3b Theory JS API pages 50 - 61 Arrays, Adding & Subtracting

Nicholas Bruzzese

July 24, 2025

1 Let's Learn About Arrays

Today, we're diving into **arrays** in JavaScript, focusing on how to add and subtract elements from them, and exploring some cool tricks to make your coding life easier. Arrays are like a backpack where you can store a bunch of items—numbers, strings, or even other arrays—and access them whenever you need. This guide will keep it simple, fun, and clear.

2 What Are Arrays?

An **array** is a special type of variable in JavaScript that can hold multiple values in a single place. Think of it like a list you might write on a piece of paper: "buy milk, bread, and Vegemite." In JavaScript, you can store that list in an array like this:

```
var shoppingList = ["milk", "bread", "Vegemite"];

```

Each item in the array has an **index**, which is like its position in the list. Indexes start at **0**, not 1. So, in the example above:

- `shoppingList[0]` is `"milk"`.
- `shoppingList[1]` is `"bread"`.
- `shoppingList[2]` is `"Vegemite"`.

Arrays are super handy because they let you group related data together and work with it easily.

3 Adding Arrays Together with `concat`

Sometimes, you want to combine two or more arrays into one big array. This is where the **concat** method comes in. It's like taking two shopping lists and sticking them together to make one mega-list.

3.1 How `concat` Works

The `concat` method takes one array and adds another array (or more) to the end of it, creating a new array. The original arrays don't change. Here's an example:

```
var furryAnimals = ["Alpaca", "Ring-tailed Lemur", "Yeti"];
var scalyAnimals = ["Boa Constrictor", "Godzilla"];
var furryAndScalyAnimals = furryAnimals.concat(scalyAnimals);
console.log(furryAndScalyAnimals);
// Output: ["Alpaca", "Ring-tailed Lemur", "Yeti", "Boa
Constrictor", "Godzilla"]

```

Here, `furryAnimals.concat(scalyAnimals)` creates a new array with all the furry animals followed by the scaly ones. The original `furryAnimals` and `scalyAnimals` arrays stay the same.

3.2 Joining Multiple Arrays

You can also use `concat` to combine more than two arrays. Just list them inside the parentheses, separated by commas:

```
var featheredAnimals = ["Macaw", "Dodo"];
var allAnimals = furryAnimals.concat(scalyAnimals,
featheredAnimals);
console.log(allAnimals);
// Output: ["Alpaca", "Ring-tailed Lemur", "Yeti", "Boa
Constrictor", "Godzilla", "Macaw", "Dodo"]

```

This is great for when you have multiple lists—like your favourite books and your mate's favourite books—and you want to combine them into one list to check at the bookstore.

4 Finding an Element's Index with `indexOf`

What if you want to know **where** a specific item is in your array? The **indexOf** method helps you find the index (position) of an element. It's like looking at your shopping list and asking, "Where's Vegemite on this list?"

4.1 How `indexOf` Works

You give `indexOf` an element, and it tells you its index in the array. For example:

```
var colors = ["red", "green", "blue"];
console.log(colors.indexOf("blue")); // Output: 2
console.log(colors.indexOf("green")); // Output: 1

```

- `"blue"` is at index 2 because we count from 0 (`red` is 0, `green` is 1, `blue` is 2).
- If the element isn't in the array, `indexOf` returns `-1`:

```
console.log(colors.indexOf("purple")); // Output: -1

```

This means `"purple"` isn't in the array.

4.2 Multiple Copies of an Element

If an element appears more than once, `indexOf` only gives you the **first** index where it shows up:

```
var insects = ["Bee", "Ant", "Bee", "Bee", "Ant"];
console.log(insects.indexOf("Bee")); // Output: 0

```

Here, `"Bee"` appears multiple times, but `indexOf` returns `0` because that's the first time it appears.

5 Turning an Array into a String with `join`

Sometimes, you want to take an array and turn it into a single string—like combining all your middle names into a full name. The **join** method does this by gluing all the array's elements together into one string.

5.1 Basic `join`

By default, `join` separates elements with commas:

```
var boringAnimals = ["Monkey", "Cat", "Fish", "Lizard"];
console.log(boringAnimals.join()); // Output: "Monkey,Cat,Fish
,Lizard"

```

5.2 Custom Separators

You can choose your own separator by putting it in the parentheses. The separator is a string, so use quotes:

```
console.log(boringAnimals.join(" - ")); // Output: "Monkey -
Cat - Fish - Lizard"
console.log(boringAnimals.join("*")); // Output: "Monkey*Cat*
Fish*Lizard"
console.log(boringAnimals.join(" ")); // Output: "Monkey Cat
Fish Lizard"

```

This is super useful for things like joining names:

```
var myNames = ["Nicholas", "Andrew", "Maxwell", "Morgan"];
console.log(myNames.join(" ")); // Output: "Nicholas Andrew
Maxwell Morgan"

```

Without `join`, you'd have to write something messy like:

```
{lstlisting}[style=javascript]
myNames[0] + " " + myNames[1] + " " + myNames[2] + " " + myNames
 [3];
// Output: "Nicholas Andrew Maxwell Morgan"
```

The `join` method is way easier, especially if your array length changes!

5.3 Non-String Elements

If your array has numbers or other non-string elements, `join` converts them to strings before combining them:

```
var ages = [11, 14, 79];
console.log(ages.join(" ")); // Output: "11 14 79"
```

6 Adding and Removing Elements with `push` and `pop`

Now, let's talk about adding and subtracting elements from an array. Imagine you're walking to a friend's house and noting landmarks. You can add landmarks to your array with **push** and remove them with **pop**.

