JavaScript Arrays

JavaScript arrays are like ordered lists that can hold different types of data, such as numbers, strings, or even other arrays. Each item in an array has a special number called an "index" that tells you its position, starting from 0 for the first item.

Here's a simple way to think about it:

Imagine a shelf where you keep your favorite books.

- **The shelf is your Array.**
- **Each book on the shelf is an item/element in the array.**
- **The position of each book (e.g., the first, second, third book) is its index.**

1. Creating Arrays

You can create arrays in several ways:

- **Array Literal (most common):**

```
let fruits = ["apple", "banana", "orange"];
let numbers = [1, 2, 3, 4, 5];
let mixed = [1, "hello", true, { key: "value" }];
```

- **Array Constructor:**

```
let fruits = new Array("apple", "banana", "orange");
let numbers = new Array(1, 2, 3, 4, 5);
let emptyArray = new Array(); // Creates an empty array
let arrayWithSize = new Array(5); // Creates an array with 5 empty slots
```

2. Accessing Array Elements

Elements in an array are accessed using their zero-based index:

```
let fruits = ["apple", "banana", "orange"];
console.log(fruits[0]); // Output: "apple"
console.log(fruits[1]); // Output: "banana"
console.log(fruits[2]); // Output: "orange"
console.log(fruits[fruits.length - 1]); // Accessing the last element
```

3. Modifying Array Elements

You can change the value of an element by assigning a new value to its index:

```
let fruits = ["apple", "banana", "orange"];
fruits[1] = "grape";
console.log(fruits); // Output: ["apple", "grape", "orange"]
```

4. Array Length

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

```
let fruits = ["apple", "banana", "orange"];
.log(fruits.length); // Output: 3
```

5. Iterating Over Arrays

There are several ways to iterate over array elements:

- **for loop:**

```
let fruits = ["apple", "banana", "orange"];
for (let i = 0; i < fruits.length; i++) {
 console.log(fruits[i]);
}
```

- **for...of loop (ES6+):**

```
Let fruits = ["apple", "banana", "orange"];
for (let fruit of fruits) {
 console.log(fruit);
}
```

- **forEach() method (covered below):**

```
let fruits = ["apple", "banana", "orange"];
fruits.forEach(function(fruit, index) {
    console.log(`${index}: ${fruit}`);
});
```

JavaScript Array Methods

Here's a breakdown of common and useful array methods, categorized by their primary function:

A. Adding/Removing Elements

1. **push()**
   - Adds one or more elements to the end of an array.
   - Returns the new length of the array.
   - Modifies the original array.

```
let arr = [1, 2];
arr.push(3, 4); // arr is now [1, 2, 3, 4]
console.log(arr.length); // 4
```

2. **pop()**
   - Removes the last element from an array.
   - Returns the removed element.
   - Modifies the original array.

```
let arr = [1, 2, 3];
let lastElement = arr.pop(); // lastElement is 3, arr is now [1, 2]
```

3. **unshift()**
    - Adds one or more elements to the beginning of an array.
    - Returns the new length of the array.

```
let arr = [3, 4];
arr.unshift(1, 2); // arr is now [1, 2, 3, 4]
console.log(arr.length); // 4
```

4. **shift()**
    - Removes the first element from an array.
    - Returns the removed element.
    - Modifies the original array.

```
let arr = [1, 2, 3];
let firstElement = arr.shift(); // firstElement is 1, arr is now [2, 3]
```

5. **splice(startIndex, deleteCount, item1, item2, ...)**
    - A versatile method for adding, removing, or replacing elements at any position.
    - Returns an array containing the deleted elements (if any).
    - Modifies the original array.

```
let colors = ["red", "green", "blue", "yellow"];

// Remove 1 element at index 1
let removed = colors.splice(1, 1); // removed is ["green"], colors is ["red", "blue",
"yellow"]
console.log(colors);
// Add elements at index 1 (no deletion)

colors.splice(1, 0, "purple", "orange"); // colors is ["red", "purple", "orange", "blue",
"yellow"]
console.log(colors);
// Replace 2 elements at index 2 with new ones

.splice(2, 2, "pink", "black"); // colors is ["red", "purple", "pink", "black", "yellow"]
console.log(colors)
```
;

B. Searching/Finding Elements

1. **indexOf(element, fromIndex)**
   - Returns the first index at which a given element can be found, or -1 if not present.
   - fromIndex (optional) specifies the index to start the search from.

```
let numbers = [10, 20, 30, 20, 40];
console.log(numbers.indexOf(20));    // 1
console.log(numbers.indexOf(50));    // -1
console.log(numbers.indexOf(20, 2)); // 3
```

