# Array

JavaScript arrays are a fundamental part of the language, used to store multiple values in a single variable. Here's a detailed overview of JavaScript arrays:

### 1. Declaration and Initialization

You can create an array in JavaScript using various methods:

#### Using Array Literal

let array = [];  
let array = [1, 2, 3, "four", true];

#### Using the Array Constructor

let array = new Array();  
let array = new Array(1, 2, 3, "four", true);

### 2. Accessing Array Elements

Array elements can be accessed using their index, which starts at 0.

let array = [1, 2, 3];  
console.log(array[0]); // Outputs: 1  
console.log(array[2]); // Outputs: 3

### 3. Modifying Array Elements

You can change an element in an array by accessing its index and assigning a new value.

let array = [1, 2, 3];  
array[0] = 10;  
console.log(array); // Outputs: [10, 2, 3]

### 4. Array Properties

#### length

The length property of an array gives the number of elements in the array.

let array = [1, 2, 3];  
console.log(array.length); // Outputs: 3

### 5. Common Array Methods

#### push() and pop()

* push() adds one or more elements to the end of an array.
* pop() removes the last element from an array.

let array = [1, 2, 3];  
array.push(4); // [1, 2, 3, 4]  
array.pop(); // [1, 2, 3]

#### shift() and unshift()

* shift() removes the first element from an array.
* unshift() adds one or more elements to the beginning of an array.

let array = [1, 2, 3];  
array.shift(); // [2, 3]  
array.unshift(0); // [0, 2, 3]

#### concat()

The concat() method merges two or more arrays.

let array1 = [1, 2];  
let array2 = [3, 4];  
let array3 = array1.concat(array2); // [1, 2, 3, 4]

#### slice()

The slice() method returns a shallow copy of a portion of an array into a new array object.

let array = [1, 2, 3, 4];  
let newArray = array.slice(1, 3); // [2, 3]

#### splice()

The splice() method changes the contents of an array by removing or replacing existing elements and/or adding new elements in place.

let array = [1, 2, 3, 4];  
array.splice(1, 2, "a", "b"); // [1, 'a', 'b', 4]

#### forEach()

The forEach() method executes a provided function once for each array element.

let array = [1, 2, 3];  
array.forEach(function (element) {  
 console.log(element); // Outputs: 1, 2, 3  
});

#### map()

The map() method creates a new array populated with the results of calling a provided function on every element in the calling array.

let array = [1, 2, 3];  
let newArray = array.map(function (element) {  
 return element \* 2;  
}); // [2, 4, 6]

#### filter()

The filter() method creates a new array with all elements that pass the test implemented by the provided function.

let array = [1, 2, 3, 4];  
let newArray = array.filter(function (element) {  
 return element > 2;  
}); // [3, 4]

#### reduce()

The reduce() method executes a reducer function (that you provide) on each element of the array, resulting in a single output value.

let array = [1, 2, 3, 4];  
let sum = array.reduce(function (accumulator, currentValue) {  
 return accumulator + currentValue;  
}, 0); // 10

### 6. Iterating Over Arrays

You can iterate over arrays using various methods like for loops, for...of loops, and the forEach() method.

let array = [1, 2, 3];  
  
// Using for loop  
for (let i = 0; i < array.length; i++) {  
 console.log(array[i]);  
}  
  
// Using for...of loop  
for (let element of array) {  
 console.log(element);  
}  
  
// Using forEach method  
array.forEach(function (element) {  
 console.log(element);  
});

### 7. Multidimensional Arrays

JavaScript allows you to create arrays within arrays, known as multidimensional arrays.

let array = [  
 [1, 2, 3],  
 [4, 5, 6],  
 [7, 8, 9],  
];  
console.log(array[0][1]); // Outputs: 2

JavaScript arrays are versatile and powerful, allowing for complex data manipulation and storage. Understanding these basics will help you utilize arrays effectively in your JavaScript programming.

# JavaScript Array Methods

## 1. pop()

In JavaScript, the pop() method removes the last element from an array and returns that element. This method changes the length of the array.

