

JavaScript Array Methods

push()

sort()

indexOf()

pop()

includes()

lastIndexOf()

shift()

slice()

reverse()

unshift()

map()

concat()

find()

filter()

join()

some()

reduce()

toString()

every()

forEach()



save it, like and share

swipe →



push()

Adds one or more elements to the end of an array and returns the new length of the array..



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

swipe →



pop()

Removes the last element from an array and returns that element.



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

swipe →



shift()

Removes the first element from an array and returns that element.



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

swipe →



unshift()

Adds one or more elements to the beginning of an array and returns the new length of the array.



```
const numbers = [1, 2, 3];
numbers.unshift(0, -1);
console.log(numbers); // [0, -1, 1, 2, 3]
```

swipe →



find()

Returns the value of the **first** element in the array that **satisfies** the provided testing function. Otherwise, undefined is returned.



```
const numbers = [1, 2, 3, 4, 5];
const foundNumber = numbers.find((num) => num > 3);
console.log(foundNumber); // 4
```

swipe →



some()

Tests whether at least one element in the array passes the test implemented by the provided function. It returns true if any element passes the test, otherwise it returns false.



```
const numbers = [1, 2, 3, 4, 5];
const hasEvenNumber = numbers.some(
  (num) => num % 2 === 0);
console.log(hasEvenNumber); // true
```

swipe →



every()

Tests whether **all** elements in the array pass the **test** implemented by the provided function. It returns true if all elements pass the test, otherwise it returns false.



```
const numbers = [1, 2, 3, 4, 5];
const allEvenNumbers = numbers.every
((num) => num % 2 === 0);
console.log(allEvenNumbers); // false
```

swipe →



sort()

Sorts the elements of an array in place and returns the sorted array. The default sort order is built upon converting the elements into strings, then comparing their sequences of UTF-16 code units values.

```
● ● ●  
  
const fruits = ['banana', 'apple', 'orange', 'grape'];  
fruits.sort();  
console.log(fruits);  
// ['apple', 'banana', 'grape', 'orange']
```



```
const numbers = [100, 20, 200, 30];  
numbers.sort((a, b) => a - b);  
console.log(numbers); // [20, 30, 100, 200]
```

swipe →



includes()

Determines whether an array includes a certain element, returning true or false as appropriate.



```
const numbers = [1, 2, 3, 4, 5];
const includesThree = numbers.includes(3);
console.log(includesThree); // true
```

swipe →



slice()

Returns a shallow copy of a portion of an array into a new array object selected from start to end (end not included). The original array will not be modified.



```
const numbers = [1, 2, 3, 4, 5];
const slicedNumbers = numbers.slice(0, 2);
console.log(slicedNumbers); // [1,2]
```

swipe →



map()

Creates a **new** array with the results of calling a provided function on **every** element in the calling array.



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

swipe →



filter()

Creates a **new** array with all elements that **pass** the test implemented by the provided function.



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

swipe →



reduce()

Executes a reducer function on each element of the array, resulting in a **single output value**.



```
const numbers = [1, 2, 3, 4, 5];
const sum = numbers.reduce((total, num) =>
  total + num, 0);
console.log(sum); // 15
```

swipe →



forEach()

Executes a provided function once for each array element.



```
const numbers = [1, 2, 3];
numbers.forEach((num) =>
  console.log(num * 2)); // 2, 4, 6
```

swipe →



indexOf()

Returns the **first index** at which a given element can be found in the **array**, or **-1** if it is not present.



```
const fruits =  
  ['banana', 'apple', 'orange', 'grape'];  
const appleIndex = fruits.indexOf('apple');  
console.log(appleIndex); // 1
```

swipe →



lastIndexof()

Returns the **last index** at which a given element can be found in the **array**, or -1 if it is not present.



```
const fruits =  
['banana', 'apple', 'orange', 'grape', 'apple'];  
const lastAppleIndex = fruits.lastIndexof('apple');  
console.log(lastAppleIndex); // 4
```

swipe →



reverse()

Reverses the order of the elements of an array in place. The first element becomes the last, and the last element becomes the first.



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

swipe →



concat()

Returns a new array that includes elements from the original array and additional elements.



```
const numbers = [1, 2, 3];
const moreNumbers = [4, 5];
const allNumbers = numbers.concat(
  moreNumbers);
console.log(allNumbers);
// [1, 2, 3, 4, 5]
```

swipe →



join()

Joins all elements of an array into a string.
The elements are separated by a specified
separator string.



```
const fruits =  
['banana', 'apple', 'orange', 'grape'];  
const joinedFruits = fruits.join(', ');  
console.log(joinedFruits);  
// 'banana, apple, orange, grape'
```

swipe →



toString()

Returns a **string** representing the specified number or array and its elements.



```
const numbers = [1, 2, 3];
const numbersString = numbers.toString();
console.log(numbersString); // '1, 2, 3'
```

swipe →

Thats a Wrap!

If you liked it, visit my profile and checkout for other
short and easy explanations

Context API vs Redux-Toolkit

Feature ▾	Context API ▾	Redux-Toolkit ▾
State Management	Not a full-fledged state management tool. Passes down values and update functions, but does not have built-in ability to store, get, update, and notify changes in values.	A full-fledged state management tool with built-in ability to store, get, update, and notify changes in values.
Usage	Best for passing static or infrequently updated values and moderately complex state that does not cause performance issues when passed using props.	Best for managing large-scale, complex state that requires asynchronous actions and side-effects.
Code Complexity	Minimal setup and low learning curve. However, can become complex when used with a large number of components and nested Contexts.	
Performance	Can cause unnecessary re-renders if the state passed down is not simple and can require the use of additional memoization techniques to optimize performance.	
Developer Tools	Does not come with pre-built developer tools but can be used with third-party tools like React DevTools.	
Community	Has a large and active community.	

React

Virtual DOM

useRef()
referencing values in React

When you want a component to remember some information, but you don't want that information to trigger new renders, you can use a ref.

Lets See into

1. How to add a ref to component?
2. How to update a ref's value?
3. How refs are different from state?
4. When to use refs?
5. Best practices for using refs?

{ Current }

⚠ Please Like & Share for no reason

VS

TF is a Virtual DOM?

real DOM

swipe →

JavaScript Evolution

ES6 ES2015

1. let and const
2. Arrow functions
3. Default parameters
4. Rest and spread operators
5. Template literals
6. Destructuring assignment
7. Classes and inheritance
8. Promises for asynchronous programming
9. Symbols for creating unique object keys
10. Iterators and generators

ES9 ES2018

1. Object.getOwnPropertyDescriptors()
2. Spread syntax for objects
3. Promise.prototype.finally()

ES10 ES2019

1. Array.prototype.flat()
2. Array.prototype.flatMap()
3. String.prototype.trimStart()
4. String.prototype.trimEnd()
5. Array.prototype.sort() (stable)

ES11 ES2020

1. BigInt
2. Nullish coalescing operator (??)
3. Optional chaining operator (?)
4. Promise.allSettled()

ES12 ES2021

1. String.prototype.replaceAll()
2. Logical assignment operators (|=, &=&, ??=)

ES13 ES2022

1. Array.prototype.lastIndexOf()
2. Object.hasOwn()
3. at() for strings and arrays
4. Top level await()

share

Redux Toolkit
Easiest Explanation Ever

React Redux Toolkit

swipe →

like and share

Instagram icon @linkinsunil LinkedIn icon @linkinsunil Twitter icon @officialskv

P.S.

**Repost this if you
think your followers
will like it**



Enjoyed this?

1. Follow me
2. Click the  notification
3. Never miss a post