

Arrays

Ref Javascript

Making & changing arrays

An array holds a list of values in order, inside square brackets. Each item is an element, and the first index is 0:

```
const scores = [72, 88, 95];
console.log(scores[0]);      // 72
console.log(scores.length);  // 3
scores[1] = 90;              // change one element
console.log(scores);         // [72,90,95]
```

- `const` stops you giving `scores` a whole new array, but you can still change what is inside it.
- Reading past the end gives `undefined`: `scores[3]` is `undefined`.

Add and remove at the end with `push` and `pop`:

```
const queue = ["Ann", "Ben"];
queue.push("Cai");          // add to the end
console.log(queue);         // ["Ann","Ben","Cai"]
const last = queue.pop();   // remove from the end and return it
console.log(last);          // Cai
console.log(queue.length);  // 2
```

An array can hold arrays. That makes a grid, read with two indexes, row first:

```
const grid = [
  [1, 2, 3],
  [4, 5, 6],
];
console.log(grid[1][2]);    // 6 (row 1, column 2)
```

map, filter & reduce

These methods take a function and apply it to the array elements. They return a **new** array (or value) without themselves replacing the original items. The function you pass in can still change the original array or its objects:

- `map` makes a new array by changing every element.
- `filter` keeps only the elements for which the function returns `true`.
- `reduce` combines all the elements into one value, such as a total.

```
const prices = [4, 10, 25];
const withTax = prices.map((p) => p * 1.1);
console.log(withTax.map((p) => p.toFixed(2))); //
↪ ["4.40", "11.00", "27.50"]

const cheap = prices.filter((p) => p < 20);
console.log(cheap); // [4,10]

const total = prices.reduce((sum, p) => sum + p, 0);
console.log(total); // 39
```

In `reduce`, `sum` is the accumulator. It starts at the last argument (0) and takes the function's result after each element.

Searching & sorting

`[10, 9, 100, 1]`



`.sort()`

as text: "100" < "9" because "1" < "9"



`.sort((a, b) => a - b)`

as numbers: smallest first



Method	Gives back	Example with [5, 8, 2]	Result
<code>.includes(x)</code>	true or false	<code>.includes(8)</code>	true
<code>.indexOf(x)</code>	the first index of x, or -1	<code>.indexOf(2)</code>	2
<code>.find(f)</code>	the first element where f is true	<code>.find((n) => n > 6)</code>	8
<code>.findIndex(f)</code>	its index, or -1	<code>.findIndex((n) => n > 6)</code>	1

`sort` puts the elements in order. With no function, it sorts them **as text**, so numbers come out in the wrong order:

```
const nums = [10, 9, 100, 1];
console.log([...nums].sort());           // [1,10,100,9]    (as
↳ text!)
console.log([...nums].sort((a, b) => a - b)); // [1,9,10,100]
console.log([...nums].sort((a, b) => b - a)); // [100,10,9,1]
```

- The compare function returns a negative number when **a** should come first, and a positive number when **b** should.
- `sort` changes the array itself. `[...nums]` makes a copy first, so `nums` keeps its order.

Common mistakes

- The last element is `arr[arr.length - 1]`, not `arr[arr.length]`.
- `map` and `filter` do not change the array: store their result.
- `nums.sort()` on numbers sorts them as text. Pass `(a, b) => a - b`.