Method Category	Method Name	Description
Adding/Removing Elements	**push()**	Adds one or more elements to the end of the array.
	pop()	Removes the last element from the array and returns it.
	unshift()	Adds one or more elements to the beginning of the array.
	shift()	Removes the first element from the array and returns [1] it.
	splice()	Adds/removes elements at any position in the array.
	concat()	Creates a new array by concatenating existing arrays.
Iteration and Access	**forEach()**	Executes a provided function once for each array element.
	map()	Creates a new array with the results of calling a provided function on every element in [2] the array.
	filter()	Creates a new array with all elements that pass the test implemented by the provided function.
	reduce()	Applies a function against an accumulator and each element in the array (from left to right) to reduce it to a single value.
	reduceRight()	Same as reduce(), but processes the array from right to left.
	find()	Returns the first element in the array that satisfies the provided testing function.
	findIndex()	Returns the index of the first element in the array that satisfies the provided testing function.
	some()	Checks if at least one element in the array passes the test implemented by the provided function.
	every()	Checks if all elements in the array pass the test implemented by the provided function.
	indexOf()	Returns the first index at which a given element can be found in the array, or -1 if it is not present.
	lastIndexOf()	Returns the last index at which a given element can be found in the array, or -1 if it is not present.
	includes()	Determines whether an array includes a certain value among its entries. [3]
Sorting and Searching	**sort()**	Sorts the elements of an array in place and returns the sorted array.
	reverse()	Reverses the order of the elements of an array in place. [4]
Joining and Slicing	**join()**	Creates and returns a new string by concatenating the elements of an array, separated by commas or a specified separator.
	slice()	Extracts a section of the array and returns a new array.

Adding/Removing Elements

1. push() - Add elements to the end of an array.

Description: The `push()` method adds one or more elements to the end of an array and returns the new length of the array.

Syntax:

```javascript
array.push(element1, element2, ..., elementN);
```

Example:

```javascript
let fruits = ['apple', 'banana'];
fruits.push('orange');
console.log(fruits); // ['apple', 'banana', 'orange']

// Add multiple elements
fruits.push('grape', 'pear');
console.log(fruits); // ['apple', 'banana', 'orange', 'grape', 'pear']
```

2. pop() - Remove the last element of an array.

Description: The `pop()` method removes the last element from an array and returns that element.

Syntax:

```javascript
let element = array.pop();
```

Example:

```javascript
let fruits = ['apple', 'banana', 'orange'];
let removedFruit = fruits.pop();
console.log(fruits); // ['apple', 'banana']
console.log(removedFruit); // 'orange'
```

3. unshift() - Add elements to the beginning of an array.

Description: The `unshift()` method adds one or more elements to the beginning of an array and returns the new length of the array.

Syntax:

javascript Copy code

```javascript
array.unshift(element1, element2, ..., elementN);
```

Example:

javascript Copy code

```javascript
let fruits = ['banana', 'orange'];
fruits.unshift('apple');
console.log(fruits); // ['apple', 'banana', 'orange']

// Add multiple elements
fruits.unshift('grape', 'pear');
console.log(fruits); // ['grape', 'pear', 'apple', 'banana', 'orange']
```

4. shift() - Remove the first element of an array.

Description: The `shift()` method removes the first element from an array and returns that element.

Syntax:

javascript Copy code

```javascript
let element = array.shift();
```

Example:

javascript Copy code

```javascript
let fruits = ['apple', 'banana', 'orange'];
let removedFruit = fruits.shift();
console.log(fruits); // ['banana', 'orange']
console.log(removedFruit); // 'apple'
```

5. splice() - Add/remove elements at specific indexes.

Description: The `splice()` method can be used to add, remove, or replace elements at a specific index in an array. The method can remove elements or insert new ones without creating a new array.

Syntax:

```javascript
array.splice(startIndex, deleteCount, element1, element2, ..., elementN);
```

- `startIndex`: The index at which to start changing the array.
- `deleteCount`: The number of elements to remove (optional).
- `element1, element2, ..., elementN`: The elements to add (optional).

Example (Removing elements):

```javascript
let fruits = ['apple', 'banana', 'orange', 'grape'];
let removedFruits = fruits.splice(1, 2); // Removes 2 elements starting from index 1
console.log(fruits); // ['apple', 'grape']
console.log(removedFruits); // ['banana', 'orange']
```

Example (Adding elements):

```javascript
let fruits = ['apple', 'banana'];
fruits.splice(1, 0, 'orange', 'grape'); // Adds 'orange' and 'grape' at index 1
console.log(fruits); // ['apple', 'orange', 'grape', 'banana']
```

Example (Replacing elements):

```javascript
let fruits = ['apple', 'banana', 'orange'];
fruits.splice(1, 1, 'grape'); // Replaces 'banana' with 'grape' at index 1
console.log(fruits); // ['apple', 'grape', 'orange']
```

6. concat() - Combine arrays.

Description: The `concat()` method is used to merge two or more arrays into a new array. It does not modify the original arrays.

