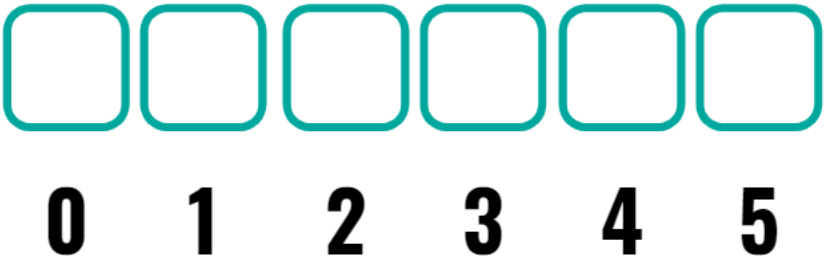
23. What is an Array?

Brief summary: Array is the basic collection type in C#, storing elements in an indexed structure of fixed size. Arrays can be single-dimensional, multi-dimensional, or jagged.

With this lecture, we start a series about collection data structures in C#. We will begin with the simple but common question from the interviews: What is an **array**?

Array is the most basic collection in C#. You can think of an array as a collection of boxes, each one holding a single value. Each box has its index, **starting at zero** and ending at array length minus 1.



The important thing to understand is that **once an array has been created, its size cannot be changed**. Because of that arrays are not the best choice if the collection that we need is going to grow or shrink over time.

If we try to get or set the value at an index that's not in the array, we will get **IndexOutOfRangeException**. In the below code, we declare an array of size 5, so its last index is 4, but we try to access the element at index 10.

```
var array = new int[5];
array[10] = 7; //this will throw an IndexOutOfRangeException
```

When an array is created, it is filled with the default values for the given type. For example, an array of ints will be filled with zeros, and an array of strings will be filled with nulls.

We can set the values of the array right at the moment of initialization using the **collection initializer**. In this case, we don't need to specify the array's size, as it will be set to the count of provided elements:

```
var stringsArray = new [] {"a", "b", "c"};
```

The above code will naturally create an array of size 3.

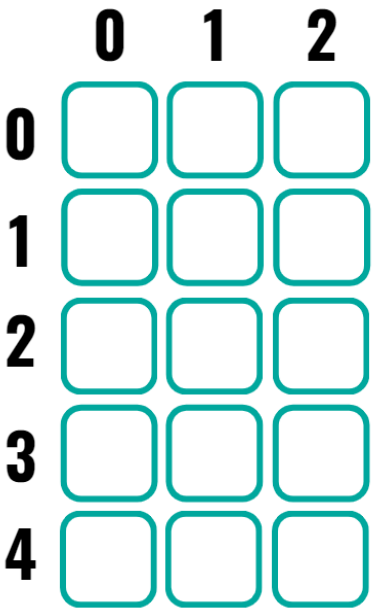
Arrays can store any objects of the same type. Arrays can be single-dimensional, multidimensional, or jagged. For example, this is a single-dimensional array that holds 5 ints.

```
var numbers = new int[5];
```:

We can also define multidimensional arrays that resemble matrices we know from mathematics. For example, this array can hold up to 15 elements:

```
var matrix = new int[3, 5];
```

The data held in this array can be visualized like this:



We can think of a multidimensional array as an array of arrays, which all have the same length. In this case, we have an array of size 3, and at each index, there is an array of size 5.

