JavaScript Array Method Examples

JavaScript array methods are built-in functions that are used to manipulate arrays in various ways. These methods allow developers to perform common operations on arrays, such as adding or removing elements, sorting and filtering elements, and transforming arrays into new arrays.

Here are some of the most commonly used JavaScript array methods:

push():

This method adds one or more elements to the end of an array and returns the new length of the array.

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

pop():

This method removes the last element from an array and returns the removed element.

```
let fruits = ['apple', 'banana', 'mango'];
let lastFruit = fruits.pop();
console.log(fruits);   // ["apple", "banana"]
console.log(lastFruit);  // "mango"
```

shift():

This method removes the first element from an array and returns the removed element.

```
let fruits = ['apple', 'banana', 'mango'];
let firstFruit = fruits.shift();
console.log(fruits);   // ["banana", "mango"]
console.log(firstFruit);  // "apple"
```

unshift():

This method adds one or more elements to the beginning of an array and returns the new length of the array.

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

concat():

This method merges two or more arrays and returns a new array without modifying the existing arrays.

```
let fruits = ['apple', 'banana'];
let moreFruits = ['mango', 'orange'];
let allFruits = fruits.concat(moreFruits);
console.log(allFruits);   // ["apple", "banana", "mango", "orange"]
```

slice():

This method extracts a portion of an array and returns a new array without modifying the existing array.

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

splice():

This method adds or removes elements from an array and returns the removed elements.

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

forEach():

This method executes a provided function once for each element in an array.

```
let fruits = ['apple', 'banana', 'mango'];
fruits.forEach(function(fruit) {
  console.log(fruit);
});
// Output:
// apple
// banana
// mango
```

map():

This method creates a new array with the results of calling a provided function on every element in the array.

```
let numbers = [1, 2, 3, 4];
let doubledNumbers = numbers.map(function(number) {
  return number * 2;
});
console.log(doubledNumbers); // [2, 4, 6, 8]
```

filter():

This method creates a new array with all elements that pass the test implemented by the provided function.

```
let numbers = [1, 2, 3, 4, 5];
let evenNumbers = numbers.filter(function(number) {
  return number % 2 === 0;
});
console.log(evenNumbers);   // [2, 4]
```

To use these array methods, you need to call them on an array object and pass in any necessary arguments. For example, to use the push() method to add an element to an array, you would call the method on the array object and pass in the element to be added:

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

Similarly, to use the forEach() method to loop through an array and log each element to the console, you would call the method on the array object and pass in a function that logs each element:

```
let fruits = ['apple', 'banana', 'mango'];
fruits.forEach(function(fruit) {
  console.log(fruit);
});
// Output:
// apple
// banana
// mango
```

forEach() method:

This method executes a provided function once for each element in an array.

```
let fruits = ['apple', 'banana', 'mango'];
fruits.forEach(function(fruit) {
  console.log(fruit);
});
// Output:
// apple
// banana
// mango
```

forEach():

This method executes a provided function once for each array element. It does not change the original array.

```
const numbers = [1, 2, 3, 4, 5];
numbers.forEach(function(number) {
  console.log(number);
});
// Output: 1 2 3 4 5
```

forEach()

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

```
const numbers = [1, 2, 3, 4, 5];
numbers.forEach((number) => console.log(number));
// Output:
// 1
// 2
// 3
// 4
// 5
```

In this example, the forEach() method takes a function that logs each number in the original array to the console.

forEach():

This method executes a provided function once for each array element.

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

forEach():

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

```
let fruits = ['apple', 'banana', 'orange'];
fruits.forEach(function(fruit) {
```

```
  console.log(fruit);
});
// Output:
// apple
// banana
// orange
```

Array.map():

This method creates a new array by calling a provided function on every element in the original array. It does not modify the original array.

```
const numbers = [1, 2, 3, 4, 5];
const squaredNumbers = numbers.map(number => number * number);
console.log(squaredNumbers); // [1, 4, 9, 16, 25]
```

In this example, we create a new array called squaredNumbers by calling the map() method on the numbers array. The map() method takes a function as an argument, which is called on each element in the array. The result of this function is added to the new array.

map():

This method 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); // Output: [2, 4, 6]
```

map():

This method creates a new array with the results of calling a provided function on every element in the original array.

