



DAY - 15 : Javascript Arrays [Part-1]



From Basics to Advanced

Arrays: Part -1 (CRUD)

Created by: Neeraj | [LinkedIn: neeraj-kumar1904](#) | [X: @_19_neeraj](#) | [GitHub: Neeraj05042001](#) |



Introduction

Arrays are one of the most fundamental and powerful data structures in JavaScript. They allow you to store multiple values in a single variable, which makes organizing and manipulating data much easier and more efficient.

In this lesson, we'll explore arrays in depth, from creating them to manipulating them, and learn about modern JavaScript features like destructuring, rest parameters, and the spread operator.



What is an Array in JavaScript?

An array is an ordered collection of values that can be accessed by their position (index). In JavaScript:

- Arrays can store any type of data (numbers, strings, objects, other arrays, etc.)
- Array indices start at 0 (zero-based indexing)
- Arrays are dynamic - they can grow or shrink as needed
- Arrays are objects with special behaviors and methods

```
// Array examples
const numbers = [1, 2, 3, 4, 5];           // 📋 Array of numbers
const fruits = ['apple', 'banana', 'orange']; // 🍎 Array of strings
const mixedArray = [100, true, 'taScript', {}]; // 🗨 Mixed types
const nestedArray = [1, 2, [3, 4, [5, 6]]]; // 📦 Nested arrays
```

💡 **Key Point:** JavaScript arrays are special objects with numeric keys and a `length` property, which makes them different from regular objects.



Knowledge Check

- ★ **Question 1:** What is the starting index of a JavaScript array?
- ★ **Question 2:** Can JavaScript arrays store different data types in the same array?

How to Create an Array in JavaScript?

There are several ways to create arrays in JavaScript:

1 Array Literal (Most Common)

```
const salad = ["🍎", "🍄", "🥬", "🥒", "🌽", "🥕", "🥑"];
```

2 Using the `Array()` Constructor

```
const anotherSalad = new Array("🍎", "🍄", "🥬", "🥒", "🌽", "🥕", "🥑");
```

⚠️ **Warning:** Be careful with the `new Array()` constructor! If you pass a single number, it creates an array with that many empty slots:

```
const five = new Array(5);    // Creates [<5 empty slots>]
const two = new Array(1, 2);  // Creates [1, 2]
```

3 `Array.of()` Method

```
const numbers = Array.of(1, 2, 3, 4, 5); // Creates [1, 2, 3, 4, 5]
const singleNumber = Array.of(5);        // Creates [5] (not an array of 5 empty slots)
```


4 `Array.from()` Method

Converts array-like or iterable objects into arrays:

```
const arrayFromString = Array.from("hello"); // Creates ["h", "e", "l", "l", "o"]
```

✅ Knowledge Check

 **Question 1:** What's the difference between `new Array(5)` and `Array.of(5)`?

 **Question 2:** Write code to create an array with the values 10, 20, and 30 using both array literal and constructor syntax.

How to Get/Access Elements from an Array in JS?

You can access array elements using square bracket notation with the index:

```
const salad = ["🍎", "🍄", "🥬", "🥒", "🌽", "🥕", "🥑"];

console.log(salad[0]);    // "🍎" (first element)
console.log(salad[2]);    // "🥬" (third element)
console.log(salad[5]);    // "🥕" (sixth element)
```

You can also loop through arrays to access each element:

```
const salad = ["🍎", "🍄", "🥬", "🥒", "🌽", "🥕", "🥑"];

// Using for loop
for (let i = 0; i < salad.length; i++) {
  console.log(`Element at index ${i} is ${salad[i]}`);
}

// Using for...of loop (ES6)
for (const vegetable of salad) {
  console.log(vegetable);
}

// Using forEach method
salad.forEach((vegetable, index) => {
  console.log(`Element at index ${index} is ${vegetable}`);
});
```

💡 **Key Point:** If you try to access an index that doesn't exist, JavaScript will return `undefined` rather than throwing an error.

✅ Knowledge Check

🧩 **Question 1:** What is returned if you try to access an array element at an index that doesn't exist?

🧩 **Question 2:** Write code to access the last element of an array without knowing its length in advance.

+ How to Add Elements to an Array in JS?

