Trabajando con arrays y listas

Cuando programamos, una de las funcionalidades que más se utilizan son las colecciones de datos, ya sean estas desde una base de datos, desde un servicio externo o creadas por nosotros mismos. Por ejemplo imaginémonos que tenemos una lista de empleados o de alumnos en un colegio.

En este tutorial veremos como trabajar con este tipo de datos, no creamos una variable para cada uno de los empleados o alumnos, sino que creamos una estructura de datos llamada colecciones. Y estas colecciones pueden ser `arrays` o `Listas` .

Índice

- 1 - Qué es un Array?
  - 1.1 - Declaración de un Array
  - 1.2 - Dar valor a los elementos de un array
  - 1.3 - Trabajar con el array
- 2 - Array multidimensional
  - 2.1 - Declaración del array multidimensional
  - 2.2 - Dar valor a los elemenos de un array multidimensional
  - 2.3 - Trabajar con un array multidimensional
- 3 - Jagged array
  - 3.1 - Declaración de un jagged array
  - 3.2 - Dar valor a los elementos de un jagged array
  - 3.3 - Trabajar con un jagged array
- 4 - Listas o List
  - 4.1 - Creación de una lista
  - 4.2 - Añadir elemenos a una lista
  - 4.3 - Trabajar con una lista
- 5 - LINQ
  - 5.1 - Qué es LINQ
  - 5.2 - Cómo utilizar LINQ

1 - Qué es un Array?

Un array es un tipo que nos permite almacenar una colección de datos de un tipo deseado. Estos tipos pueden ser tanto primitivos como **tipos** creados por nosotros mismos.

Otra de las características que contienen los arrays es que no tienen límite de tamaño y puede contener tantos registros como queramos siempre y cuando tengamos memoria suficiente en el ordenador. Pero lo importante es que el compilador no nos pondrá un límite.

Finalmente, los arrays son Inmutables, lo que quiere decir que una vez los inicializamos estos no cambian. Por ejemplo no podemos añadir o quitar elementos de un array.

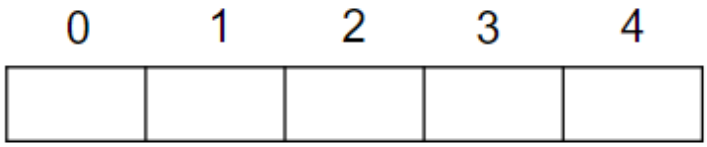
1.1 - Declaración de un Array

Para declarar un array tenemos que indicar primero de todo el tipo de dato, como hemos indicado, tanto primitivo como creado por nosotros mismos. Seguido de un corchete para abrir

y otro para cerrar `[]` que indican que es un array y cuando lo declaramos indicamos el tamaño que ese array va a tener.

```
int[] arrayEnteros = new int[5];
```

Esta seria la representación gráfica del codigo:

