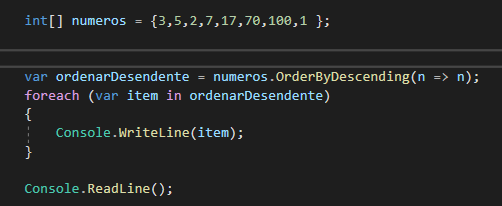
Ordena por tamaño y con el thenby ordena alfabéticamente

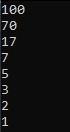




**Thenbydescending:** realiza lo mismo que thenby solo que el resultado lo muestra de forma descendente

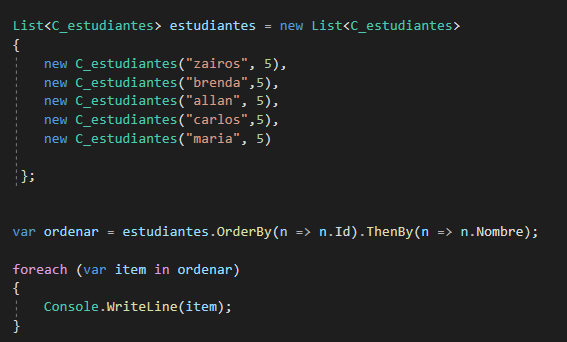
**ordenBydescending**

****

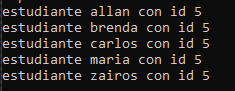
****

**Thenby:**

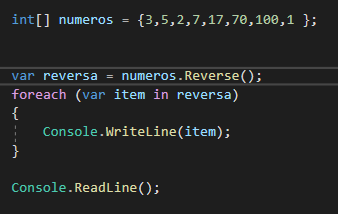
si realizamos un orderby y un grupo de elementos cumplen con la misma condicion de orden, utilizamos el thenby para poner otra condicion de orden a ese grupo.

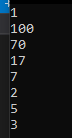


Como todos tienen id 5 entonces el thenby los ordenara alfabericamente



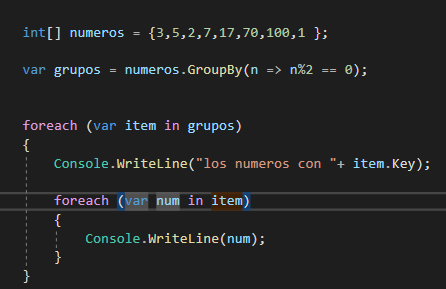
**reverse**

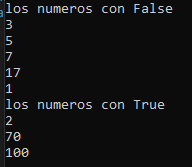




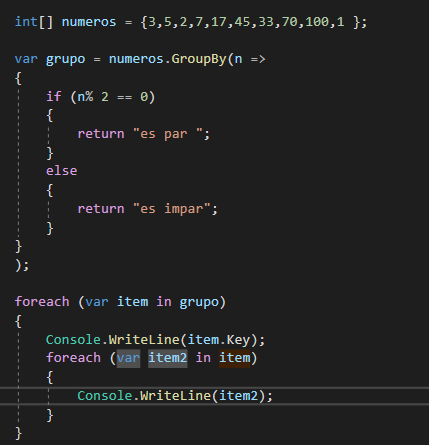
**AGRUPAMIENTO:**

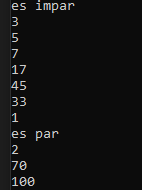
**Groupby: FALTA**

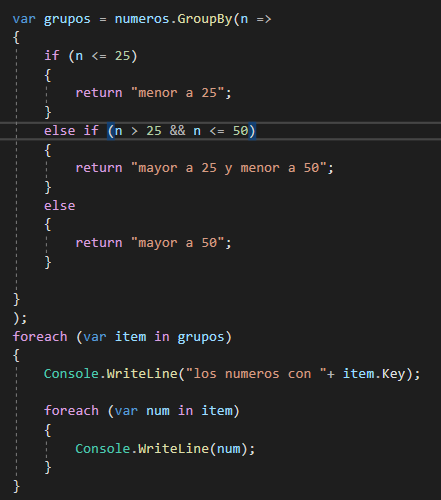
****

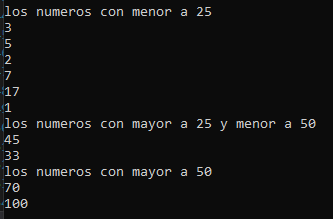
****

**Otro ejemplo**

****

****

****

****

<https://www.youtube.com/watch?v=8iW4sMyd3D0&list=PLM-p96nOrGcYGyfYpStN3jnaTuIosO3V5&index=16>

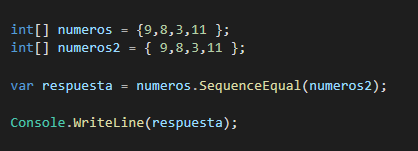
link de udemy

https://www.udemy.com/course/linq-de-cero-a-experto/

**AGREGAMIENTO:**

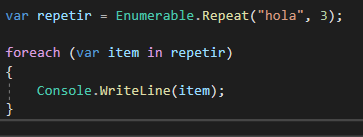
**SequenceEquals:**

Me compara si los 2 si son totalmente iguales

****

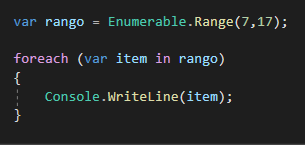
**GENERACION:**

**repeat:**

****

****

**range:**

****

****