```
const numbers = [1, 2, 3, 4, 5];
const doubledNumbers = numbers.map(function(number) {
  return number * 2;
});
console.log(doubledNumbers);
// Output: [2, 4, 6, 8, 10]
```

map():

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

```
let numbers = [1, 2, 3];
let doubledNumbers = numbers.map(function(number) {
  return number * 2;
});
console.log(doubledNumbers); // [2, 4, 6]
```

map()

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

```
const numbers = [1, 2, 3, 4, 5];
const doubledNumbers = numbers.map((number) => number * 2);
console.log(doubledNumbers); // [2, 4, 6, 8, 10]
```

In this example, the map() method takes a function that multiplies each number in the original array by 2, creating a new array with the doubled numbers.

filter():

This method creates a new array with all elements that pass the test implemented by the provided function.

```
const numbers = [1, 2, 3, 4, 5];
const filteredNumbers = numbers.filter(function(number)
{
  return number % 2 === 0;
});
console.log(filteredNumbers);
// Output: [2, 4]
```

reduce():

This method applies a function against an accumulator and each element in the array to reduce it to a single value.

```
const numbers = [1, 2, 3, 4, 5];
const sum = numbers.reduce(function(accumulator,
currentValue) {
  return accumulator + currentValue;
}, 0);
console.log(sum);
// Output: 15
```

find():

This method returns the first element in the array that satisfies the provided function.

```
const numbers = [1, 2, 3, 4, 5];
const foundNumber = numbers.find(function(number) {
  return number > 3;
});
console.log(foundNumber);
```

```
// Output: 4
```

indexOf():

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

```
const numbers = [1, 2, 3, 4, 5];
const index = numbers.indexOf(3);
console.log(index);
// Output: 2
```

lastIndexOf():

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

```
const numbers = [1, 2, 3, 4, 5, 3];
const index = numbers.lastIndexOf(3);
console.log(index);
// Output: 5
```

every():

This method tests whether all elements in the array pass the test implemented by the provided function. It returns a Boolean value.

```
const numbers = [1, 2, 3, 4, 5];
const allNumbersAreEven =
numbers.every(function(number) {
  return number % 2 === 0;
});
console.log(allNumbersAreEven);
// Output: false
```

some():

This method tests whether at least one element in the array passes the test implemented by the provided function. It returns a Boolean value.

```
const numbers = [1, 2, 3, 4, 5];
const someNumbersAreEven =
numbers.some(function(number) {
  return number % 2 === 0;
});
console.log(someNumbersAreEven);
// Output: true
```

push():

This method adds one or more elements to the end of an array and returns the new length of the array.

```
const fruits = ['apple', 'banana', 'orange'];
fruits.push('grape', 'pineapple');
console.log(fruits); // Output: ['apple', 'banana',
'orange', 'grape', 'pineapple']
```

pop():

This method removes the last element from an array and returns that element.

```
const numbers = [1, 2, 3, 4, 5];
const lastNumber = numbers.pop();
console.log(lastNumber); // Output: 5
console.log(numbers); // Output: [1, 2, 3, 4]
```

shift():

This method removes the first element from an array and returns that element.

```
const colors = ['red', 'green', 'blue'];
const firstColor = colors.shift();
console.log(firstColor); // Output: 'red'
console.log(colors); // Output: ['green', 'blue']
```

unshift():

This method adds one or more elements to the beginning of an array and returns the new length of the array.

```
const animals = ['cat', 'dog', 'bird'];
const newLength = animals.unshift('lion', 'elephant');
console.log(newLength); // Output: 5
console.log(animals); // Output: ['lion', 'elephant',
'cat', 'dog', 'bird']
```

concat():

This method returns a new array that is a combination of two or more arrays.

```
const array1 = [1, 2, 3];
const array2 = [4, 5, 6];
const combinedArray = array1.concat(array2);
console.log(combinedArray); // Output: [1, 2, 3, 4, 5,
6]
```

slice():

This method returns a new array that is a subset of an existing array.

```
const letters = ['a', 'b', 'c', 'd', 'e'];
const subset = letters.slice(1, 4);
console.log(subset); // Output: ['b', 'c', 'd']
```

splice():

This method adds or removes elements from an array at a specified index.

```
const days = ['Monday', 'Tuesday', 'Thursday',
'Friday'];
days.splice(2, 0, 'Wednesday'); // Inserts 'Wednesday'
at index 2
console.log(days); // Output: ['Monday', 'Tuesday',
'Wednesday', 'Thursday', 'Friday']
days.splice(3, 1); // Removes one element at index 3
console.log(days); // Output: ['Monday', 'Tuesday',
'Wednesday', 'Friday']
```

reverse():

This method reverses the order of elements in an array.

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

sort():

This method sorts the elements in an array in alphabetical or numerical order.

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

concat():

This method is used to merge two or more arrays into a new array without changing the original arrays.