### Syntax

let poppedElement = array.pop();

### Example 1: Basic Usage

const numbers = [1, 2, 3];  
const poppedElement = numbers.pop();  
console.log(poppedElement); // 3  
console.log(numbers); // [1, 2]

### Key Points

* **Element Removal**: The pop() method removes the last element from the array.
* **Length Adjustment**: The pop() method changes the length of the array.

### Use Cases

* Removing the last element of a stack.
* Modifying arrays where the last item needs to be processed and removed.

## 2. push()

In JavaScript, the push() method adds one or more elements to the end of an array and returns the new length of the array.

### Syntax

let newLength = array.push(element1, element2, ...);

* **element1, element2, ...**: Elements to add to the end of the array.

### Example 1: Basic Usage

const numbers = [1, 2];  
const newLength = numbers.push(3);  
console.log(newLength); // 3  
console.log(numbers); // [1, 2, 3]

### Key Points

* **Length Return**: The push() method returns the new length of the array after adding elements.
* **Array Modification**: The push() method modifies the original array.

### Use Cases

* Adding new elements to a stack.
* Building arrays incrementally by adding new items.

## 3. reverse()

In JavaScript, the reverse() method reverses the order of the elements in an array. The first array element becomes the last, and the last array element becomes the first.

### Syntax

array.reverse();

### Example 1: Basic Usage

const numbers = [1, 2, 3];  
numbers.reverse();  
console.log(numbers); // [3, 2, 1]

### Key Points

* **In-place Reversal**: The reverse() method reverses the array in place and returns the reversed array.
* **Array Modification**: The reverse() method modifies the original array.

### Use Cases

* Reversing the order of elements for display purposes.
* Implementing algorithms that require elements to be processed in reverse order.

## 4. shift()

In JavaScript, the shift() method removes the first element from an array and returns that removed element. This method changes the length of the array.

### Syntax

let shiftedElement = array.shift();

### Example 1: Basic Usage

const numbers = [1, 2, 3];  
const shiftedElement = numbers.shift();  
console.log(shiftedElement); // 1  
console.log(numbers); // [2, 3]

### Key Points

* **Element Removal**: The shift() method removes the first element from the array.
* **Length Adjustment**: The shift() method changes the length of the array.

### Use Cases

* Removing the first element in a queue.
* Modifying arrays where the first item needs to be processed and removed.

## 5. unshift()

In JavaScript, the unshift() method adds one or more elements to the beginning of an array and returns the new length of the array.

### Syntax

let newLength = array.unshift(element1, element2, ...);

* **element1, element2, ...**: Elements to add to the front of the array.

### Example 1: Basic Usage

const numbers = [2, 3];  
const newLength = numbers.unshift(1);  
console.log(newLength); // 3  
console.log(numbers); // [1, 2, 3]

### Key Points

* **Length Return**: The unshift() method returns the new length of the array after adding elements.
* **Array Modification**: The unshift() method modifies the original array.

### Use Cases

* Adding elements to the beginning of a queue.
* Building arrays by prepending elements.

## 6. sort()

In JavaScript, the sort() method sorts the elements of an array in place and returns the sorted array. The default sort order is according to string Unicode code points.

### Syntax

array.sort(compareFunction);

* **compareFunction** (Optional): Function that defines the sort order.

### Example 1: Basic Usage

const fruits = ["banana", "apple", "cherry"];  
fruits.sort();  
console.log(fruits); // ["apple", "banana", "cherry"]

### Example 2: Custom Sort Order

const numbers = [4, 2, 5, 1, 3];  
numbers.sort((a, b) => a - b);  
console.log(numbers); // [1, 2, 3, 4, 5]

### Key Points

* **In-place Sorting**: The sort() method sorts the array in place and returns the sorted array.
* **Default Sort Order**: The default sort order is according to string Unicode code points.

### Use Cases

* Sorting arrays of strings or numbers.
* Custom sorting with a compare function.

## 7. splice()

In JavaScript, the splice() method changes the contents of an array by removing or replacing existing elements and/or adding new elements.

