Array Object

The Array object is used to store multiple values in a single variable

## String properties

**1. constructor**

The constructor property returns the function that created the Array prototype.

For JavaScript arrays the constructor property returns:

**function Array() { [native code] }**

## Syntax

### *array*.constructor

### **Example**

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
let text = fruits.constructor;

**2. length**

## The length property sets or returns the number of elements in an array.

## Syntax

### *array*.length

### **Example**

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
let length = fruits.length;

**3. prototype**

prototype allows you to add new properties and methods to arrays.

prototype is a property available with all JavaScript objects.

## Syntax

### Array.prototype.*name*= value

### **Example**

var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.myUcase();

## Array Methods

**1. concat()**

The concat() method concatenates (joins) two or more arrays.

The concat() method returns a new array, containing the joined arrays.

The concat() method does not change the existing arrays.

**Syntax**

*array1*.concat(*array2*,*array3*, ...,*arrayX*)

### **Examples**

const arr1 = ["Cecilie", "Lone"];  
const arr2 = ["Emil", "Tobias", "Linus"];  
const children = arr1.concat(arr2);

**2. copyWithin()**

The copyWithin() method copies array elements to another position in the array.

The copyWithin() method overwrites the existing values.

The copyWithin() method does not add items to the array.

**Syntax**

*array*.copyWithin(*target, start, end*)

### **Examples**

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.copyWithin(2, 0);

**3. entries()**

The entries() method returns an Array Iterator object with key/value pairs:

[0, "Banana"]  
[1, "Orange"]  
[2, "Apple"]  
[3, "Mango"]

The entries() method does not change the original array.

**Syntax**

*array*.entries()

### **Example**

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
const f = fruits.entries();  
  
for (let x of f) {  
  document.getElementById("demo").innerHTML += x;  
}

**4. every()**

The every() method executes a function for each array element.

The every() method returns true if the function returns true for all elements.

The every() method returns false if the function returns false for one element.

The every() method does not execute the function for empty elements.

The every() method does not change the original array

**Syntax**

*array*.every(*function(currentValue, index, arr), thisValue*)

### **Example**

const ages = [32, 33, 16, 40];  
  
ages.every(checkAge)  
  
function checkAge(age) {  
  return age > 18;  
}

**5. fill()**

The fill() method fills specified elements in an array with a value.

The fill() method overwrites the original array.

Start and end position can be specified. If not, all elements will be filled.

**Syntax**

*array*.fill(*value, start, end*)

### **Examples**

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.fill("Kiwi");

**6. filter()**

The filter() method creates a new array filled with elements that pass a test provided by a function.

The filter() method does not execute the function for empty elements.

The filter() method does not change the original array.

**Syntax**

*array*.filter(*function(currentValue, index, arr), thisValue*)

### **Example**

const ages = [32, 33, 16, 40];  
const result = ages.filter(checkAdult);  
  
function checkAdult(age) {  
  return age >= 18;  
}

**7. find()**

The find() method returns the value of the first element that passes a test.

The find() method executes a function for each array element.

The find() method retuns undefined if no elements are found.

The find() method does not execute the function for empty elements.

The find() method does not change the original array.

**Syntax**

*array*.find(*function(currentValue, index, arr),thisValue*)

### **Example**

const ages = [3, 10, 18, 20];  
  
function checkAge(age) {  
  return age > 18;  
}  
  
function myFunction() {  
  document.getElementById("demo").innerHTML = ages.find(checkAge);  
}

**8. findIndex()**

The findIndex() method executes a function for each array element.

The findIndex() method returns the index (position) of the first element that passes a test.

The findIndex() method returns -1 if no match is found.

The findIndex() method does not execute the function for empty array elements.

The findIndex() method does not change the original array.

**Syntax**

*array*.findIndex(*function(currentValue, index, arr), thisValue*)

### **Example**

const ages = [3, 10, 18, 20];  
  
ages.findIndex(checkAge);  
  
function checkAge(age) {  
  return age > 18;  
}

**9.** **forEach()**

The forEach() method calls a function for each element in an array.

The forEach() method is not executed for empty elements.

**Syntax**

*array*.forEach(*function(currentValue, index, arr), thisValue*)

### **Example**

Calls a function for each element in fruits:

const fruits = ["apple", "orange", "cherry"];  
fruits.forEach(myFunction);

**10. Array.from()**

The Array.from() method returns an array from any object with a length property.

The Array.from() method returns an array from any iterable object.

**Syntax**

Array.from(*object, mapFunction, thisValue*)

### **Example**

Array.from("ABCDEFG")

**11. valueOf()**

The valueOf() method returns the array itself.

The valueOf() method does not change the original array.

fruits.valueOf() returns the same as fruits.