Adding Elements to the End: `push()`

```
const salad = ["🍎", "🍄", "🥬", "🥒", "🌽", "🥕", "🥑"];
```

```
// Using push() method
const newLength = salad.push("🥜"); // Adds "🥜" to the end
console.log(newLength); // 8 (new length of the array)
console.log(salad);      // ["🍎", "🍄", "🥬", "🥒", "🌽", "🥕", "🥑", "🥜"]
```

Adding Elements to the Beginning: `unshift()`

```
const salad = ["🍎", "🍄", "🥬", "🥒", "🌽", "🥕", "🥑"];

// Using unshift() method
const newLength = salad.unshift("🥜"); // Adds "🥜" to the beginning
console.log(newLength); // 8 (new length of the array)
console.log(salad);      // ["🥜", "🍎", "🍄", "🥬", "🥒", "🌽", "🥕", "🥑"]
```

Adding Elements at a Specific Position: `splice()`

```
const salad = ["🍎", "🍄", "🥬", "🥒", "🌽", "🥕", "🥑"];

// Using splice() method
// syntax: array.splice(startIndex, deleteCount, item1, item2, ...)
salad.splice(2, 0, "🥜"); // Insert "🥜" at index 2, delete 0 elements
console.log(salad);      // ["🍎", "🍄", "🥜", "🥬", "🥒", "🌽", "🥕", "🥑"]
```

💡 **Key Point:** `push()` and `unshift()` both return the new length of the array after the operation.

✅ Knowledge Check

🌱 **Question 1:** What methods can you use to add elements to an array?

🌱 **Question 2:** Which method would you use to insert an element in the middle of an array?

— How to Remove Elements from an Array in JS?

Removing Elements from the End: `pop()`

```
const salad = ["🍎", "🍄", "🥬", "🥒", "🌽", "🥕", "🥑"];

// Using pop() method
const removedItem = salad.pop(); // Removes the last element and returns it
console.log(removedItem);        // "🥑"
console.log(salad);              // ["🍎", "🍄", "🥬", "🥒", "🌽", "🥕"]
```

Removing Elements from the Beginning: `shift()`

```
const salad = ["🍎", "🍄", "🥬", "🥒", "🌽", "🥕", "🥑"];

// Using shift() method
const removedItem = salad.shift(); // Removes the first element and returns it
console.log(removedItem);          // "🍎"
console.log(salad);                 // ["🍄", "🥬", "🥒", "🌽", "🥕", "🥑"]
```

Removing Elements at a Specific Position: `splice()`

```
const salad = ["🍎", "🍄", "🥬", "🥒", "🌽", "🥕", "🥑"];

// Using splice() method
// syntax: array.splice(startIndex, deleteCount)
const removedItems = salad.splice(2, 2); // Remove 2 elements starting at index 2
console.log(removedItems);               // ["🥬", "🥒"]
console.log(salad);                      // ["🍎", "🍄", "🌽", "🥕", "🥑"]
```

💡 **Key Point:** `pop()` and `shift()` both return the removed element, while `splice()` returns an array of removed elements.

✅ Knowledge Check

- 🔄 **Question 1:** What is the difference between `pop()` and `shift()`?
- 🔄 **Question 2:** What do the methods `pop()` and `shift()` return when called on an array?

📋 How to Copy and Clone an Array in JS?

There are several ways to copy arrays in JavaScript:

1. Using `slice()` Method

```
const salad = ["🍎", "🍄", "🥬", "🥒", "🌽", "🥕", "🥑"];
const saladCopy = salad.slice(); // Creates a shallow copy of the array

console.log(salad === saladCopy); // false (different array references)
```

NOTE: Since `push()`, `unshift()`, `pop()` and `shift()` all these methods bring changes to the original array meaning that it **mutate** the source array but the `slice()` method does not mutate the source array meaning that it does not modify the original array rather creates a new copy of array. It deals in **immutable** way.

2. Using Spread Operator (ES6)

```
const salad = ["🍎", "🍄", "🥬", "🥒", "🌽", "🥕", "🥑"];
const saladCopy = [...salad]; // Creates a shallow copy using spread operator

console.log(saladCopy); // ["🍎", "🍄", "🥬", "🥒", "🌽", "🥕", "🥑"]
console.log(salad === saladCopy); // false (different array references)
```

3. Using Array.from() Method

```
const salad = ["🍎", "🍄", "🥬", "🥒", "🌽", "🥕", "🥑"];
const saladCopy = Array.from(salad); // Creates a shallow copy

console.log(saladCopy); // ["🍎", "🍄", "🥬", "🥒", "🌽", "🥕", "🥑"]
```

4. Using Array.concat() Method

```
const salad = ["🍎", "🍄", "🥬", "🥒", "🌽", "🥕", "🥑"];
const saladCopy = [].concat(salad); // Creates a shallow copy

console.log(saladCopy); // ["🍎", "🍄", "🥬", "🥒", "🌽", "🥕", "🥑"]
```

💡 **Key Point:** All the methods above create shallow copies, meaning that nested arrays or objects still share references. For deep copying complex arrays, you might need to use methods like `JSON.parse(JSON.stringify())` or a library like Lodash.