### Syntax

array.splice(start, deleteCount, item1, item2, ...);

* **start**: The index at which to start changing the array.
* **deleteCount** (Optional): The number of elements to remove.
* **item1, item2, ...** (Optional): Elements to add to the array.

### Example 1: Basic Usage

const numbers = [1, 2, 3, 4, 5];  
numbers.splice(2, 1);  
console.log(numbers); // [1, 2, 4, 5]

### Example 2: Adding Elements

const numbers = [1, 2, 3];  
numbers.splice(1, 0, "a", "b");  
console.log(numbers); // [1, "a", "b", 2, 3]

### Key Points

* **Array Modification**: The splice() method modifies the original array by removing, replacing, or adding elements.
* **Multiple Operations**: The splice() method can perform multiple operations in a single call.

### Use Cases

* Removing specific elements from an array.
* Inserting elements at a specific index.

## 8. copyWithin()

In JavaScript, the copyWithin() method shallow copies part of an array to another location in the same array and returns it, without modifying its length.

### Syntax

array.copyWithin(target, start, end);

* **target**: The index at which to copy the sequence to.
* **start** (Optional): The index at which to start copying elements from.
* **end** (Optional): The index at which to end copying elements from.

### Example 1: Basic Usage

const array = [1, 2, 3, 4, 5];  
console.log(array.copyWithin(0, 3)); // [4, 5, 3, 4, 5]

### Key Points

* **In-place Modification**: The copyWithin() method modifies the array in place.
* **No Length Change**: The copyWithin() method does not change the length of the array.

### Use Cases

* Shifting elements within an array.
* Duplicating elements at different positions.

## 9. fill()

In JavaScript, the fill() method changes all elements in an array to a static value, from a start index to an end index. It returns the modified array.

### Syntax

array.fill(value, start, end);

* **value**: Value to fill the array with.
* **start** (Optional): The index at which to start filling.
* **end** (Optional): The index at which to end filling.

### Example 1: Basic Usage

const array = [1, 2, 3, 4];  
console.log(array.fill(0, 2, 4)); // [1, 2, 0, 0]

### Key Points

* **In-place Modification**: The fill() method modifies the array in place.
* **Static Value**: The fill() method sets all elements to the specified value.

### Use Cases

* Initializing arrays with a specific value.
* Resetting elements in an array to a default value.

## 10. concat()

In JavaScript, the concat() method is used to merge two or more arrays. This method does not change the existing arrays but instead returns a new array.

### Syntax

let newArray = array1.concat(array2, array3, ...);

### Example 1: Basic Usage

const array1 = [1, 2];  
const array2 = [3, 4];  
const newArray = array1.concat(array2);  
console.log(newArray); // [1, 2, 3, 4]

### Key Points

* **Non-destructive**: The concat() method does not modify the original arrays.
* **New Array**: The concat() method returns a new array.

### Use Cases

* Merging multiple arrays into one.
* Combining arrays without modifying the originals.

## 11. includes()

In JavaScript, the includes() method determines whether an array includes a certain value among its entries, returning true or false as appropriate.

### Syntax

array.includes(valueToFind, fromIndex);

* **valueToFind**: The value to search for.
* **fromIndex** (Optional): The position in the array at which to begin the search.

### Example 1: Basic Usage

const array = [1, 2, 3];  
console.log(array.includes(2)); // true  
console.log(array.includes(4)); // false

### Key Points

* **Boolean Return**: The includes() method returns true or false.
* **Search from Index**: You can specify the index at which to begin the search.

### Use Cases

* Checking if an array contains a specific element.
* Implementing conditional logic based on array contents.

## 12. indexOf()

In JavaScript, the indexOf() method returns the first index at which a given element can be found in the array, or -1 if it is not present.

### Syntax

array.indexOf(searchElement, fromIndex);

* **searchElement**: The element to locate in the array.
* **fromIndex** (Optional): The index to start the search at.

