3a Theory JS API pages 39 - 50 Arrys, Adding & Removing Elements

Nicholas Bruzzese

July 24, 2025

Introduction

Today, we're diving into the world of **arrays** in JavaScript, based on pages 39–50 of the *3a Instructions JS API*. Arrays are super handy for organising data, and we'll explore what they are, why they're useful, how to create them, and how to add or remove elements.

What Are Arrays?

Imagine you've got a list of your favourite things—say, your top dinosaurs, like T-Rex, Velociraptor, and Stegosaurus. You could store each one in a separate variable, like this:

```
var dinosaur1 = "T-Rex";
var dinosaur2 = "Velociraptor";
var dinosaur3 = "Stegosaurus";
```

But if you had 100 dinosaurs, that'd be a nightmare to manage! You'd have 100 variables, and keeping track of them would be a mess. This is where **arrays** come in. An array is like a single shopping list that holds all your items in one place. In JavaScript, an array is a list of data values (like strings, numbers, or even other arrays) stored together in a single variable.

For example, instead of three separate variables, you can store your dinosaurs in one array:

```
var myTopThreeDinosaurs = ["T-Rex", "Velociraptor
    ", "Stegosaurus"];
```

Now, you've got one variable that holds all three dinosaurs. Arrays make it easier to work with lists of data, whether it's three dinosaurs or a thousand!

Why Should You Care About Arrays?

Arrays are awesome because they save you from creating heaps of variables. Think of it like having all your shopping items on one piece of paper instead of separate sticky notes for eggs, bread, and milk. With arrays, you can:

- **Group related data**: Keep all your dinosaurs, scores, or favourite games in one place.

- **Easily manage large lists**: Whether it's 3 items or 3000, arrays make it simple to store and access them.
- **Work smarter**: Arrays let you do cool things like add, remove, or change items without juggling multiple variables.

For example, instead of:

```
var dinosaur1 = "T-Rex";
var dinosaur2 = "Velociraptor";
var dinosaur3 = "Stegosaurus";
var dinosaur4 = "Triceratops";
```

You can have:

```
var dinosaurs = ["T-Rex", "Velociraptor", "
   Stegosaurus", "Triceratops"];
```

One variable, one list, way less hassle!

Creating an Array

Creating an array in JavaScript is super easy. You use square brackets `[]` to define it. An empty array looks like this:

```
var emptyArray = [];
```

But empty arrays aren't much fun. Let's fill one with dinosaurs:

```
var dinosaurs = ["T-Rex", "Velociraptor", "
   Stegosaurus", "Triceratops", "Brachiosaurus",
   "Pteranodon", "Apatosaurus", "Diplodocus", "
   Compsognathus"];
```

Each item (or **element**) in the array is separated by a comma. In this case, the elements are strings (dinosaur names), but arrays can hold numbers, other arrays, or even a mix of different data types.

You can also write an array with each element on a new line to make it easier to read:

```
var dinosaurs = [
"T-Rex",
"Velociraptor",
"Stegosaurus",
"Triceratops",
"Brachiosaurus",
"Pteranodon",
"Apatosaurus",
"Diplodocus",
"Compsognathus"
];
```

Both formats work the same way in JavaScript. The new-line version is just easier on the eyes, especially for long lists.

Accessing Elements in an Array

Each element in an array has an **index**, which is like its address in the list. Indexes start at **0** (not 1!), so the first element is at `array[0]`, the second at `array[1]`, and so on.

Using our dinosaurs array:

```
var dinosaurs = ["T-Rex", "Velociraptor", "
   Stegosaurus"];
console.log(dinosaurs[0]); // Outputs: "T-Rex"
console.log(dinosaurs[1]); // Outputs: "
   Velociraptor"
console.log(dinosaurs[2]); // Outputs: "
   Stegosaurus"
```

If you want to show just your favourite dinosaur, you can grab it by its index. For example, `dinosaurs[0]` gives you "T-Rex".

Setting or Changing Elements in an Array

You can change an element in an array by using its index and assigning a new value. Let's say you decide "T-Rex" should be "Tyrannosaurus Rex" instead:

```
dinosaurs[0] = "Tyrannosaurus Rex";
console.log(dinosaurs); // Outputs: ["
   Tyrannosaurus Rex", "Velociraptor", "
   Stegosaurus"]
```

You can also add elements to specific indexes. For example, to add a new dinosaur at index 3:

```
dinosaurs[3] = "Triceratops";
console.log(dinosaurs); // Outputs: ["
   Tyrannosaurus Rex", "Velociraptor", "
   Stegosaurus", "Triceratops"]
```

If you add an element at a high index, like 33, JavaScript will fill the gaps with `undefined`:

```
dinosaurs[33] = "Philosoraptor";
console.log(dinosaurs); // Outputs: ["
   Tyrannosaurus Rex", "Velociraptor", "
   Stegosaurus", "Triceratops", ...undefined x
   29..., "Philosoraptor"]
```

The `undefined` elements are placeholders, meaning those spots exist but don't have values yet.