✅ Knowledge Check

🔍 **Question 1:** What's the difference between a shallow copy and a deep copy?

🔍 **Question 2:** Which method would you use to create a copy of an array that includes only the first three elements?

🌟 How to Determine if a Value is an Array in JS?

JavaScript provides the `Array.isArray()` method to check if a value is an array:

```
Array.isArray(["🍎", "🍄", "🥬", "🥒", "🌽", "🥕", "🥑"]); // true
Array.isArray("🍎"); // false (string)
Array.isArray({ tomato: "🍎" }); // false (object)
Array.isArray([]); // true (empty array)
```

```
const arr = [1, 2, 3, 4];
Array.isArray(arr);           // true
```

💡 **Key Point:** The `typeof` operator doesn't help much with arrays because it returns "object" for arrays. Always use `Array.isArray()` to check specifically for arrays.

✅ Knowledge Check

💡 **Question 1:** Why doesn't `typeof arr === 'array'` work to check if a value is an array?

💡 **Question 2:** What does `typeof []` return in JavaScript?

🧩 Array Destructuring in JavaScript

Array destructuring is a convenient way to extract values from arrays and assign them to variables.

📌 Basic Destructuring

```
const salad = ["🍅", "🍄", "🥬", "🥒", "🥦", "🥕", "🥑"];

// Without destructuring 🚫
const tomato = salad[0];
const mushroom = salad[1];
const carrot = salad[5];

// With destructuring ✅
const [tomato, mushroom, , , , carrot] = salad;
console.log(tomato, mushroom, carrot); // "🍅" "🍄" "🥕"
```

📌 How to Assign a Default Value to a Variable?

You can provide default values in case the array doesn't have a value at that position:

```
const [tomato, mushroom = "🍄"] = ["🍅"];
console.log(tomato); // "🍅"
console.log(mushroom); // "🍄" (default value used) ⭐
```

📌 How to Skip a Value in an Array?

You can use commas to skip elements in the array:

```
const [tomato, , carrot] = ["🍅", "🍄", "🥕"];
console.log(tomato); // "🍅"
```

```
console.log(carrot); // "🥕" (skipped "🍄")
```

🌟 Nested Array Destructuring in JS

You can destructure nested arrays too:

```
const fruits = ["🍏", "🍓", "🌽", "🍉", ["🍎", "🍄", "🥕"]];

// Without destructuring 🚫
const veg = fruits[4]; // ["🍎", "🍄", "🥕"]
const carrot = veg[2]; // "🥕"
const directCarrot = fruits[4][2]; // "🥕"

// With destructuring ✅
const [, , , , [, , carrot]] = ["🍏", "🍓", "🌽", "🍉", ["🍎", "🍄", "🥕"]];
console.log(carrot); // "🥕"
```

✅ Knowledge Check

🧩 Question 1: What happens if you try to destructure more variables than there are elements in the array?

🧩 Question 2: Write code to destructure the first and last elements of an array of unknown length.

💠 How to Use the Rest Parameter in JS?

The rest parameter (`...`) collects multiple elements into an array.


`...rest` --> at left side(variable naming side) becomes `rest` parameter.

```
const [tomato, mushroom, ...rest] = ["🍎", "🍄", "🥬", "🥒", "🌻", "🥕", "🥑"];
console.log(tomato); // "🍎"
console.log(mushroom); // "🍄"
console.log(rest); // ["🥬", "🥒", "🌻", "🥕", "🥑"]
```

💡 Key Point: The rest parameter must be the last parameter in the destructuring pattern.

✅ Knowledge Check

🧩 Question 1: What happens if you use the rest parameter in the middle of a destructuring pattern?

 **Question 2:** Can you use more than one rest parameter in a single destructuring?

How to Use the Spread Operator in JS?

The spread operator (`...`) expands an array into individual elements.

We can create a clone or copy of the array using `spread operator` .