6.1 Adding with `push`

The `push` method adds an element to the **end** of an array:

```
var landmarks = [];
landmarks.push("My house");
landmarks.push("Front path");
landmarks.push("Flickering streetlamp");
console.log(landmarks);
// Output: ["My house", "Front path", "Flickering streetlamp"]
```

Each `push` adds a new item to the end of the array.

6.2 Removing with `pop`

The `pop` method removes the **last** element from the array and returns it. When heading home, you can pop off landmarks to know where to go next:

```
console.log(landmarks.pop()); // Output: "Flickering streetlamp"
console.log(landmarks); // Output: ["My house", "Front path"]
console.log(landmarks.pop()); // Output: "Front path"
console.log(landmarks); // Output: ["My house"]
```

Notice how `pop` takes the last item off and gives it back to you. This is called a **stack**, like a stack of pancakes: the last pancake you put on the plate is the first one you eat. This is also known as **Last In, First Out (LIFO)**.

7 Generating Random Numbers with `Math.random` and `Math.floor`

Arrays are awesome for picking random items, like building a Magic 8-Ball or a random insult generator. To do this, we need random numbers, and JavaScript has two handy methods: **Math.random** and **Math.floor**.

7.1 `Math.random`

The `Math.random()` method gives you a random decimal between **0** and just under **1** (e.g., 0.984 or 0.369). To get a bigger range, multiply it:

```
console.log(Math.random() * 10); // Output: a number like 7.648 or 0.214
```

7.2 `Math.floor`

Array indexes need whole numbers, so we use `Math.floor` to round down a decimal to the nearest whole number:

```
console.log(Math.floor(3.746)); // Output: 3
console.log(Math.floor(9.999)); // Output: 9
```

Combine them to get a random index for an array:

```
var randomWords = ["Explosion", "Cave", "Princess", "Pen"];
var randomIndex = Math.floor(Math.random() * randomWords.length);
console.log(randomWords[randomIndex]); // Output: e.g., "Cave" or "Princess"
```

Here, `Math.random() * randomWords.length` gives a number between 0 and just under 4 (since the array has 4 elements). `Math.floor` rounds it down to 0, 1, 2, or 3, which we use as an index.

8 Building a Decision Maker

Let's put this together to make a **decision maker**, like a Magic 8-Ball. You create an array of responses and pick a random one:

```
var phrases = [
"That sounds good",
"Yes, you should definitely do that",
"I'm not sure that's a great idea",
"Maybe not today?",
"Computer says no"
];
console.log(phrases[Math.floor(Math.random() * phrases.length)]);
// Output: e.g., "Maybe not today?"
```

Each time you run this, you get a random response. You can use it to answer questions like, "Should I have another milkshake?" (Spoiler: the computer might say, "I'm not sure that's a great idea!")

9 Creating a Random Insult Generator

Now, let's build a **random insult generator**. You use multiple arrays and pick a random word from each to create a funny insult:

```
var randomBodyParts = ["Upperlip", "Nosehair", "Nail;
var randomAdjectives = ["Long", "Short", "Upsidedown"];
var randomWords = ["Fly", "Marmot", "Stick", "Monkey", "Rat"];

var randomBodyPart = randomBodyParts[Math.floor(Math.random() *
 randomBodyParts.length)];
var randomAdjective = randomAdjectives[Math.floor(Math.random()
 * randomAdjectives.length)];
var randomWord = randomWords[Math.floor(Math.random() *
 randomWords.length)];

var randomInsult = "Your " + randomBodyPart + " is like a " +
 randomAdjective + " " + randomWord + "!!!";
console.log(randomInsult);
// Output: e.g., "Your Nail is like a Upsidedown Marmot!!!"
```

Notice we use `array.length` to make the code flexible—if you add more words to an array, it still works without changing the math.

You can also write it using `join`:

```
var randomInsult = ["Your", randomBodyPart, "is", "like", "a",
 randomAdjective, randomWord, "!!!"].join(" ");
console.log(randomInsult);
// Output: e.g., "Your Hair is like a Smelly Fly!!!"
```

This joins the words with spaces, except for the last bit where we stick the exclamation marks right onto the word.

10 Programming Challenges

Here are some challenges to practice what you've learned:

1. **New Insults**: Create your own insult generator with your own words. Try making arrays with Aussie-themed words like "Your footy skills are like a dodgy dingo!!!"
2. **Sophisticated Insults**: Make insults like "Your [body part] is more [adjective] than a [animal]'s [animal body part]." You'll need another array for animal body parts, like `["tail", "snout", "claws"]`.
3. **Use + or join?**: Build two versions of your insult generator—one using `+` to combine strings, one using `join`. Think about which is easier to read or change later.
4. **Joining Numbers**: Turn the array `[3, 2, 1]` into the string `"3 is bigger than 2 is bigger than 1"` using `join`. Here's how:

```
var numbers = [3, "is bigger than", 2, "is bigger than", 1];
```

```
2  console.log(numbers.join(" ")); // Output: "3 is bigger than 2
    is bigger than 1"
```

11 Why Arrays Matter

Arrays are like your Swiss Army knife in JavaScript. They let you store lists of data, combine them, find items, turn them into strings, and even build fun programs like decision makers or insult generators. By using methods like `concat`, `indexOf`, `join`, `push`, and `pop`, you can manipulate arrays to do all sorts of tasks. The random number tricks with `Math.random` and `Math.floor` add a bit of magic, letting you create dynamic and unpredictable programs.

Next time you're coding, think about how arrays can help you organise your data—like keeping track of landmarks, generating random responses, or even managing your footy team's player list. Keep experimenting, and you'll find arrays are one of the most powerful tools in your coding toolkit!

12 Wrap-Up

That's it for today's lecture. Arrays are heaps of fun and super powerful. Want to try one of those challenges together? Let's make some code that's as fun as a barbie on a sunny arvo!