### Example 1: Basic Usage

const array = [1, 2, 3, 2];  
console.log(array.indexOf(2)); // 1  
console.log(array.indexOf(4)); // -1

### Key Points

* **First Occurrence**: The indexOf() method returns the index of the first occurrence of the specified element.
* **Not Found**: If the element is not found, -1 is returned.

### Use Cases

* Finding the position of an element in an array.
* Determining if an array contains a specific element.

## 13. join()

In JavaScript, the join() method creates and returns a new string by concatenating all of the elements in an array, separated by a specified separator string.

### Syntax

let str = array.join(separator);

* **separator** (Optional): A string to separate each pair of adjacent elements. Defaults to a comma (,) if omitted.

### Example 1: Basic Usage

const array = [1, 2, 3];  
console.log(array.join()); // "1,2,3"  
console.log(array.join("-")); // "1-2-3"

### Key Points

* **String Return**: The join() method returns a string.
* **Custom Separator**: You can specify a custom separator string.

### Use Cases

* Creating a CSV string from an array.
* Joining array elements into a single string.

## 14. lastIndexOf()

In JavaScript, the lastIndexOf() method returns the last index at which a given element can be found in the array, or -1 if it is not present. The array is searched backwards, starting at fromIndex.

### Syntax

array.lastIndexOf(searchElement, fromIndex);

* **searchElement**: The element to locate in the array.
* **fromIndex** (Optional): The index at which to start searching backwards.

### Example 1: Basic Usage

const array = [1, 2, 3, 2];  
console.log(array.lastIndexOf(2)); // 3  
console.log(array.lastIndexOf(4)); // -1

### Key Points

* **Last Occurrence**: The lastIndexOf() method returns the index of the last occurrence of the specified element.
* **Search Backwards**: The array is searched backwards.

### Use Cases

* Finding the position of the last occurrence of an element.
* Determining if an array contains a specific element.

## 15. slice()

In JavaScript, the slice() method returns a shallow copy of a portion of an array into a new array object selected from start to end (end not included).

### Syntax

let newArray = array.slice(start, end);

* **start** (Optional): The beginning index of the portion to extract.
* **end** (Optional): The end index of the portion to extract.

### Example 1: Basic Usage

const array = [1, 2, 3, 4];  
const newArray = array.slice(1, 3);  
console.log(newArray); // [2, 3]

### Key Points

* **New Array**: The slice() method returns a new array.
* **Portion Extraction**: The slice() method extracts a portion of the array.

### Use Cases

* Creating a subset of an array.
* Copying part of an array to a new array.

## 16. toSource()

In JavaScript, the toSource() method returns a string representing the source code of the array.

### Syntax

array.toSource();

### Example 1: Basic Usage

const array = [1, 2, 3];  
console.log(array.toSource()); // "[1, 2, 3]"

### Key Points

* **Source Code**: The toSource() method returns a string representing the array's source code.

### Use Cases

* Debugging and logging array structures.

## 17. toString()

In JavaScript, the toString() method returns a string representing the array and its elements.

### Syntax

let str = array.toString();

### Example 1: Basic Usage

const array = [1, 2, 3];  
console.log(array.toString()); // "1,2,3"

### Key Points

* **String Return**: The toString() method returns a string representing the array and its elements.

### Use Cases

* Converting an array to a string for display or logging.
* Creating a simple representation of an array's contents.

## 18. toLocaleString()

In JavaScript, the toLocaleString() method returns a string representing the elements of the array using locale-specific conventions.

### Syntax

let str = array.toLocaleString(locales, options);

* **locales** (Optional): A string with a BCP 47 language tag, or an array of such strings.
* **options** (Optional): An object with configuration properties for the formatting.

### Example 1: Basic Usage

const date = [new Date()];  
console.log(date.toLocaleString()); // "7/4/2024, 12:00:00 AM"

### Key Points

* **Locale-specific**: The toLocaleString() method formats elements using locale-specific conventions.

### Use Cases

* Displaying array elements in a locale-specific format.
* Formatting dates and numbers in an array for internationalization.

## 19. entries()

In JavaScript, the entries() method returns a new array iterator object that contains the key/value pairs for each index in the array.