```
const array1 = [1, 2, 3];
const array2 = [4, 5, 6];
const newArray = array1.concat(array2);
console.log(newArray); // Output: [1, 2, 3, 4, 5, 6]
```

filter():

This method creates a new array with all elements that pass the test implemented by the provided function.

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

find():

This method returns the value of the first element in the array that satisfies the provided testing function.

```
const numbers = [1, 2, 3, 4, 5];
const evenNumber = numbers.find(num => num % 2 === 0);
console.log(evenNumber); // Output: 2
```

includes():

This method determines whether an array includes a certain element, returning true or false as appropriate.

```
const numbers = [1, 2, 3, 4, 5];
```

```
const includes3 = numbers.includes(3);
console.log(includes3); // Output: true
```

pop():

This method removes the last element from an array and returns that element.

```
const fruits = ['apple', 'banana', 'orange'];
const lastFruit = fruits.pop();
console.log(lastFruit); // Output: orange
console.log(fruits); // Output: ['apple', 'banana']
```

push():

This method adds one or more elements to the end of an array and returns the new length of the array.

```
const fruits = ['apple', 'banana'];
const newLength = fruits.push('orange');
console.log(newLength); // Output: 3
console.log(fruits); // Output: ['apple', 'banana',
'orange']
```

reduce():

This method applies a function against an accumulator and each element in the array to reduce it to a single value.

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

sort():

This method sorts the elements of an array in place and returns the sorted array.

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

Array.filter():

This method creates a new array with all elements that pass a provided test. It does not modify the original array.

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

In this example, we create a new array called evenNumbers by calling the filter() method on the numbers array. The filter() method takes a function as an argument, which is called on each element in the array. The function returns true for elements that pass the test, which are added to the new array.

Array.reduce():

This method applies a function to each element in the array to reduce it to a single value. It does not modify the original array.

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

In this example, we use the reduce() method to calculate the sum of all elements in the numbers array. The first argument to the

reduce() method is a function that takes two arguments: an accumulator and the current value. The function is called on each element in the array, and the result is added to the accumulator. The second argument to the reduce() method is the initial value of the accumulator.

Array.concat():

This method combines two or more arrays into a new array. It does not modify the original arrays.

```
const numbers1 = [1, 2, 3];
const numbers2 = [4, 5, 6];
const combinedNumbers = numbers1.concat(numbers2);
console.log(combinedNumbers); // [1, 2, 3, 4, 5, 6]
```

In this example, we use the concat() method to combine the numbers1 and numbers2 arrays into a new array called combinedNumbers.

Array.slice():

This method returns a new array with a portion of the original array. It does not modify the original array.

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

In this example, we use the slice() method to create a new array called slicedNumbers that contains elements 1 and 2 from the numbers array.

push():

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

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

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

pop():

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

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

shift():

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

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

unshift():

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

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

splice():

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

```
let fruits = ['apple', 'banana', 'orange'];
// Remove 'banana' and add 'pear' and 'kiwi'
fruits.splice(1, 1, 'pear', 'kiwi');
console.log(fruits); // ['apple', 'pear', 'kiwi',
'orange']
```

slice():

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

```
let fruits = ['apple', 'banana', 'orange', 'pear', 'kiwi'];
let citrus = fruits.slice(2, 4);
console.log(citrus); // ['orange', 'pear']
```

concat():

The concat() method returns a new array that includes elements from the original array(s) and/or additional elements passed as arguments.

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

push()

The push() method adds one or more elements to the end of an array and returns the new length of the array. Here's an example:

```
const fruits = ['apple', 'banana', 'orange'];
const newLength = fruits.push('pear', 'peach');
console.log(fruits); // ['apple', 'banana', 'orange',
'pear', 'peach']
console.log(newLength); // 5
```

Array.pop()

The pop() method removes the last element from an array and returns that element. Here's an example:

```
const fruits = ['apple', 'banana', 'orange'];
const lastFruit = fruits.pop();
console.log(fruits); // ['apple', 'banana']
console.log(lastFruit); // 'orange'
```

Array.shift()

The shift() method removes the first element from an array and returns that element. Here's an example:

