Objectives:

Arrays come with a few built-in methods that make our life easier. We're going to cover:

- *push/pop*
- *shift/unshift*
- *indexOf*
- *slice*

In this lesson, we are going to learn about some built in array methods, that every single array comes with, that are going to make arrays even more useful for us. There are six methods that we are going to cover here. *Push* and *pop* are paired together and then *shift* and *unshift* are also a pair, and then we are going to talk about *indexOf* and *slice*.

Let's start by talking about *Push* & *Pop*. We already saw that if we want to add an item to our array then we need to access the index of an empty array and then initialize the array of that index with the item. If we want to add green in our colors' array, then we would do it as below

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

colors[3] = "green"

➔ *"green"*

colors

➔ *["red", "orange", "yellow", "green"]*

colors[4] = "blue"

colors = ["red", "orange", "yellow", "green", "blue"]

There is a built-in method however that makes this much easier, this is something we do a lot, which means *push* into an array. *Push* refers to adding something to the very end of an array. On the picture above you can see we also have a colors' array and to push into it, to add green to the very end, we write *colors.push(green)* to add green in our array. Let's demonstrate this with an example,

colors

➔ *["red', "orange", "yellow", "green", "blue"]*

colors.push("indigo")

➔ *6*

colors

➔ *["red", "orange", "yellow", "green", "blue", "indigo"]*

As you can see, we passed indigo as an argument using the push property in the colors' array and it has been added to the end of our array. We did not even have to specify the index of the array

to add it in, but after we used the push property, it even returned the length of the array (6) after it adds the color in. You can see that the array returns 6 and we have 6 items in our array.

We will push one more time

colors.push("violet")

➔ *7*

We pushed violet in our array, and it returned us 7 as there are now 7 items in our array.

colors

➔ *["red", "orange", "yellow", "green", "blue", "indigo", "violet"]*

The next method is the sister method of push, which is called *pop*. Pop does the opposite of push; it removes an item from an array. Let's see how it works on our colors' array.

color.pop()

➔ *"violet"*

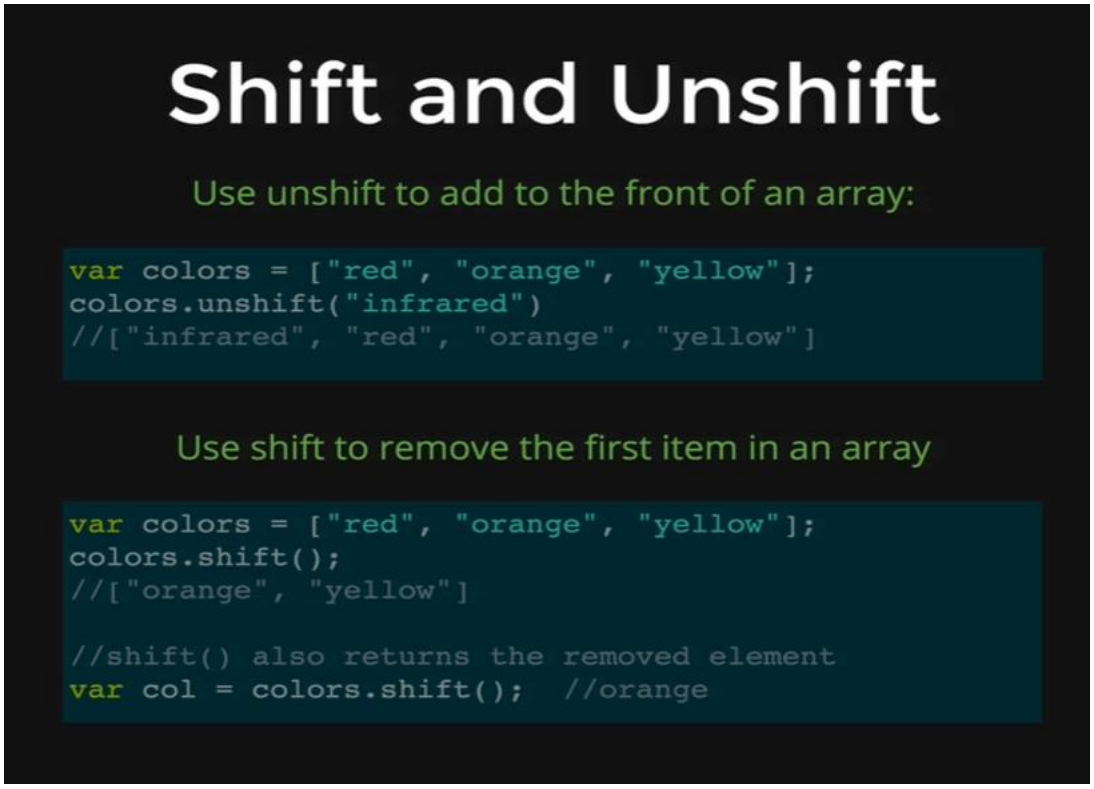
It returns us the last item that was in our array, which was violet, now let's check our array.