### Syntax

let iterator = array.entries();

### Example 1: Basic Usage

const array = ["a", "b", "c"];  
const iterator = array.entries();  
  
for (const [index, element] of iterator) {  
 console.log(index, element);  
}  
// 0 "a"  
// 1 "b"  
// 2 "c"

### Key Points

* **Iterator Return**: The entries() method returns an iterator with key/value pairs.

### Use Cases

* Iterating over array elements with their indices.
* Implementing algorithms that require both index and value.

## 20. every()

In JavaScript, the every() method tests whether all elements in the array pass the test implemented by the provided function. It returns a Boolean value.

### Syntax

let result = array.every(callback(element, index, array), thisArg);

* **callback**: Function to test each element.
* **thisArg** (Optional): Value to use as this when executing callback.

### Example 1: Basic Usage

const array = [1, 2, 3, 4];  
const allAboveZero = array.every((num) => num > 0);  
console.log(allAboveZero); // true

### Key Points

* **Boolean Return**: The every() method returns true if all elements pass the test, otherwise false.
* **Callback Function**: The every() method uses a callback function to test elements.

### Use Cases

* Checking if all elements in an array meet a condition.
* Validating data in arrays.

## 21. filter()

In JavaScript, the filter() method creates a new array with all elements that pass the test implemented by the provided function.

### Syntax

let new  
  
Array = array.filter(callback(element, index, array), thisArg);

* **callback**: Function to test each element.
* **thisArg** (Optional): Value to use as this when executing callback.

### Example 1: Basic Usage

const array = [1, 2, 3, 4];  
const evenNumbers = array.filter((num) => num % 2 === 0);  
console.log(evenNumbers); // [2, 4]

### Key Points

* **New Array**: The filter() method returns a new array with elements that pass the test.
* **Callback Function**: The filter() method uses a callback function to test elements.

### Use Cases

* Creating a subset of an array based on a condition.
* Removing unwanted elements from an array.

## 22. find()

In JavaScript, the find() method returns the value of the first element in the array that satisfies the provided testing function. Otherwise, undefined is returned.

### Syntax

let element = array.find(callback(element, index, array), thisArg);

* **callback**: Function to test each element.
* **thisArg** (Optional): Value to use as this when executing callback.

### Example 1: Basic Usage

const array = [1, 2, 3, 4];  
const foundElement = array.find((num) => num > 2);  
console.log(foundElement); // 3

### Key Points

* **Single Element**: The find() method returns the first element that satisfies the condition.
* **Callback Function**: The find() method uses a callback function to test elements.

### Use Cases

* Finding the first element that meets a specific condition.
* Searching for elements in arrays.

## 23. findIndex()

In JavaScript, the findIndex() method returns the index of the first element in the array that satisfies the provided testing function. Otherwise, -1 is returned.

### Syntax

let index = array.findIndex(callback(element, index, array), thisArg);

* **callback**: Function to test each element.
* **thisArg** (Optional): Value to use as this when executing callback.

### Example 1: Basic Usage

const array = [1, 2, 3, 4];  
const foundIndex = array.findIndex((num) => num > 2);  
console.log(foundIndex); // 2

### Key Points

* **Single Index**: The findIndex() method returns the index of the first element that satisfies the condition.
* **Callback Function**: The findIndex() method uses a callback function to test elements.

### Use Cases

* Finding the index of the first element that meets a specific condition.
* Searching for element positions in arrays.

## 24. forEach()

In JavaScript, the forEach() method executes a provided function once for each array element.

### Syntax

array.forEach(callback(element, index, array), thisArg);

* **callback**: Function to execute on each element.
* **thisArg** (Optional): Value to use as this when executing callback.

### Example 1: Basic Usage

const array = [1, 2, 3];  
array.forEach((num) => console.log(num));  
// 1  
// 2  
// 3

### Key Points

* **No Return**: The forEach() method does not return a value.
* **Callback Function**: The forEach() method uses a callback function to execute on each element.

### Use Cases

* Iterating over array elements for side effects.
* Applying a function to each element in an array.