2. **lastIndexOf(element, fromIndex)**
   - Returns the last index at which a given element can be found, or -1 if not present.
   - fromIndex (optional) specifies the index to start the search *backwards* from.

```
let numbers = [10, 20, 30, 20, 40];
console.log(numbers.lastIndexOf(20));    // 3
console.log(numbers.lastIndexOf(50));    // -1
console.log(numbers.lastIndexOf(20, 2)); // 1
```

3. **includes(element, fromIndex) (ES7+)**
   - Checks if an array contains a certain element.
   - Returns true or false.
   - fromIndex (optional) specifies the index to start the search from.

```
let fruits = ["apple", "banana", "orange"];
console.log(fruits.includes("banana"));  // true
console.log(fruits.includes("grape"));   // false
```

4. **find(callbackFunction) (ES6+)**
   - Returns the **first element** in the array that satisfies the provided callbackFunction.
   - Returns undefined if no element satisfies the condition.
   - The callbackFunction receives (element, index, array).

```
let ages = [12, 18, 25, 30];
let adult = ages.find(age => age >= 18); // adult is 18
let senior = ages.find(age => age > 60); // senior is undefined
```

5. **findIndex(callbackFunction) (ES6+)**
   - Returns the **index of the first element** in the array that satisfies the provided callbackFunction.
   - Returns -1 if no element satisfies the condition.
   - The callbackFunction receives (element, index, array).

```
let ages = [12, 18, 25, 30];
let adultIndex = ages.findIndex(age => age >= 18); // adultIndex is 1
let seniorIndex = ages.findIndex(age => age > 60); // seniorIndex is -1
```

C. Iteration and Transformation

1. **forEach(callbackFunction)**
   - Executes a provided callbackFunction once for each array element.
   - Does not return a new array.
   - The callbackFunction receives (element, index, array).

```
let numbers = [1, 2, 3];
numbers.forEach(function(num) {
 console.log(num * 2);
});
// Output: 2, 4, 6
```

2. **map(callbackFunction)**
   - Creates a **new array** populated with the results of calling a provided callbackFunction on every element.
   - Does not modify the original array.
   - The callbackFunction receives (element, index, array).

```
let numbers = [1, 2, 3];
let doubledNumbers = numbers.map(num => num * 2); // doubledNumbers is [2,
4, 6]
console.log(numbers);       // [1, 2, 3] (original unchanged)
```

3. **filter(callbackFunction)**
   - Creates a **new array** with all elements that pass the test implemented by the provided callbackFunction.
   - Does not modify the original array.
   - The callbackFunction receives (element, index, array).

```
let numbers = [1, 2, 3, 4, 5];
let evenNumbers = numbers.filter(num => num % 2 === 0); // evenNumbers is [2,
        4]
```

4. **reduce(callbackFunction, initialValue)**
   - Executes a callbackFunction on each element of the array, resulting in a single output value.
   - The callbackFunction receives (accumulator, currentValue, currentIndex, array).
   - accumulator: The value resulting from the previous call to callbackFunction.
   - initialValue (optional): Value to use as the first argument to the first call of the callbackFunction.

```
let numbers = [1, 2, 3, 4];

// Sum all numbers
let sum = numbers.reduce((acc, current) => acc + current, 0); // sum is 10

// Flatten an array of arrays
let arrayOfArrays = [[1, 2], [3, 4], [5]];
let flattened = arrayOfArrays.reduce((acc, current) => acc.concat(current), []); //
flattened is [1, 2, 3, 4, 5]
```

5. **reduceRight(callbackFunction, initialValue)**
   - Similar to reduce(), but processes the array from right to left.

D. Testing Elements

1. **every(callbackFunction)**
   - Tests whether **all** elements in the array pass the test implemented by the provided callbackFunction.
   - Returns true if all elements pass, false otherwise.

```
let numbers = [2, 4, 6, 8];
let allEven = numbers.every(num => num % 2 === 0); // allEven is true
let mixedNumbers = [2, 3, 4];
let allEvenMixed = mixedNumbers.every(num => num % 2 === 0); //
allEvenMixed is false
```

2. **some(callbackFunction)**
   - Tests whether **at least one** element in the array passes the test implemented by the provided callbackFunction.
   - Returns true if at least one element passes, false otherwise.

```
let numbers = [1, 3, 5, 6];
let hasEven = numbers.some(num => num % 2 === 0); // hasEven is true
let oddNumbers = [1, 3, 5];
hasEvenOdd = oddNumbers.some(num => num % 2 === 0); // hasEvenOdd is false
```

