Resizable arrays and language support

Selecting transcript lines in this section will navigate to timestamp in the video

- Some languages let you resize basic arrays and others do not. When I say resize I mean that you are able to add additional elements and not just interchange values within a specific set of slots. For example, in Java and C++ once a basic array is created it cannot be resized. This means if you create an array with seven slots, or an array that can store seven items, then it will always have seven slots. You cannot change the array to have six slots or eight slots or any other number for that matter. You can change the values that are in those seven slots but there can only ever be seven slots if that's how the array is created. Other languages are not as strict, such as Ruby and JavaScript which let you add and remove items without worrying about how many slots are initially in the array. While the basic array functionality for languages can be different, most provide some kind of resizable array behavior when you need it. You might hear these resizable arrays called dynamic arrays or mutable arrays. But when at mean here is we can change the size of the array. For example, in Java we have an immutable basic array data type. Meaning the size of it cannot be changed. However, we also have a few Java specific classes, or to keep it simple think Java specific data structures that give us a resizable version. The most common of these is the mutable and resizable ArrayList. With an ArrayList we get extra methods that we can use like add and remove and get, in order to access and change the items in our data structure. However, although an ArrayList is Java specific and other languages have similar implementations, that does not mean that they're baked into the language. We have to import this functionality in our code in order to use it. But, under the surface, an ArrayList is just an array. Under the surface the programming language is dealing with actual arrays, and the ArrayList allows us to focus less on maintaining the data structure and more on our data and what we actually want to create. This means when we insert an item to an ArrayList in Java, it's just one line of code. For example, to insert the value 10 at index two, we don't have to worry about the arrays and space allocation and all of that other stuff. Now I'm going to throw you into some code, but don't worry too much about the syntax. Essentially, in this piece of code, we create an ArrayList called myArrayList, add the numbers zero through 10 back to back and then insert the number 10 at index two. At the end we print the modified ArrayList. Notice we have an import statement at the top, too, that allows us to use the ArrayList. When this code is executed we get this output. You've probably noticed this in the print out, but if we do not add an index as an argument to the add function, the item is just added to the end of the ArrayList, and this is true for most programming languages. Of course, the terminology differs depending on the language, but usually add or push is adding items to the back of the array, and remove or pop is removing items from the end of the array. In some cases it might be different. For example, in JavaScript we use splice instead of push. Of course, you don't need to memorize all of this syntax. As you work with a specific language you will pick up on how to work with a dynamic array. Understanding that most languages provide dynamic array functionality is key. Now since many programming languages deal with basic non-mutable arrays under the surface, they can implement this insert functionality in one of two ways. If the array is big enough the language will shuffle all of the items down and add the new one. If not, it will copy all the contents into a new basic array into their new places and simply coy the new item over with it. This can have major performance repercussions, depending on how many items you are dealing with and the way the language handles its data structures. So be careful.