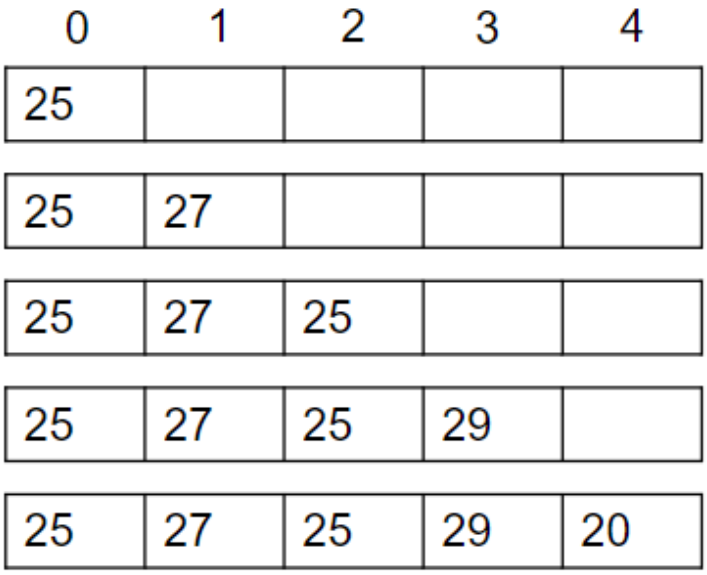


1.2 - Dar valor a los elementos de un array

Una vez tenemos nuestro array inicializado, este se crea "vacío" y lo que tenemos que hacer es darle valores. Para ello lo que hacemos es acceder a la posición del array a través del índice, el cual en programación recordamos que los índices empiezan en la posición 0. Y que el último elemento estará colocado en la posición anterior al tamaño del array, en este caso 4 (n-1 = útlima posición)

```
arrayEnteros[0] = 25;
arrayEnteros[1] = 27;
arrayEnteros[2] = 25;
arrayEnteros[3] = 29;
arrayEnteros[4] = 20;
```

De esta forma asignamos valor a los elementos del array, como vemos accedemos por posición. Esta será la representación en cada una de las líneas de código:



Como vimos en el **video sobre las variables y los operadores**, en este caso también podemos inicializar los arrays cuando los declaramos, y lo hacemos de la siguiente forma.

```
int[] arrayEnteros = { 25, 27, 25, 29, 20 }
```

Lo cual nos genera exactamente el mismo resultado que el ejemplo anterior. Pero como podemos observar no indicamos el tamaño de forma explícita, sino que lo obtiene del número de parámetros que indicamos.

1.3 - Trabajar con el array

Como hemos indicado anteriormente los arrays son inmutables, lo que quiere decir que no los podemos modificar, pero ello no nos impide poder trabajar con ellos.

Normalmente cuando tenemos una colección, ya sean listas o arrays, solemos trabajar con los datos tanto en conjunto como individualmente. Para trabajar individualmente lo que tenemos que hacer es acceder a cada uno de los elementos del array de forma individual, y ello lo haremos a través del índice.

Comúnmente accedemos a los registros uno por uno utilizando un **bucle for** de la siguiente manera:

```
for(int i = 0; i < arrayEnteros.Length; i++){
   Console.WriteLine($"El numero es: {arrayEnteros[i]}")
}
```

Como podemos observar en el bucle para indicar el segundo parámetro y saber cuantas iteraciones ejecutar hemos utilizado una función que nos viene dentro de Array, la cual es `Array.Length` .

Como `Array.Length` el propio tipo nos viene con una gran variedad de métodos que podemos utilizar en caso de que los necesitemos.

- `Array.Contains(elemento)` Devolverá `True` o `False` en caso de que el elemento esté dentro del array.
- `Array.Reverse()` Devolverá el array de forma inversa.

Como estos tenemos otros muchos métodos que iremos viendo más adelante conforme trabajemos.

2 - Array multidimensional

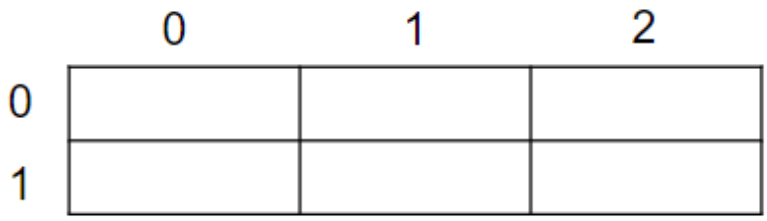
No siempre un array común va a cumplir con todo lo que necesitamos, para algunos escenarios necesitaremos un array de n dimensiones. y para ello tenemos los arrays multidimensionales.