colors

➔ *["red", "orange", "yellow", "green", "blue", "indigo"]*

As you can see, our array does not have violet anymore. So, we use *pop* a lot, to remove something from an array and do something with them.

Push takes one argument and it adds to an array and returns the length of that array and *pop* takes no argument and returns the last item in our array that it removes from our array.



Next, we have *shift* and *unshift*, and this is a pair, just like *push* and *pop*, and they work very similarly, except rather than adding and removing to the end of the array, they add and remove from the beginning of the array. The names *push, pop, shift* and *unshift* originated from the data structures called *stacks* and *queues*, that exist in other programming languages. They are sort of just leftovers; they exist mainly because they have been used for a long time in other programming languages.

Shift and *Unshift* are opposites of *push* and *pop*. We can use unshift to add to a front of an array, and we know it is confusing that its called unshift when we are adding something, but that is just how it is, if we want to add *infrared* at the beginning of our array, we can use *colors.unshift(infrared)* and that would add infrared to the beginning of the array.

Then we have *shift* that removes an item from an array and then returns that item. Here, we can use *colors.shift()* and it removes, the first item red from the array and returns it to us, now we only have orange and yellow in our array.

If we initialize the shift method on our array to a variable, then that variable will store that item.

var col = colors.shift();

If we apply the shift method on that array, it will remove orange from our array and orange will be stored in the *col* variable. Let's demonstrate this with an example.

var nums = [34, 54, 22]

nums.unshift("HELLO");

