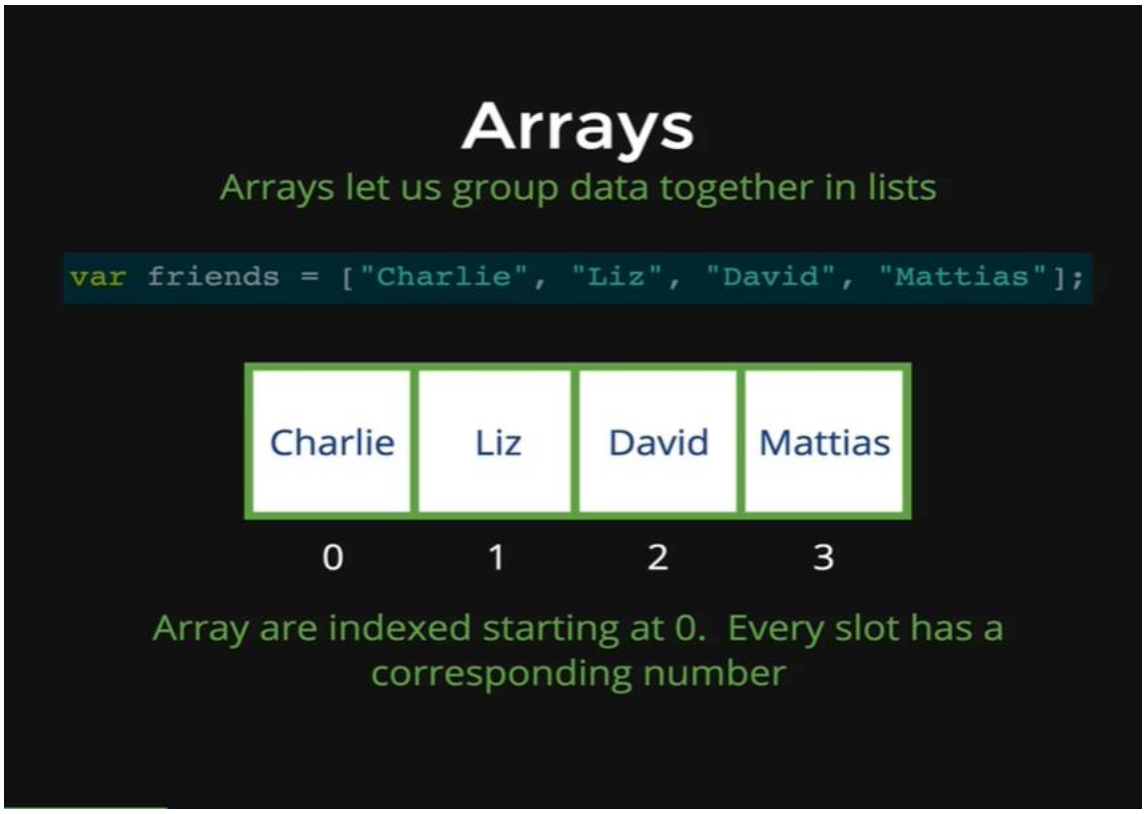
Objectives:

- *Understand arrays conceptually*
- *Write code using JS arrays*

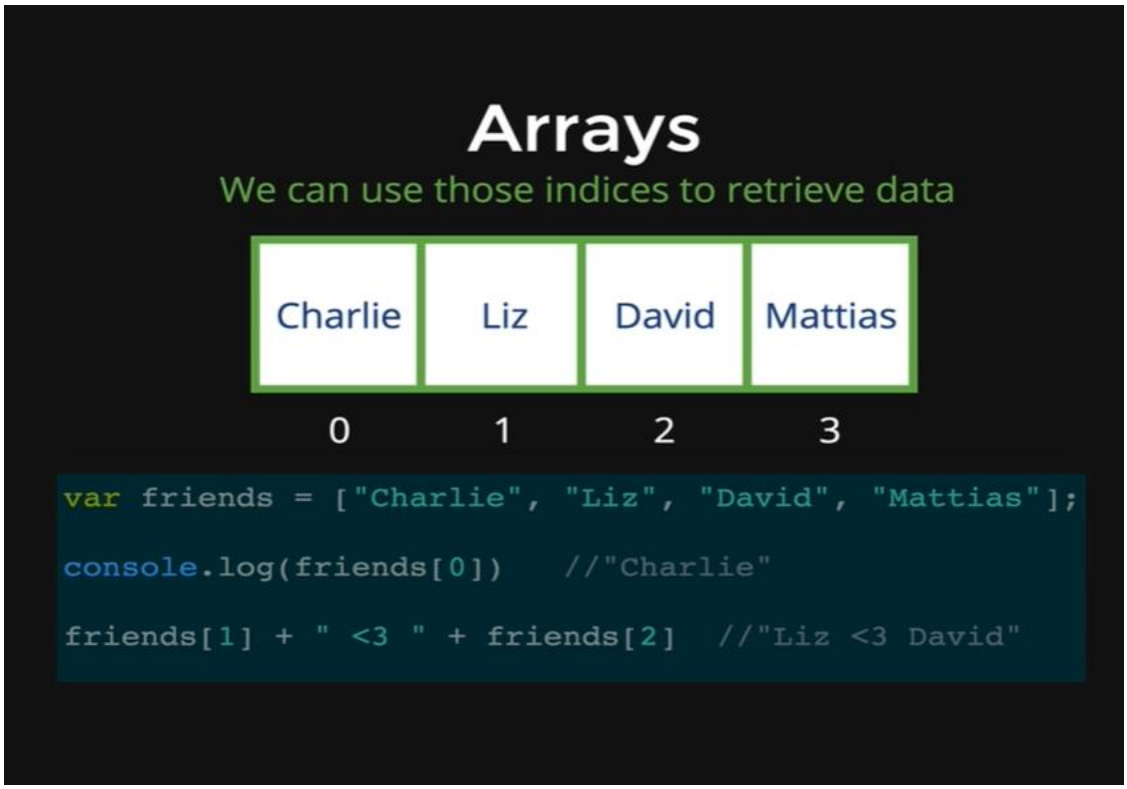
In this lesson we are going to learn about our first main data structure, which is called Array. Let's see what arrays are conceptually.

Suppose we want to model a group of friends. Every friend is a string with their name. Like, Charlie, Liz, David and Mattias. We can make four friends variable. Each one is a separate variable and if we want to add another one, we can say var friend5 equal to some other name, and then for the next one we need to do friend6, then friend7. This is problematic for few reasons, first is this is not the code that we will consider DRY as there are lot of repeated code here. Also, these friends are not related to one another, they are totally separate. It is not really a group of friends.

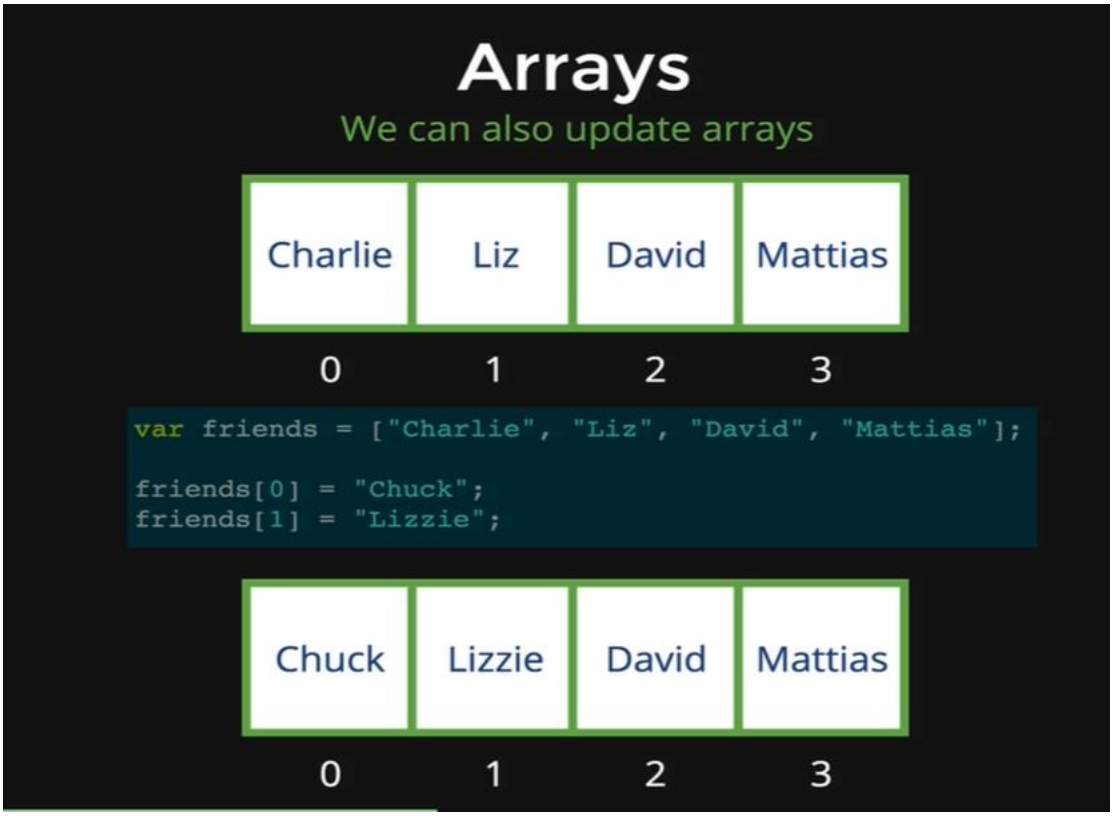
That's why it is a perfect use case for an array. An array lets us group data in a list. So, rather than writing four different variables, we can write one variable and inside of it, we can have four different names. Let's take a closer look at how arrays work.