2.1 - Declaración del array multidimensional

Muy similar al anterior, únicamente que en este caso en vez de crearnos una lista única nos creará una especie de tabla. Como en el caso anterior, para los arrays multidimensionales, también tenemos que introducir la cantidad de elementos que va a contener.

```
string[,] ciudades= new string[2, 3];
```

Como podemos observar el resultado gáfico sería algo así:



2.2 - Dar valor a los elemenos de un array multidimensional

Para acceder a los elementos del array multidimensional lo hacemos exactamente de la misma forma que accedemos en el array normal. La única diferencia es que en este caso, necesitamos dos índices.

```
ciudades[0, 0] = "Teruel";
ciudades[0, 1] = "Huesca";
ciudades[0, 2] = "Zaragoza";
ciudades[1, 0] = "Valencia";
ciudades[1, 1] = "castellon";
ciudades[1, 2] = "Alicante";
```

Y el resultado final de introducir los datos en este array multidimensional

2.3 - Trabajar con un array multidimensional

El tener que Acceder por dos índices genera que la forma de recorrer el array sea algo diferente. En este caso, necesitaremos bule for dentro de otro bucle for, para posteriormente acceder con ambos índices `ciudades[i, j]` .

```
for (int i = 0; i < 2; i++)
{
    for (int j = 0; j < 3; j++)
    {
        Console.WriteLine($"La ciudad es {ciudades[i,j]}");
    }
}
```

3 - Jagged array

Como hemos indicado en el caso anterior, muchas veces necesitamos un array con varias dimensiones. pero tenemos porque necesitar el mismo número de dimensiones para todos los elementos del array.

Para poner un ejemplo sencillo, imaginemos que queremos leer de una base de datos todos los cambios de salario de dos empleados. Para ello utilizaremos un array multidimensional, la primera de las filas del array para el primer empleado y la segunda para el segundo.

Para este ejemplo el empleado1 ha cambiado de sueldo 2 veces desde que está trabajando, mientras que el empleado2 ha cambiado de sueldo 7 veces.

Utilizando un array multidimensional, tenemos que crear un array de la siguiente manera `decimal[,] arraySueldos = new decimal[2, 7]` . Este ejemplo, llevado a una lista con 10 mil o 10 millones de registros nos puede causar unos problemas de rendimiento y memoria terribles, ya que para el primer empleado, estamos reservando 4 espacios de memoria que nunca vamos a utilizar. Para evitar este problema, disponemos de los jagged arrays.

3.1 - Declaración de un jagged array

Los jagged array se definen de una forma un tanto diferente, ya que lo que hacemos es mezclar los dos vistos previamente, el array y el array multidimensional.

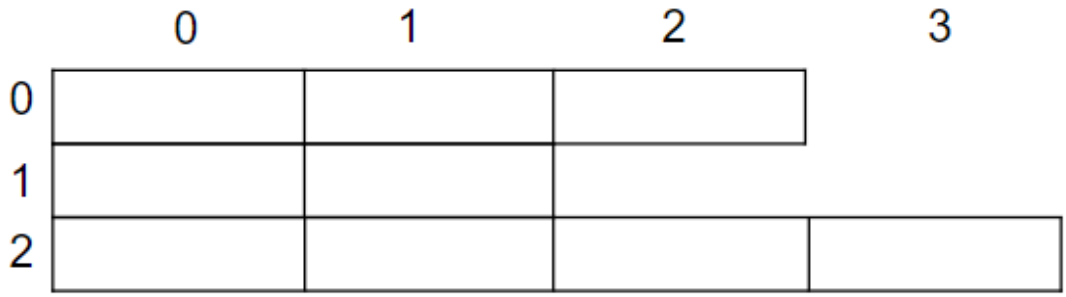
```
string[][] provincias= new string[3][];
```

De esta forma nos creará espacio para 3 arrays, cada uno independiente, a los que tenemos que asignarles un tamaño. De la siguiente manera:

```
provincias[0] = new string[3];
provincias[1] = new string[2];
provincias[2] = new string[4];
```

Como podemos observar accedemos por índice a cada uno de los elementos del array e instanciamos un nuevo array dentro del mismo. Este nuevo array es que nos indicará el tamaño del mismo.

La representación gráfica es la siguiente:



Como podemos observar hay huecos en "blanco", eso es porque ese espacio de memoria no esta utlizado.