```
const fruits = ['apple', 'banana', 'orange'];
const firstFruit = fruits.shift();
console.log(fruits); // ['banana', 'orange']
console.log(firstFruit); // 'apple'
```

Array.unshift()

The unshift() method adds one or more elements to the beginning of an array and returns the new length of the array. Here's an example:

```
const fruits = ['apple', 'banana', 'orange'];
const newLength = fruits.unshift('pear', 'peach');
console.log(fruits); // ['pear', 'peach', 'apple',
'banana', 'orange']
console.log(newLength); // 5
```

Array.slice()

The slice() method returns a shallow copy of a portion of an array into a new array. Here's an example:

```
const fruits = ['apple', 'banana', 'orange', 'pear',
'peach'];
const citrusFruits = fruits.slice(2, 4);
console.log(citrusFruits); // ['orange', 'pear']
```

In this example, the slice() method is used to create a new array citrusFruits containing a shallow copy of the portion of the fruits array between the indices 2 and 4 (excluding the element at index 4).

Array.splice()

The splice() method changes the contents of an array by removing or replacing existing elements and/or adding new elements. Here's an example:

```
const fruits = ['apple', 'banana', 'orange', 'pear',
'peach'];
const removedFruits = fruits.splice(1, 3, 'kiwi',
'mango');
console.log(fruits); // ['apple', 'kiwi', 'mango',
'peach']
console.log(removedFruits); // ['banana', 'orange',
'pear']
```

In this example, the splice() method is used to remove three elements starting at index 1, replace them with 'kiwi' and 'mango', and return the removed elements.

filter()

The filter() method 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((number) => number % 2 === 0);
console.log(evenNumbers); // [2, 4]
```

In this example, the filter() method takes a function that returns true for even numbers in the original array, creating a new array with only the even numbers.

reduce()

The reduce() method applies a function against an accumulator and each element in the array to reduce it to a single value.

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

In this example, the reduce() method takes a function that adds each number in the original array to an accumulator, resulting in the sum of all numbers.

sort()

The sort() method sorts the elements of an array in place and returns the sorted array.

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

In this example, the sort() method sorts the original array of numbers in ascending order.

concat()

The concat() method merges two or more arrays and returns a new array.

```
const numbers1 = [1, 2, 3];
const numbers2 = [4, 5, 6];
const mergedNumbers = numbers1.concat(numbers2);
console.log(mergedNumbers); // [1, 2, 3, 4, 5, 6]
```

In this example, the concat() method merges two arrays of numbers into a new array.

includes()

The includes() method 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
```

In this example, the includes() method checks whether the original array includes the number 3 and returns true.

push()

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

```
let fruits = ['apple', 'banana', 'mango'];
let length = fruits.push('orange', 'grape');
console.log(fruits);  // ["apple", "banana", "mango",
"orange", "grape"]
console.log(length);  // 5
```

pop()

pop() method: This method removes the last element from an array and returns the removed element.

```
let fruits = ['apple', 'banana', 'mango'];
let lastFruit = fruits.pop();
console.log(fruits);   // ["apple", "banana"]
console.log(lastFruit);  // "mango"
```

shift()

shift() method: This method removes the first element from an array and returns the removed element.

```
let fruits = ['apple', 'banana', 'mango'];
let firstFruit = fruits.shift();
console.log(fruits);   // ["banana", "mango"]
console.log(firstFruit);  // "apple"
```

unshift()

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

```
let fruits = ['apple', 'banana', 'mango'];
```

```
let length = fruits.unshift('orange', 'grape');
console.log(fruits);   // ["orange", "grape", "apple",
"banana", "mango"]
console.log(length);  // 5
```

concat()

concat() method: This method merges two or more arrays and returns a new array without modifying the existing arrays.

```
let fruits = ['apple', 'banana'];
let moreFruits = ['mango', 'orange'];
let allFruits = fruits.concat(moreFruits);
console.log(allFruits);   // ["apple", "banana",
"mango", "orange"]
```

slice()

slice() method: This method extracts a portion of an array and returns a new array without modifying the existing array.

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

splice()

splice() method: This method adds or removes elements from an array and returns the removed elements.

```
let fruits = ['apple', 'banana', 'mango', 'orange',
'grape'];
let removedFruits = fruits.splice(2, 2, 'kiwi',
'pear');
```

```
console.log(fruits);    // ["apple", "banana", "kiwi",
"pear", "grape"]
console.log(removedFruits);  // ["mango", "orange"]
```