➔ *4*

nums

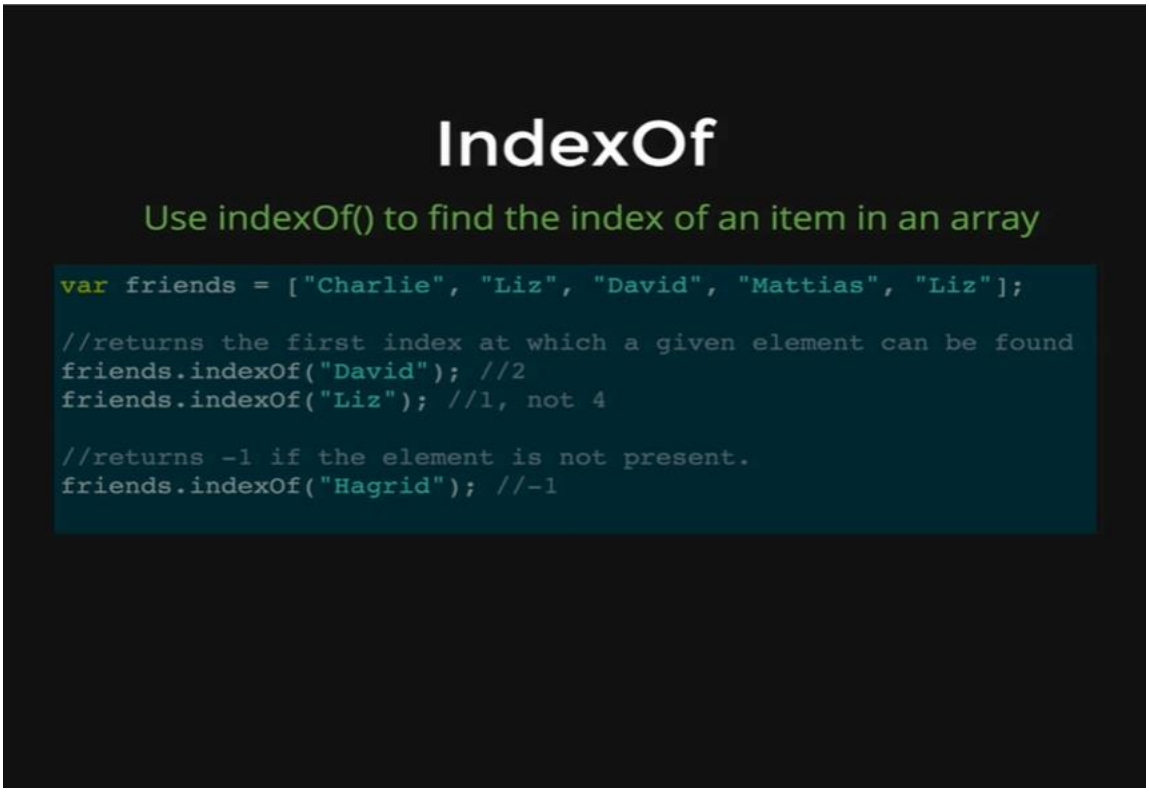
➔ *["HELLO", 34, 54, 22]*

nums.shift()

➔ *"HELLO"*

nums

➔ *[34, 54, 22]*



The next method is called *IndexOf,* what it does is that it takes an argument, like a string or a number and it tries to find an argument in a given array and if it finds it, it will return the index of where it is found.

We have an example here of a friends' array, and in this array, we have five different friends, Charlie, Liz, David, Mattias and another Liz. If we wanted to know where David is located in the

array, we write *friends.indexOf(David)* and we need to make sure that it matches exactly, it is going to go and find the string in the array, which is the third item with index of 2, and that's why it would return 2.

If we try it on Liz, however it is going to return the first instance of Liz. So, amongst the two Liz, its going to return the first Liz, which has index of 1, and not the one that has index of 4.

We can also use *indexOf* to determine if an element is not present in an array, and to do that we just check to see if *indexOf* returns -1. That is how it behaves if it does not find a given argument. Let's try it with an example.

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

colors.indexOf("yellow");

➔ *2*

We made a colors' array, now we want to know where the color yellow is located, so we passed "yellow" as an argument in the *indexOf* method applied over the colors array and we are returned with 2, thus telling us that the color yellow is at 2. Now we can access the color by *colors[2]*.

colors[2]

➔ *"yellow"*

If we want to know if green is in our array, then we do it as follows.

colors.indexOf("green");

➔ *-1*

It returns us -1, thus telling us that green is not in our array.

The last method that we want to talk about is called *Slice.* We use Slice to copy different portions of an array. We have an example of fruits here, an array with 'Banana', 'Orange', 'Apple' and 'Mango'. If we want to copy the orange and lemon out of this array, with just orange and lemon we can use a new method called *slice().* This method takes two arguments, the first argument is the index number of the item where the cut begins, so here it is 1, which represents Orange, and the second argument is where the slice ends, which is 3 which represents Apple, though we are not extracting Apple, our slice ends at Lemon.

When we run the variable citrus, it will store a new array with Orange and Lemon. It is also important to know that the original array of fruits is unaltered, it still contains all the items that it had before we applied the method *slice()* on our array.

We can also slice an entire array, in that case we do not pass any arguments, we just use the name of the array as *fruits.slice()* to get the full array. Let's demonstrate with an example.

var nums = [1, 2, 3, “a”, “b”, 445, 34]

nums

➔ *[1, 2, 3, “a”, “b”, 445, 34]*

var letters = nums.slice(3,5)

letters

➔ *[“a”, “b”]*