The map() method in JavaScript is used to create a new array by applying a callback function to each element of an existing array. It does not modify the original array but returns a new one.

---

### \*Syntax\*

javascript

array.map(callback(currentValue, index, array), thisArg);

- \*callback\*: A function that is called for each element. It receives the following arguments:

- \*currentValue\*: The current element being processed.

- \*index\*: (Optional) The index of the current element.

- \*array\*: (Optional) The array map() was called on.

- \*thisArg\*: (Optional) Value to use as this when executing the callback.

---

### \*Examples of map()\*

---

### \*1. Basic Example\*

javascript

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

let squares = numbers.map(num => num \* num);

console.log(squares); // Output: [1, 4, 9, 16]

---

### \*2. Using Index\*

You can include the index in the callback function.

javascript

let numbers = [10, 20, 30];

let withIndex = numbers.map((num, index) => num + index);

console.log(withIndex); // Output: [10, 21, 32]

---

### \*3. Mapping Strings\*

The map() method works with arrays of strings as well.

javascript

let names = ["Alice", "Bob", "Charlie"];

let upperCaseNames = names.map(name => name.toUpperCase());

console.log(upperCaseNames); // Output: ["ALICE", "BOB", "CHARLIE"]

---

### \*4. Converting Objects\*

You can use map() to transform an array of objects.

javascript

let users = [

{ firstName: "John", lastName: "Doe" },

{ firstName: "Jane", lastName: "Smith" }

];

let fullNames = users.map(user => `${user.firstName} ${user.lastName}`);

console.log(fullNames); // Output: ["John Doe", "Jane Smith"]

---

### \*5. Chaining with Other Methods\*

map() can be chained with other array methods like filter().

javascript

let numbers = [1, 2, 3, 4, 5];

let evenSquares = numbers.filter(num => num % 2 === 0).map(num => num \* num);

console.log(evenSquares); // Output: [4, 16]

---

### \*6. Using thisArg\*

The thisArg parameter allows you to specify the this context.

javascript

let multiplier = {

factor: 2,

multiply(num) {

return num \* this.factor;

}

};

let numbers = [1, 2, 3];

let result = numbers.map(function (num) {

return this.multiply(num);

}, multiplier);

console.log(result); // Output: [2, 4, 6]

---

### \*7. Modifying Elements\*

You can use map() to create a new array with modified elements.

javascript

let fruits = ["apple", "banana", "cherry"];

let pluralFruits = fruits.map(fruit => fruit + "s");

console.log(pluralFruits); // Output: ["apples", "bananas", "cherries"]

---

### \*8. Creating Array of Properties\*

Extract specific properties from an array of objects.

javascript

let products = [

{ id: 1, name: "Laptop", price: 1000 },

{ id: 2, name: "Phone", price: 500 }

];

let prices = products.map(product => product.price);

console.log(prices); // Output: [1000, 500]

---

### \*9. Nested Arrays\*

map() works with nested arrays as well.

javascript

let matrix = [[1, 2], [3, 4], [5, 6]];

let doubledMatrix = matrix.map(row => row.map(num => num \* 2));

console.log(doubledMatrix); // Output: [[2, 4], [6, 8], [10, 12]]

---

### \*10. Using map() for a Calculation\*

Performing calculations with each element.

javascript

let temperaturesCelsius = [0, 10, 20];

let temperaturesFahrenheit = temperaturesCelsius.map(temp => temp \* 1.8 + 32);

console.log(temperaturesFahrenheit); // Output: [32, 50, 68]

---

### Summary

- The map() method is great for transforming arrays without modifying the original.

- It returns a new array based on the logic defined in the callback function.