`...arrayname` ---> at right side(value of array side) becomes `spread operator`

```
const mySalad = ["🍎", "🍄", "🥬", "🥒", "🌽", "🥕", "🥑"];


// Copying an array
const mySaladCopy = [...mySalad];
console.log(mySaladCopy); // ["🍎", "🍄", "🥬", "🥒", "🌽", "🥕", "🥑"]
console.log(mySalad === mySaladCopy); // false (different array references)


// Concatenating arrays
const moreSalad = [...mySalad, "🍌", "🍌"];
console.log(moreSalad); // ["🍎", "🍄", "🥬", "🥒", "🌽", "🥕", "🥑", "🍌", "🍌"]

// Passing array elements as function arguments
function makeSalad(a, b, c) {
  return `Salad with ${a}, ${b}, and ${c}`;
}
const ingredients = ["🍎", "🍄", "🥬"];
console.log(makeSalad(...ingredients)); // "Salad with 🍎, 🍄, and 🥬"
```

 **Key Point:** The spread operator is great for creating copies of arrays, combining arrays, and converting array-like objects to arrays.

Knowledge Check

 **Question 1:** What's the difference between the rest parameter and the spread operator?

 **Question 2:** How can you use the spread operator to create a new array with one item added at the beginning?

Destructuring Use Cases in JavaScript

How to Swap Values with Destructuring?

```
let first = "😞";
let second = "😄";

// Swap values using destructuring ✨
[first, second] = [second, first];


console.log(first); // "😄"
console.log(second); // "😞"
```


How to Merge Arrays?

```
const emotions = ["😄", "😞"];
const veggies = ["🥬", "🥒", "🌽", "🥕"];

// Merge arrays using spread operator ✨
const emotionalVeggies = [...emotions, ...veggies];
console.log(emotionalVeggies); // ["😄", "😞", "🥬", "🥒", "🌽", "🥕"]
```

✅ Knowledge Check

 **Question 1:** Without using a temporary variable or destructuring, how would you swap two values?

 **Question 2:** How would you merge three arrays into one using the spread operator?

The length property

The `length` property of an array returns the number of elements in the array:

```
const arr1 = [11, 21, 73];
console.log(arr1.length); // 3

const arr2 = new Array(7);
console.log(arr2.length); // 7 (an array with 7 empty slots)
```

You can also modify the length property to truncate or expand an array:

```
const arr = [1, 2, 3, 4, 5];
console.log(arr.length); // 5

// Truncate array
arr.length = 3;
console.log(arr); // [1, 2, 3]
```

```
// Expand array (adds empty slots)
arr.length = 6;
console.log(arr); // [1, 2, 3, empty × 3]
```

💡 **Key Point:** The maximum length of an array in JavaScript is $2^{32} - 1$ (4,294,967,295) elements.

✅ Knowledge Check

🔪 **Question 1:** What happens when you set the length property of an array to a value less than its current length?

🔪 **Question 2:** What happens when you set the length property of an array to a value greater than its current length?

🏋️ Practice Tasks

Test your understanding of JavaScript Arrays with these practice tasks:

1. **Create an array** of your five favorite foods.

2. **Access elements:**

- Get the first and last elements of your array.
- Try accessing an element at an index that doesn't exist. What happens?

3. **Modify your array:**

- Add a new food to the beginning of your array.
- Add a new food to the end of your array.
- Remove the first food from your array.
- Remove the last food from your array.

4. **Create a function** that takes your array and returns a new array with all elements in reverse order.

5. **Create a function** that merges two arrays and removes any duplicate elements.

6. **Array of objects:**

- Create an array of objects where each object represents a person with properties for name, age, and city.
- Write code to find all people who are older than 30.

- Sort the array by age in descending order.

7. Nested arrays:

- Create a 3x3 grid represented as a nested array.
- Write a function to get the element at a specific row and column.
- Write a function to calculate the sum of all elements in the grid.

8. Destructuring challenge:

- Given the array `const data = [1, [2, 3], 4, [5, [6, 7]]]`, use destructuring to get the values 1, 3, and 7 in separate variables in one line.

9. Spread operator challenge:




- Given two arrays `const arr1 = [1, 2, 3]` and `const arr2 = [4, 5, 6]`, create a new array that has the elements in the following order: [4, 5, 6, 1, 2, 3, 0].

10. Real-world application:

- Create an array of objects representing tasks in a to-do list.
- Write functions to add a task, mark a task as completed, and delete a task.
- Write a function to filter tasks by their completion status.
- Use array methods to sort tasks by due date.



For Part 2 of the note see other post or visit the github repo for full notes & practise Question and it's solution

Created by: Neeraj | [LinkedIn: neeraj-kumar1904](#)  | [X: @_19_neeraj](#)  | [GitHub: Neeraj05042001](#)  |