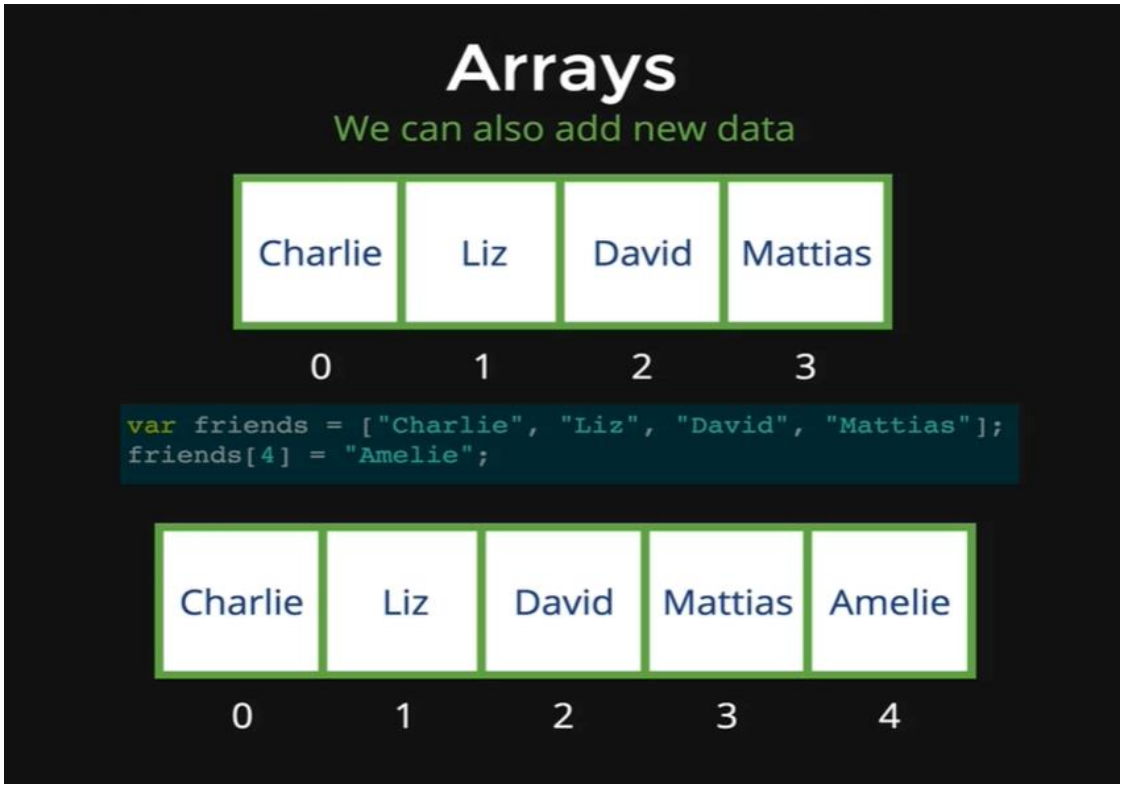
We have the same example code here, var friends equal to square brackets and that is what tells JavaScript that this is an array. Up until this point here, we are dealing with a regular JavaScript variable, where on the right side we could just have a string or a number or a Boolean or undefined or null but instead we have an array. The square brackets represent an array and inside them we have our list of data. To declare an array, it is a comma separated list, so we have our values, “Charlie”, “Liz”, “David”, “Mattias” and that results in an array of four items, the second important piece of information about arrays is that they are indexed, just like characters in a string, where there is a specific number that corresponds with every character starting at 0, arrays are also indexed starting at 0. Every slot in this array has a corresponding number, when we made up this array with all the names, the diagram with boxes shows how the arrays are made in memory. We have “Charlie” stored with number 0, “Liz” with 1, “David” with 2 and “Mattias” with 3.