3.2 - Dar valor a los elementos de un jagged array

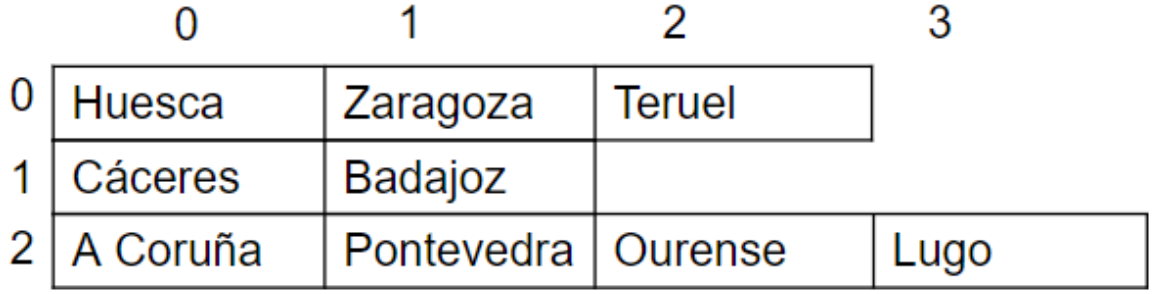
Para asignar valores debemos hacer realizar el mismo proceso, que es acceder por índice, la diferencia es que en este caso los índices no son fijos, por lo que a la hora de leer lo haremos de una forma diferente, para asignar valores, lo hacemos directamente, ya que conocemos, o debemos, el tamaño.

```
provincias[0][0] = "Huesca";
provincias[0][1] = "Zaragoza";
provincias[0][2] = "Teruel";

provincias[1][0] = "Cáceres";
provincias[1][1] = "Badajoz";

provincias[2][0] = "A Coruña";
provincias[2][1] = "Pontevedra";
provincias[2][2] = "Ourense";
provincias[2][3] = "Lugo";
```

Y la rerpresentación seria la siguiente:



3.3 - Trabajar con un jagged array

En este caso el array es totalmente dinamico, para cada uno de los registros tenemos un tamaño diferente, por lo que para recorrerlo debemos consultar el tamaño de cada uno de los registros, como hemos visto antes con .Length.

```
for (int i = 0; i < provincias.Length; i++)
{
   System.Console.Write($"registro ({i}): ", i);

   for (int j = 0; j < provincias[i].Length; j++)
   {
         System.Console.Write("{0}{1}", provincias[i][j], j == (provincias[i].Length
   }
   System.Console.WriteLine();
}

//Resultado
registro (1):
Hueca Zaragoza Teruel
registro (2):
Cáceres Badajoz
registro (3):
A coruña Pontevedra, ourense Lugo
```

Como nota final indicar que podemos mezclar los arrays multidimensionales con los jagged array, pero sinceramente no lo recomiendo, hay formas mejores de llegar a una solución, por no hablar de que apenas se ven en el mundo laboral.

4 - Listas o List<T>

Ahora dejamos atrás los arrays para pasar a uno de los elemenos mas utilzados en el entorno C#, las listas o `List<T>` y es un tipo de dato utilizado tan asiduamente gracias a su facilidad y a su gran funcionalidad.

`List<T>` es un tipo de dato, similar al tipo `array`, que acabamos de ver, pero que en este caso son mutables, lo que quiere decir que podemos añadir o quitar elementos cuando queramos.

Otro dato muy importante a tener en cuenta sobre las listas es que implementan `IEnumerable<T>` la cual es una interfaz que permite ejecutar consultas `LINQ`, las cuales veremos en este mismo post. Debido a ello el uso de las listas es continuo en nuestras aplicaciones.

4.1 - Creación de una lista

Cuando creamos una lista a diferencia de los arrays no utilizamos un índice, ya que estas al ser mutables no tienen un tamaño predefinido desde la inicialización. Para replicar el ejemplo

anterior, para crear una lista de string lo hacemos de la siguiente forma:

```
List<string> provincias = new List<string>();
```

Como vemos la `T` de `List<T>` es el tipo de dato que queremos utilizar, y otra vez, este tipo puede ser tanto primitivo como no primitivo.