Syntax:

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

Example:

```javascript
let fruits1 = ['apple', 'banana'];
let fruits2 = ['orange', 'grape'];
let combinedFruits = fruits1.concat(fruits2);
console.log(combinedFruits); // ['apple', 'banana', 'orange', 'grape']

// Concatenating multiple arrays
let fruits3 = ['pear', 'kiwi'];
let allFruits = fruits1.concat(fruits2, fruits3);
console.log(allFruits); // ['apple', 'banana', 'orange', 'grape', 'pear', 'kiwi']
```

Iteration and Access

1. forEach() - Executes a function for each element in the array.

Description: The `forEach()` method executes a provided function once for each element in the array. It does not return anything.

Syntax:

javascript

```javascript
array.forEach(callback(currentValue, index, array), thisArg);
```

Example:

javascript

```javascript
let fruits = ['apple', 'banana', 'orange'];
fruits.forEach(function(fruit, index) {
    console.log(index, fruit);
});
// Output:
// 0 'apple'
// 1 'banana'
// 2 'orange'
```

2. map() - Creates a new array with the results of calling a provided function on every element in the array.

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

Syntax:

javascript

```javascript
let newArray = array.map(callback(currentValue, index, array), thisArg);
```

Example:

javascript

```javascript
let numbers = [1, 2, 3, 4];
let squaredNumbers = numbers.map(num => num * num);
console.log(squaredNumbers); // [1, 4, 9, 16]
```

3. filter() - Creates a new array with all elements that pass the test implemented by the provided function.

Description: The `filter()` method creates a new array with all elements that pass a given test (provided by a function).

Syntax:

```javascript
let newArray = array.filter(callback(currentValue, index, array), thisArg);
```

Example:

```javascript
let numbers = [1, 2, 3, 4, 5, 6];
let evenNumbers = numbers.filter(num => num % 2 === 0);
console.log(evenNumbers); // [2, 4, 6]
```

4. reduce() - Applies a function against an accumulator and each element in the array (from left to right) to reduce it to a single value.

Description: The `reduce()` method applies a function against an accumulator and each element (from left to right) in the array to reduce it to a single value (e.g., sum, product).

Syntax:

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

Example:

```javascript
let numbers = [1, 2, 3, 4];
let sum = numbers.reduce((acc, num) => acc + num, 0);
console.log(sum); // 10
```

5. reduceRight() - Same as `reduce()`, but processes the array from right to left.

Description: The `reduceRight()` method works similarly to `reduce()`, but iterates over the array from the last element to the first.

Syntax:

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

Example:

```javascript
let numbers = [1, 2, 3, 4];
let product = numbers.reduceRight((acc, num) => acc * num, 1);
console.log(product); // 24 (4 * 3 * 2 * 1)
```

6. find() - Returns the first element that satisfies the provided testing function.

Description: The `find()` method returns the first element in the array that satisfies the provided testing function. If no element is found, it returns `undefined`.

Syntax:

```javascript
let result = array.find(callback(currentValue, index, array), thisArg);
```

Example:

```javascript
let numbers = [5, 12, 8, 130, 44];
let found = numbers.find(num => num > 10);
console.log(found); // 12
```

7. findIndex() - Returns the index of the first element that satisfies the provided testing function.

Description: The `findIndex()` method returns the index of the first element that satisfies the provided testing function. If no element is found, it returns `-1`.

Syntax:

javascript

```javascript
let index = array.findIndex(callback(currentValue, index, array), thisArg);
```

Example:

javascript

```javascript
let numbers = [5, 12, 8, 130, 44];
let index = numbers.findIndex(num => num > 10);
console.log(index); // 1
```

8. some() - Checks if at least one element in the array satisfies the provided function.

Description: The `some()` method checks if at least one element in the array satisfies the provided testing function. It returns `true` if any element satisfies the condition, otherwise `false`.

Syntax:

javascript

```javascript
let result = array.some(callback(currentValue, index, array), thisArg);
```

Example:

javascript

```javascript
let numbers = [1, 2, 3, 4];
let hasEven = numbers.some(num => num % 2 === 0);
console.log(hasEven); // true
```

9. every() - Checks if all elements in the array satisfy the provided function.

Description: The `every()` method checks if all elements in the array satisfy the provided testing function. It returns `true` if all elements pass the test, otherwise `false`.

Syntax:

javascript Copy code

```javascript
let result = array.every(callback(currentValue, index, array), thisArg);
```

Example:

javascript Copy code

```javascript
let numbers = [2, 4, 6, 8];
let allEven = numbers.every(num => num % 2 === 0);
console.log(allEven); // true
```

10. indexOf() - Returns the first index at which a given element can be found in the array, or -1 if not found.

Description: The `indexOf()` method returns the index of the first occurrence of a specified element in the array, or `-1` if it is not found.