We use array indices all the time to get the data out of the array. Here we are initializing the same array, and we need to get "Charlie" out of the array, we need to know the index that corresponds to Charlie, which is 0 and then all we do is write friends as the variable name and then inside the square brackets 0, just like we would do it with a string if we want the first character. Again, the difference is we are not dealing with a string of characters, we are dealing with an array of strings. So, *friends[0]* is going to give us *"Charlie",* likewise if we do *friends[1]* that gives us *"Liz"* and *friends[2]* gives us David, which is why we get *"Liz <3 David".*



We can also use the indices to update array values, so if *"Charlie"* wants his name to change to *"Chuck",* all we have to do is find *"Charlie"* in the array with index 0, and then we write *friends[0]* and set it equal to another string, in this case with *"Chuck".* Then our array looks like the diagram below where we have Chuck with index 0. Same thing with Liz, she changes her name to Lizzie, then we find the correct index for Liz which is 1, and *friends[1]* is set equal to "Lizzie" and our array looks like Chuck at 0 and Lizzie at 1.



The next feature of arrays is that we can also add data to an existing array. So, if we get a new array, then all we need to do is write the variable name which is friends, followed by square brackets, the brackets carrying an index number which does not exist yet. In this case that number is 4, and if we do *friends[4]* is equal to *"Amelie",* then we will get *Amalie* added to our array at index 4.

Now we are going to do some demonstration at our console, we are going set an array to model colors.

var colors = ["red", "orange", "yellow"];

We now have our colors array defined and we have three elements in it. If we access colors, it shows us the three items in it.

colors

➔ *["red", "orange", "yellow"]*

If we want to print out orange, then we code as follows,

colors[1]

➔ *“orange”*

We used index of 1, because orange has index of 1 and then we get orange. If we want to add in another element after Yellow, for example Green, then we need to count from 0 and see that index of 3 is the empty array after “Yellow”.

colors[3] = “green”

➔ *“green”*

Now if we look at colors, we will get our full and updated list.

colors

➔ *[“red”, “orange”, “yellow”, “green”]*

Suppose now we want to change green to dark green, then we need to access the index of green that is 3 and we set the variable at that index to be dark green.

colors[3]

➔ *“green”*

colors[3] = “dark green”

➔ *“dark green”*

That’s it.

colors

➔ *[“red”, “orange”, “yellow”, “dark green”]*

Now quick thing we need to note that if we add an element to a direct empty index but after some empty indexes then it will be as below