4.2 - Añadir elemenos a una lista

Como hemos indicado anteriormente, ya no accedemos por índice, por lo tanto para añadir elemenos lo hacemos de una forma mas dinámica con el metodo `.Add(T)` en el cual debemos pasar por parámetro un tipo de dato del cual es la lista, en nuestro caso `string` .

```
provincias.Add("Teruel");
```

además de esto, si tenemos una lista y queremos añadirla dentro de otra lista, por ejemplo, estamos listando todas las provincias y primero las listamos por comunidades, podemos añadir a una lista, otra lista, utlizando .AddRange(List<T>)

```
aragon.Add("Huesca");
aragon.Add("Zaragoza");
aragon.Add("Teruel");

provincias.AddRange(aragon);
```

Como observamos en el ejemplo, pasamos por parámetro la lista de aragon.

4.3 - Trabajar con una lista

En este caso, para trabajar con una lista podemos utilizar cualquiera dentro de la infinidad de métodos que el compilador nos provee, aquí vamos a ver sólo unos pocos a modo de ejemplo.

Primero vamos a ver cómo iterar la lista, para ello utilizaremos un bucle `foreach`

```
foreach (string provincia in provincias){
   Console.WriteLine($"la provincia es {provincia}");
}
```

Más métodos que podemos utilizar:

- `lista.ToArray()` ; nos convierte la lista en un array.
- `lista.Count` ; nos devuelve un entero con el número de elementos dentro de una lista.
- `lista.First()` ; nos devuelve el primer elemento.
- `lista.Last()` ; nos devuelve el último elemento.
- `lista.Clear()` ; vacía o remueve todos los elemenots de la lista.
-

5 - LINQ

Primero de todo indicar que lo que vamos a ver es una pequeña introucción al lenguaje `LINQ` ya que verlo todo completamente nos da para hacer un centenar de videos.

5.1 - Qué es LINQ

LINQ es un lenguaje de consultas a bases de datos, o al menos esa era su intención original. Mientras LINQ se estaba desarrollando se dieron cuenta que una List<T> no deja de ser los resultados que obtenemos sobre hacer una consulta sobre una BBDD así que porqué no aprovecharlo y poder utilizarlo dentro de nuestro código, no solo para consultar, sino, para filtrar.

5.2 - Cómo utilizar LINQ

Utilizar LINQ dentro de nuestro código es muy sencillo, tan solo tenemos que utilizar un tipo que extienda de la clase `IEnumerable<T>` , como puede ser LIST<T>.

Para utilizarlo indicaremos: lista `.Where(condicion)`

Dentro de "condicion" debemos indicar el elemntos en el que estamos, comunmente yo utilizo la letra a pero mucha gente utiliza la inicial del elemento que estamos iterando, en este caso (ejemplo provincias) la p.

En nuesttro ejemplo `provincias.Where(p => p)` con `p=>p` indicamos donde se va a ejecutar esa consulta, que a su vez, pasa individualmente por cada uno de los elementos de la lista.

Para añadir condiciones a la consulta podemos utilizar las mismas condiciones que disponemos en sus tipos, en este ejemplo para el tipo string utilizaremos el método `.Contains()` que nos devuelve verdadero o falso si el elemento a comprobar contiene el valor pasado por parámetro.

Posteriormente recorremos el resultado imprimiendo cada uno de los registros.

```
provincias.Add("Huesca");
provincias.Add("Zaragoza");
provincias.Add("Teruel");

IEnumerable<string> provinciasFiltradas = provincias.Where(p => p.Contains("e"));
foreach(string provincia in provinciasFiltradas){
   Console.WriteLine(provincia);
}
```

En este escenario imprime tanto Huesca como Teruel, ya que ambas contienen la letra e (minúscula).