Mixing Data Types in an Array

Arrays aren't picky—they can hold different types of data in the same list. For example:

```
var dinosaursAndNumbers = [3, "dinosaurs", ["
   triceratops", "stegosaurus", 3627.5], 40];
```

This array has:

- A number (`3`)
- A string (`"dinosaurs"`)
- Another array (`["triceratops", "stegosaurus", 3627.5]`)
- Another number (`40`)

To access elements in the inner array, you use two sets of square brackets. For example:

```
console.log(dinosaursAndNumbers[2]); // Outputs:
   ["triceratops", "stegosaurus", 3627.5]
console.log(dinosaursAndNumbers[2][0]); //
   Outputs: "triceratops"
```

Here, `dinosaursAndNumbers[2]` gets the inner array, and `[0]` gets the first element of that inner array ("triceratops"). It's like diving into a box inside a box!

Working with Arrays: Properties and Methods

Arrays come with built-in tools called **properties** and **methods**. Properties tell you something about the array (like its size), and methods let you do things to the array (like adding or removing elements).

Finding the Length of an Array

The `length` property tells you how many elements are in an array. Just add `.length` to the array's name:

```
var maniacs = ["Yakko", "Wakko", "Dot"];
console.log(maniacs.length); // Outputs: 3
```

The `length` is super useful because it helps you figure out the last index of the array. Since indexes start at 0, the last index is always `length - 1`. So, to get the last element:

```
console.log(maniacs[maniacs.length - 1]); //
   Outputs: "Dot"
```

This works no matter how long the array is!

Adding Elements to an Array

You can add elements to an array using two methods: `push()` and `unshift()`.

Adding to the End with `push()`

The `push()` method adds an element to the **end** of an array. For example:

```
var animals = [];
animals.push("cat");
animals.push("dog");
animals.push("llama");
console.log(animals); // Outputs: ["cat", "dog",
   "llama"]
console.log(animals.length); // Outputs: 3
```

When you call `push()`, it does two things:

1. Adds the element to the end of the array.
2. Returns the new length of the array.

Adding to the Beginning with `unshift()`

The `unshift()` method adds an element to the **beginning** of an array:

```
animals.unshift("monkey");
console.log(animals); // Outputs: ["monkey", "cat
    ", "dog", "llama"]
animals.unshift("polar bear");
console.log(animals); // Outputs: ["polar bear",
    "monkey", "cat", "dog", "llama"]
```

Like `push()`, `unshift()` adds the element and returns the new length of the array.

Removing Elements from an Array

You can remove elements using `pop()` and `shift()`.

Removing from the End with `pop()`

The `pop()` method removes the **last** element from an array and returns it:

```
var animals = ["polar bear", "monkey", "cat", "
    dog", "llama"];
var lastAnimal = animals.pop();
console.log(lastAnimal); // Outputs: "llama"
console.log(animals); // Outputs: ["polar bear",
    "monkey", "cat", "dog"]
```

Here, `pop()` removes "llama" and stores it in `lastAnimal`. The array now has one less element.

Removing from the Beginning with `shift()`

The `shift()` method removes the **first** element from an array and returns it:

```
var firstAnimal = animals.shift();
console.log(firstAnimal); // Outputs: "polar bear
    "
console.log(animals); // Outputs: ["monkey", "cat
    ", "dog"]
```

Now, "polar bear" is removed, and the array shifts everything else down.

Combining Push, Pop, Shift, and Unshift

You can mix these methods to manage your array like a pro. For example:

```
var animals = ["polar bear", "monkey", "cat", "
   dog", "llama"];
var lastAnimal = animals.pop(); // Removes "llama
   ", saves it
console.log(animals); // Outputs: ["polar bear",
   "monkey", "cat", "dog"]
animals.pop(); // Removes "dog" (not saved)
console.log(animals); // Outputs: ["polar bear",
   "monkey", "cat"]
animals.unshift(lastAnimal); // Adds "llama" back
    to the start
console.log(animals); // Outputs: ["llama", "
   polar bear", "monkey", "cat"]
```

This shows how you can save removed elements (like "llama") and add them back later. `pop()` and `shift()` are great for working with the ends of an array, while `push()` and `unshift()` let you add elements where you need them.

Remember

Arrays are like a toolbox for organising data in JavaScript. They let you store lists of items—strings, numbers, or even other arrays—in one variable. You can access elements using indexes (starting at 0), change elements, or add new ones with methods like `push()` and `unshift()`. You can also remove elements with `pop()` and `shift()`. The `length` property helps you keep track of how many items are in your array.

In sumamry:

- **What arrays are**: Lists of data stored in a single variable.
- **Why they're useful**: They make it easy to manage lots of data without creating tons of variables.
- **Creating arrays**: Use square brackets `[]` and separate elements with commas.
- **Accessing elements**: Use indexes like `array[0]` to get specific items.
- **Changing elements**: Use indexes to update or add elements.
- **Mixing data types**: Arrays can hold strings, numbers, and even other arrays.
- **Array properties**: The `length` property tells you how many elements are in the array.
- **Array methods**:
  - `push()`: Add to the end.
  - `unshift()`: Add to the beginning.
  - `pop()`: Remove from the end.

- `shift()`: Remove from the beginning.

With arrays, you can write cleaner, more efficient code. Try creating your own arrays and playing with these methods in your JavaScript projects. Maybe make a list of your favourite games or movies and mess around with adding or removing items.