**Syntax**

*array*.valueOf()

### **Examples**

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
const myArray = fruits.valueOf();

**12. unshift()**

The unshift() method adds new elements to **the beginning** of an array.

The unshift() method overwrites the original array.

**Syntax**

*array*.unshift(*item1*,*item2*, ..., *itemX*)

### **Example**

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.unshift("Lemon","Pineapple");

**13. toString()**

The toString() method returns a string with array values separated by commas.

The toString() method does not change the original array.

**Syntax**

*array*.toString()

### **Example**

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
let text = fruits.toString();

**14. splice()**

The splice()  method adds and/or removes array elements.

The splice() method overwrites the original array.

**Syntax**

*array*.splice(*index*,*howmany*,*item1*, .....,*itemX*)

### **Examples**

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
  
fruits.splice(2, 0, "Lemon", "Kiwi");

**15. sort()**

The sort() sorts the elements of an array.

The sort() overwrites the original array.

The sort() sorts the elements as strings in alphabetical and ascending order.

**Syntax**

*array*.sort(*compareFunction*)

### **Examples**

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.sort();

**16. some()**

The some() method checks if any array elements pass a test (provided as a function).

The some() method executes the function once for each array element:

* If the function returns true, some() returns true and stops.
* If the function returns false, some() returns false and stops.

The some() method does not execute the function for empty array elements.

The some() method does not change the original array.

## Syntax

*array*.some(function(value, index, arr), this)

### **Example**

const ages = [3, 10, 18, 20];  
  
ages.some(checkAdult);  
function checkAdult(age) {  
  return age > 18;  
}

**17. slice()**

The slice() method returns selected elements in an array, as a new array.

The slice() method selects from a given *start*, up to a (not inclusive) given *end*.

The slice() method does not change the original array.

**Syntax**

*array*.slice(*start*,*end*)

### **Examples**

const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];  
const citrus = fruits.slice(1, 3);

**18. shift()**

The shift() method removes **the first item** of an array.

The shift() method changes the original array.

The shift() method returns the shifted element.

**Syntax**

*array*.shift()

### **Examples**

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.shift();

**19. reverse()**

The reverse() method reverses the order of the elements in an array.

The reverse() method overwrites the original array.

**Syntax**

*array*.reverse()

### **Example**

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.reverse();

**20. reduce()**

The reduce() method executes a reducer function for array element.

The reduce() method returns a single value: the function's accumulated result.

The reduce() method does not execute the function for empty array elements.

The reduce() method does not change the original array.

**Syntax**

*array*.reduce(*function(total, currentValue, currentIndex, arr), initialValue*)

### **Examples**

const numbers = [175, 50, 25];  
  
document.getElementById("demo").innerHTML = numbers.reduce(myFunc);  
  
function myFunc(total, num) {  
  return total - num;  
}

**21. push()**

The push() method adds new items **to the end** of an array.

The push() method changes the length of the array.

The push() method returns the new lengt

**Syntax**

*array*.push(*item1*, *item2*, ..., *itemX*)

### **Examples**

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.push("Kiwi");

**22. pop()**

The pop() method removes (pops) **the last element** of an array.

The pop() method changes the original array.

The pop() method returns the removed element.

**Syntax**

*array*.pop()

### **Examples**

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.pop();

**23. map()**

map() creates a new array from calling a function for every array element.

map() calls a function once for each element in an array.

map() does not execute the function for empty elements.

map() does not change the original array.

**Syntax**

*array*.map(*function(currentValue, index, arr), thisValue*)

### **Examples**

const numbers = [4, 9, 16, 25];  
const newArr = numbers.map(Math.sqrt)

**24. lastIndexOf()**

The lastIndexOf() method returns the last index (position) of a specified value.

The lastIndexOf() method returns -1 if the value is not found.

The lastIndexOf() starts at a specified index and searches from right to left.

By defalt the search starts at the last element and ends at the first.

Negative start values counts from the last element (but still searches from right to left).

**Syntax**

*array*.lastIndexOf(*item*,*start*)

### **Examples**

const fruits = ["Apple", "Orange", "Apple", "Mango"];  
let index = fruits.lastIndexOf("Apple");

**25. join()**

The join() method returns an array as a string.

The join() method does not change the original array.

Any separator can be specified. The default is comma (,).

**Syntax**

*array*.join(*separator*)

### **Examples**

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
let text = fruits.join();

**26. keys()**

The keys() method returns an Array Iterator object with the keys of an array.

The keys() method does not change the original array.

**Syntax**

*array*.keys()

### **Example**

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
const keys = fruits.keys();  
  
let text = "";  
for (let x of keys) {  
  text += x + "<br>";  
}