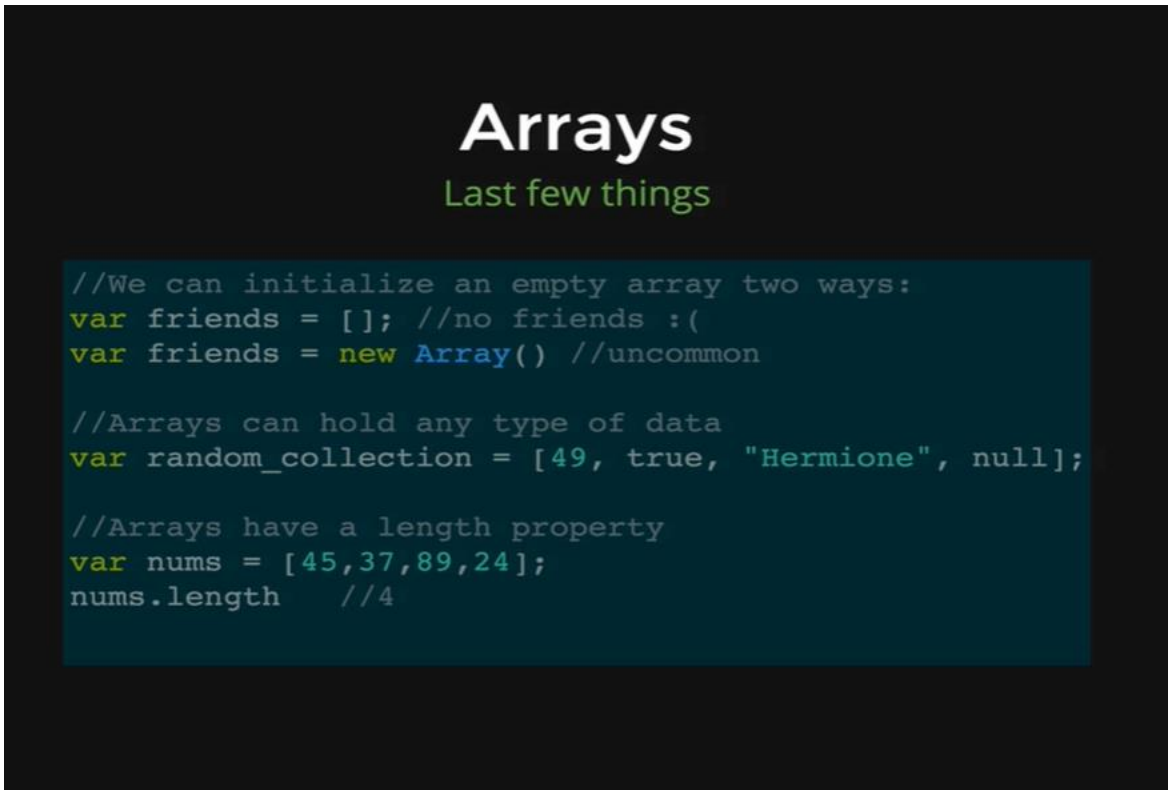
colors[10] = "violet"

➔ *"violet"*

colors

➔ *["red", "orange", "yellow", "dark green", undefined x 6, "violet"]*

We can see that we get red, orange, yellow, dark green, and then it shows undefined times 6, so what happened here is we added violet at index of 10, and that leaves us lot of blank spaces between index of 3 and index of 10. So, JavaScript makes empty spaces there and fills them with undefined. So, there are a bunch of empty spaces in this array, which is not ideal.



There are different ways of defining arrays. The first way is to define an empty array, using square brackets. Here we are defining a friends' array, but we do not have any friends

unfortunately, so its just empty square brackets. Another way of defining array that we would come across is to write *new* and then *Array* with a capital A and then parentheses after it. This is a function, we are calling it with parentheses that makes us a new array, just like the previous friends' array. In both cases we have friend arrays, and those are empty.

The next thing is that arrays can hold any types of data, so far, we have just seen arrays with strings, but we can also store them with numbers, strings and other kinds of data. So, we can fill arrays with all sorts of data, and they all don't have to be of one type. Here, we have an example, an array variable with name *random_collection* which holds a number, a Boolean, a string and a null, all in one array.

The last thing we need to point out that arrays also have a length property just like strings. Here, we are defining an array variable with nums, containing four numbers, and then we run the length property by writing *nums.length* and we get 4 as our output.

Let's demonstrate the *length* property in arrays.

var dogs = ["Rusty", "Wyatt", "Olly"]

Here, we are defining an array variable with dog names, Rusty, Wyatt and Olly. Now if we run *dogs.length,* we get 3, because *length* property just counts how many items are there in the array. But always remember that the highest index in this or any array is one less than the length, so Olly is in index 2.

dogs[2]

➔ *Olly*

This is important we can define a string as below.

var name = "Rusty"

Now if we do name.length we get an output as below.

name.length

➔ *5*

name[4]

➔ *"y"*