## 25. keys()

In JavaScript, the keys() method returns a new array iterator object that contains the keys for each index in the array.

### Syntax

let iterator = array.keys();

### Example 1: Basic Usage

const array = ["a", "b", "c"];  
const iterator = array.keys();  
  
for (const key of iterator) {  
 console.log(key);  
}  
// 0  
// 1  
// 2

### Key Points

* **Iterator Return**: The keys() method returns an iterator with the keys of the array.

### Use Cases

* Iterating over array indices.
* Implementing algorithms that require index values.

## 26. map()

In JavaScript, the map() method creates a new array populated with the results of calling a provided function on every element in the calling array.

### Syntax

let newArray = array.map(callback(element, index, array), thisArg);

* **callback**: Function to execute on each element.
* **thisArg** (Optional): Value to use as this when executing callback.

### Example 1: Basic Usage

const array = [1, 2, 3];  
const doubledArray = array.map((num) => num \* 2);  
console.log(doubledArray); // [2, 4, 6]

### Key Points

* **New Array**: The map() method returns a new array with the results of the callback function.
* **Callback Function**: The map() method uses a callback function to process elements.

### Use Cases

* Transforming elements in an array.
* Applying a function to each element and creating a new array with the results.

## 27. reduce()

In JavaScript, the reduce() method executes a reducer function (that you provide) on each element of the array, resulting in a single output value.

### Syntax

let result = array.reduce(  
 callback(accumulator, currentValue, index, array),  
 initialValue  
);

* **callback**: Function to execute on each element.
* **initialValue** (Optional): Value to use as the first argument to the first call of the callback.

### Example 1: Basic Usage

const array = [1, 2, 3, 4];  
const sum = array.reduce((acc, num) => acc + num, 0);  
console.log(sum); // 10

### Key Points

* **Single Value**: The reduce() method returns a single value.
* **Callback Function**: The reduce() method uses a callback function to process elements.

### Use Cases

* Summing elements in an array.
* Aggregating or accumulating values from an array.

## 28. reduceRight()

In JavaScript, the reduceRight() method applies a function against an accumulator and each value of the array (from right to left) to reduce it to a single value.

### Syntax

let result = array.reduceRight(  
 callback(accumulator, currentValue, index, array),  
 initialValue  
);

* **callback**: Function to execute on each element.
* **initialValue** (Optional): Value to use as the first argument to the first call of the callback.

### Example 1: Basic Usage

const array = [1, 2, 3, 4];  
const sum = array.reduceRight((acc, num) => acc + num, 0);  
console.log(sum); // 10

### Key Points

* **Single Value**: The reduceRight() method returns a single value.
* **Callback Function**: The reduceRight() method uses a callback function to process elements.

### Use Cases

* Summing elements in an array from right to left.
* Aggregating or accumulating values in reverse order.

## 29. some()

In JavaScript, the some() method tests whether at least one element in the array passes the test implemented by the provided function. It returns a Boolean value.

### Syntax

let result = array.some(callback(element, index, array), thisArg);

* **callback**: Function to test each element.
* **thisArg** (Optional): Value to use as this when executing callback.

### Example 1: Basic Usage

const array = [1, 2, 3, 4];  
const hasEven = array.some((num) => num % 2 === 0);  
console.log(hasEven); // true

### Key Points

* **Boolean Return**: The some() method returns true if at least one element passes the test, otherwise false.
* **Callback Function**: The some() method uses a callback function to test elements.

### Use Cases

* Checking if any elements in an array meet a condition.
* Implementing conditional logic based on array contents.

## 30. values()

In JavaScript, the values() method returns a new array iterator object that contains the values for each index in the array.

### Syntax

let iterator = array.values();

### Example 1: Basic Usage

const array = ["a", "b", "c"];  
const iterator = array.values();  
  
for (const value of iterator) {  
 console.log(value);  
}  
// "a"  
// "b"  
// "c"

### Key Points

* **Iterator Return**: The values() method returns an iterator with the values of the array.

### Use Cases

* Iterating over array elements.
* Implementing algorithms that require array values.