To get or set an element of a multidimensional array, we must simply use two indexes instead of one, separated by a comma:

```
matrix[1, 1] = 10;
```

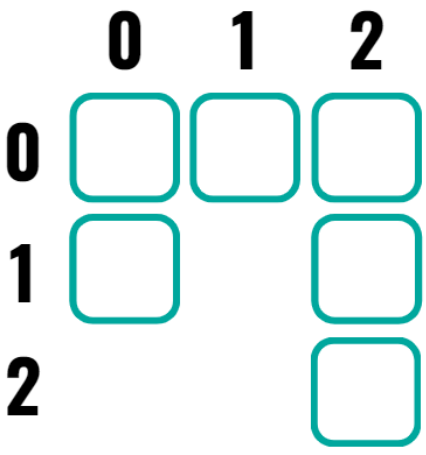
We can also define **jagged** arrays. A jagged array is an array of arrays, which don't need to be of the same length. Let's define a jagged array of integers.

```
var jagged = new int[3][];
```

Here we defined an array of size 3, for which each element will be an array of ints. Let's set those elements:

```
var jagged = new int[3][];
jagged[0] = new int[2];
jagged[1] = new int[1];
jagged[2] = new int[3];
```

It means the structure of this jagged array looks like this:



At index 0 we have an array of length 2, at index 1 of length 1, and at index 2 of length 3.

Please notice the difference in accessing the elements of a jagged array, in opposite to a multidimensional array. We must use **two** sets of brackets:

```
jagged[2][1] = 7;
```

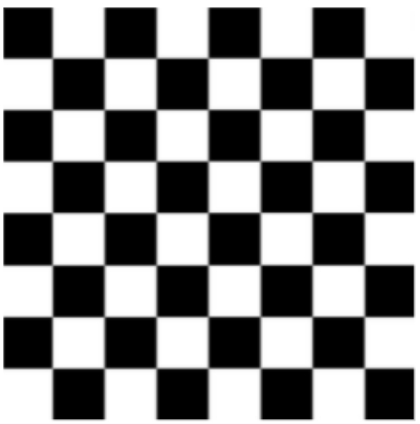
Before we continue, quick information for people with a background in languages like C or C++. Remember that in C# an **array is just an object as any others**. It's not an addressable region of memory like in those other languages. Arrays are **reference types** in C#.

All right. Let's see what arrays are best for, and when we should rather consider using other collection types.

First of all, arrays are **the most basic and native collection type** - they represent the data in a way that is very close to how the data is stored in the computer's memory. Many other collections, like for example Lists we will learn about in the next lecture, **use arrays as their underlying data structures**.

Arrays, as the most native collection type, have the advantage of being **fast**. We can get or set an element at the given index in a **constant time** (so "super-fast" in less technical terms). When we care about the **performance**, arrays are often the best choice.

Also, multidimensional arrays are commonly used in **mathematical operations**, as they represent matrices very well. They are also very useful wherever we need to represent any multidimensional structure - think of a 2D game with a tiled map, like chess or snake. The two-dimensional array of tiles would probably be the most natural and performance-efficient way to represent the game area:



The **disadvantage** of using arrays is that their **size is fixed**. In other words, it's not a dynamic collection - we can't really add or remove elements from it.

We could assume that some size of an array is large enough for our needs. Let's say we work on an e-commerce platform, and we want to store the items held in the shopping cart in an array. We can declare an array of size 100 and hope no one will need more space. But such assumptions are dangerous. What if someone is in a shopping rage and they really want to buy more things? Okay, so let's set it to 1000. But then, in 99% of the cases, we will allocate the memory for 1000 items, while actually only 2 or 3 will be added. This is a huge waste.

In short, arrays are great if we know the count of elements upfront, but not so much if we do not.

Bonus questions:

- "**What is a jagged array?**"
  A jagged array is an array of arrays, which can be all of the different lengths.

- "**What are the advantages of using arrays?**"
  They are fast when it comes to accessing an element at the given index. They are basic and easy to use and great for representing simple data of size that is known upfront.

- "**What are the disadvantages of using arrays?**"
  Arrays are of fixed size, which means once created, they can't be resized. It means that are not good for representing dynamic collections that grow or shrink over time. If we want to allocate the memory for all elements that may be stored, there is a chance we will allocate too much and waste it. We can also underestimate and not declare the array big enough for some edge cases.

- "**How to resize an array?**"
  It's not possible. An array is a collection of a fixed size and once created, it can't be resized.