Cheatsheet

with explanation and examples of each of methods for an interview

JS

ARRAY METHODS IN JS

JavaScript provides a variety of built-in methods for working with arrays.Let’s explore them together.

1. **push()**: Adds one or more elements to the end of an array and returns the new length, changing the original array.
2. **pop()**: Removes the last element from an array and returns that element, changing the original array.
3. **shift()**: Removes the first element from an array and returns that element, changing the original array.
4. **unshift()**: Adds one or more elements to the beginning of an array and returns the new length, changing the original array.
5. **concat()**: Returns a new array that combines two or more arrays.
6. **slice()**: Extracts a portion of an array into a new array without modifying the original array.
7. **splice()**: Adds or removes elements from an array, changing the original array.
8. **indexOf()**: Returns the first index at which a given element can be found in the array or -1 if it is not present.
9. **lastIndexOf()**: Returns the last index at which a given element can be found in the array or -1 if it is not present.
10. **includes()**: Determines whether an array includes a certain element, returning true or false.
11. **join()**: Joins all elements of an array into a string.
12. **reverse()**: Reverses the order of the elements in an array, changing the original array.
13. **sort()**: Sorts the elements of an array in place and returns the sorted array, changing the original array.
14. **filter()**: Creates a new array with elements that pass a test function.
15. **map()**: Creates a new array by applying a function to each element of an existing array.
16. **forEach():** Executes a provided function once for each array element.
17. **reduce()**: Reduces the array to a single value by applying a function to each element.
18. **every()**: Tests whether all elements in the array pass the test implemented by the provided function. Returns a boolean value.
19. **some()**: Tests whether at least one element in the array passes the test implemented by the provided function. Returns a boolean value.

These are the most commonly used array methods in JavaScript.

push():

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

pop():

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

shift():

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

unshift():

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

concat():

```
let fruits = ['apple', 'banana'];
let vegetables = ['carrot', 'spinach'];
let combined = fruits.concat(vegetables);
console.log(combined); // ['apple', 'banana', 'carrot', 'spinach']
```

slice():

```
let fruits = ['apple', 'banana', 'orange', 'grape'];
let slicedFruits = fruits.slice(1, 3);
console.log(slicedFruits); // ['banana', 'orange']
console.log(fruits); // ['apple', 'banana', 'orange', 'grape']
```

splice():

```
let fruits = ['apple', 'banana', 'orange', 'grape'];
let removedFruits = fruits.splice(1, 2, 'kiwi', 'watermelon');
console.log(removedFruits); // ['banana', 'orange']
console.log(fruits); // ['apple', 'kiwi', 'watermelon', 'grape']
```

indexOf():

```
let fruits = ['apple', 'banana', 'orange', 'grape'];
console.log(fruits.indexOf('orange')) // 2
 console.log(fruits.indexOf('coco')); // -1
```

lastIndexOf():

```
let fruits = ['apple', 'banana', 'orange', 'banana', 'grape'];
console.log(fruits.lastIndexOf('banana')); // 3
console.log(fruits.lastIndexOf('coco')); // -1
```

includes():

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

join():

```
let fruits = ['apple', 'banana', 'orange'];
let fruitsString = fruits.join('');
console.log(fruitsString); // 'apple, banana, orange'
console.log(typeof fruitsString); // string
```

reverse():

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

sort():

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

let numbers = [5, 2, 8, 1, 3];
numbers.sort((a, b) => a - b);
console.log(numbers); // [1, 2, 3, 5, 8]
```

filter():

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

map():

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

forEach():

```
let fruits = ['apple', 'banana', 'orange'];
fruits.forEach(fruit => console.log(fruit));
// 'apple'
// 'banana'
// 'orange'
```

reduce():

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

every():

```
let numbers = [2, 4, 6, 8, 10];
let allEven = numbers.every(num => num % 2 === 0);
console.log(allEven); // true
```

some():

```
let numbers = [1, 3, 5, 7, 9, 10];
let hasEvenNumber = numbers.some(num => num % 2 === 0);
console.log(hasEvenNumber); // true
```

These are examples for each of the array methods. You can use these methods to perform various operations on arrays in JavaScript.

Important!
Here are the array methods that return a new array:

1. **concat()**: Returns a new array that combines the arrays you provide as arguments.
2. **slice()**: Returns a new array containing the selected elements from the original array.
3. **filter()**: Returns a new array containing the elements that pass the provided test function.
4. **map()**: Returns a new array containing the results of applying a function to each element of the original array.

Remember that these methods **do not modify the original array**; instead, they **create a new array** with the desired changes.

Key differences between **map()** and **forEach()**:

- **Return Value**:
  - **map()** returns a new array with transformed values.
  - **forEach()** does not return anything (**undefined**).
- **Use Case**:
  - Use **map()** when you want to create a new array with modified values.
  - Use **forEach()** when you want to perform an action on each element of the array without creating a new array.
- **Immutability**:
  - **map()** is useful for maintaining immutability by creating a new array with transformed values.
  - **forEach()** doesn't create a new array, so it's often used for actions that don't modify the array's elements.
- **Chaining**:
  - Since **map()** returns a new array, you can chain other array methods after it.
  - **forEach()** does not return an array, so chaining other array methods after it is not common.

In summary, use **map()** when you need to transform elements and create a new array, and use **forEach()** when you want to perform an action on each element of the array.