E. Joining/Splitting Arrays

1. **join(separator)**
   - Joins all elements of an array into a string.
   - separator (optional) specifies the string to separate each element. Defaults to a comma (,).

```
let fruits = ["apple", "banana", "orange"];
let fruitString = fruits.join(", "); // fruitString is "apple, banana, orange"
let hyphenated = fruits.join("-");   // hyphenated is "apple-banana-orange"
```

2. **concat(array1, array2, ...)**
   - Used to merge two or more arrays.
   - Returns a **new array**.
   - Does not modify the existing arrays.

```
let arr1 = [1, 2];
let arr2 = [3, 4];
let arr3 = [5, 6];
let combined = arr1.concat(arr2, arr3); // combined is [1, 2, 3, 4, 5, 6]
console.log(arr1); // [1, 2] (original unchanged)
```

F. Reordering/Modifying Arrays

1. **reverse()**
   - Reverses the order of the elements in an array.
   - Modifies the original array.
   - Returns the reversed array.

```
let numbers = [1, 2, 3, 4];
numbers.reverse(); // numbers is now [4, 3, 2, 1]
```

2. **sort(compareFunction)**
   - Sorts the elements of an array in place.
   - Modifies the original array.
   - Returns the sorted array.
   - By default, sorts elements as strings (lexicographically). For numbers, provide a compareFunction.

```
let fruits = ["banana", "apple", "orange"];
fruits.sort(); // fruits is ["apple", "banana", "orange"]
let numbers = [3, 1, 10, 2];
numbers.sort(); // numbers is [1, 10, 2, 3] (incorrect numeric sort)

// Numeric sort
numbers.sort((a, b) => a - b); // Ascending: [1, 2, 3, 10]
numbers.sort((a, b) => b - a); // Descending: [10, 3, 2, 1]
```

G. Sub-arrays

1. **slice(startIndex, endIndex)**
   - Returns a **shallow copy** of a portion of an array into a new array object.
   - startIndex (optional): The index to begin extraction. Defaults to 0.
   - endIndex (optional): The index before which to end extraction. If omitted, extracts to the end.
   - Does not modify the original array.

```
let numbers = [1, 2, 3, 4, 5];
let subArray1 = numbers.slice(1, 4); // subArray1 is [2, 3, 4]
let subArray2 = numbers.slice(2);    // subArray2 is [3, 4, 5]
let copyArray = numbers.slice();     // copyArray is [1, 2, 3, 4, 5] (full copy)
```

H. Filling Arrays

1. **fill(value, startIndex, endIndex) (ES6+)**
   - Fills all the elements of an array from a startIndex to an endIndex with a static value.
   - Modifies the original array.

```
let arr = [1, 2, 3, 4, 5];
```

```
arr.fill(0);          // arr is [0, 0, 0, 0, 0]
let arr2 = [1, 2, 3, 4, 5];
arr2.fill(0, 2, 4);   // arr2 is [1, 2, 0, 0, 5]
```

I. Other Useful Methods

1. **toString()**
   - Returns a string representing the specified array and its elements.
   - Elements are converted to strings and separated by commas.

```
let numbers = [1, 2, 3];
console.log(numbers.toString()); // "1,2,3"
```

2. **at(index) (ES2022)**
   - Takes an integer value and returns the item at that index.
   - Allows for positive and negative integers. Negative integers count back from the last item.

```
const array = [5, 12, 8, 130, 44];
console.log(array.at(2));  // 8
console.log(array.at(-1)); // 44 (last element)
```

3. **flat(depth) (ES2019)**
   - Creates a new array with all sub-array elements recursively "flattened" up to the specified depth.
   - depth (optional): The number of levels to flatten. Defaults to 1. Use Infinity to flatten all nested arrays.

```
const arr1 = [1, 2, [3, 4]];
arr1.flat(); // [1, 2, 3, 4]
const arr2 = [1, 2, [3, 4, [5, 6]]];
arr2.flat(2); // [1, 2, 3, 4, 5, 6]
const arr3 = [1, 2, [3, 4, [5, 6, [7, 8]]]];
arr3.flat(Infinity); // [1, 2, 3, 4, 5, 6, 7, 8]
```

4. **flatMap(callbackFunction) (ES2019)**
   - Maps each element using a mapping function, then flattens the result into a new array.
   - Equivalent to map() followed by flat(1).

```
let words = ["hello world", "how are you"];
```

```
    let individualWords = words.flatMap(sentence => sentence.split(" "));
    // individualWords is ["hello", "world", "how", "are", "you"]
```