Syntax:

javascript Copy code

```javascript
let index = array.indexOf(element, fromIndex);
```

Example:

javascript Copy code

```javascript
let fruits = ['apple', 'banana', 'orange'];
let index = fruits.indexOf('banana');
console.log(index); // 1
```

11. lastIndexOf() - Returns the last index at which a given element can be found in the array, or -1 if not found.

Description: The `lastIndexOf()` method returns the last index at which a given element can be found in the array, or `-1` if it is not found.

Syntax:

javascript Copy code

```javascript
let index = array.lastIndexOf(element, fromIndex);
```

Example:

javascript Copy code

```javascript
let fruits = ['apple', 'banana', 'orange', 'banana'];
let index = fruits.lastIndexOf('banana');
console.log(index); // 3
```

12. includes() - Checks if the array contains a certain element.

Description: The `includes()` method checks if the array contains a specified element. It returns `true` if the element is found, otherwise `false`.

Syntax:

javascript Copy code

```javascript
let result = array.includes(element, fromIndex);
```

Example:

javascript Copy code

```javascript
let fruits = ['apple', 'banana', 'orange'];
let hasOrange = fruits.includes('orange');
console.log(hasOrange); // true
```

Sorting and Searching

1. sort() - Sort elements (alphabetically or numerically with custom logic).

Description: The `sort()` method sorts the elements of an array **in place**, meaning the original array is changed. By default, it sorts elements as strings in **lexicographical (alphabetical) order**, which can lead to unexpected results for numbers. You can provide a custom comparator function to sort elements numerically or by other criteria.

Syntax:

javascript Copy code

```javascript
array.sort(compareFunction);
```

- **compareFunction**: A function that defines the sorting order. It takes two arguments and returns a negative, zero, or positive value.

Example (Alphabetical sorting):

javascript Copy code

```javascript
let fruits = ['banana', 'apple', 'orange'];
fruits.sort();
console.log(fruits); // ['apple', 'banana', 'orange']
```

Example (Numerical sorting):

javascript Copy code

```javascript
let numbers = [10, 2, 5, 3];
numbers.sort((a, b) => a - b); // Ascending order
console.log(numbers); // [2, 3, 5, 10]

numbers.sort((a, b) => b - a); // Descending order
console.log(numbers); // [10, 5, 3, 2]
```

Example (Custom sorting by string length):

javascript Copy code

```javascript
let fruits = ['banana', 'apple', 'orange'];
fruits.sort((a, b) => a.length - b.length); // Sort by length of string
console.log(fruits); // ['apple', 'banana', 'orange']
```

2. reverse() - Reverse the array.

Description: The `reverse()` method reverses the order of the elements in the array **in place** (modifies the original array). The first element becomes the last, and the last becomes the first.

Syntax:

javascript

Copy code

```javascript
array.reverse();
```

Example:

javascript

Copy code

```javascript
let fruits = ['apple', 'banana', 'orange'];
fruits.reverse();
console.log(fruits); // ['orange', 'banana', 'apple']
```

Joining and Slicing

1. join() - Convert an array to a string with a specified delimiter.

Description: The `join()` method combines all elements of an array into a single string, with a specified delimiter between them. If no delimiter is provided, the default is a comma (`,`).

Syntax:

javascript Copy code

```javascript
let result = array.join(separator);
```

- `separator` (optional): The string that separates the array elements in the resulting string. If omitted, it defaults to `,`.

Example:

javascript Copy code

```javascript
let fruits = ['apple', 'banana', 'orange'];
let result = fruits.join(', ');
console.log(result); // 'apple, banana, orange'

// Without specifying a separator (defaults to a comma)
let result2 = fruits.join();
console.log(result2); // 'apple,banana,orange'

// Using a custom separator
let result3 = fruits.join(' - ');
console.log(result3); // 'apple - banana - orange'
```

2. slice() - Extract a section of the array into a new array.

Description: The `slice()` method returns a shallow copy of a portion of an array into a new array. It does not modify the original array. You can specify the beginning and end indices to slice a section from the array.

Syntax:

```javascript
let newArray = array.slice(beginIndex, endIndex);
```

- `beginIndex`: The index at which to begin the slice (inclusive).
- `endIndex` (optional): The index at which to end the slice (exclusive). If omitted, the slice extends to the end of the array.

Example:

```javascript
let fruits = ['apple', 'banana', 'orange', 'grape', 'kiwi'];

// Extract elements starting from index 1 to 3 (exclusive)
let slicedFruits = fruits.slice(1, 4);
console.log(slicedFruits); // ['banana', 'orange', 'grape']

// Extract from index 2 to the end of the array
let slicedFruits2 = fruits.slice(2);
console.log(slicedFruits2); // ['orange', 'grape', 'kiwi']

// Extract a portion without specifying end index (copies to the end)
let slicedFruits3 = fruits.slice(1, 1);
console.log(slicedFruits3); // []
```

Key Points:

- `join()` is useful when you need to turn an array into a string, especially with custom delimiters like commas, spaces, or other characters.
- `slice()` helps when you want to extract a part of the array without modifying the original array, giving you flexibility in manipulating the array's data.