**Arrays & Objects in JavaScript | Web Development Tutorials #52**

In the previous tutorial, we have seen the scope of the variable and different conditional statements like if-else and switch-case. Moving further, we are going to learn about Arrays and Objects used in JavaScript. Make a new file as *tut52.html* and add an instant boilerplate to get the HTML template. Give the title as **Arrays and Objects**under the <title> tag.

The **object**class represents one of JavaScript’s data types. It is used to store various keyed collections and more complex entities. Objects can be created using the **Object()**constructor. we have two types of values used in JavaScript- primitive and reference. To be more simple, either we create *objects*or *primitive data types.*The primitive data types are as follows-

let myVar = 34;

let myVar2 = "string";

let myVar3 = true;

let myVar4 = null;

let myVar5 = undefined;

Copy

Apart from primitive data types, all the other are objects. Let us now see how to define objects. If we write as follows-

let employee = {

name: "Harry",

salary: 10,

channel: "CodeWithHarry",

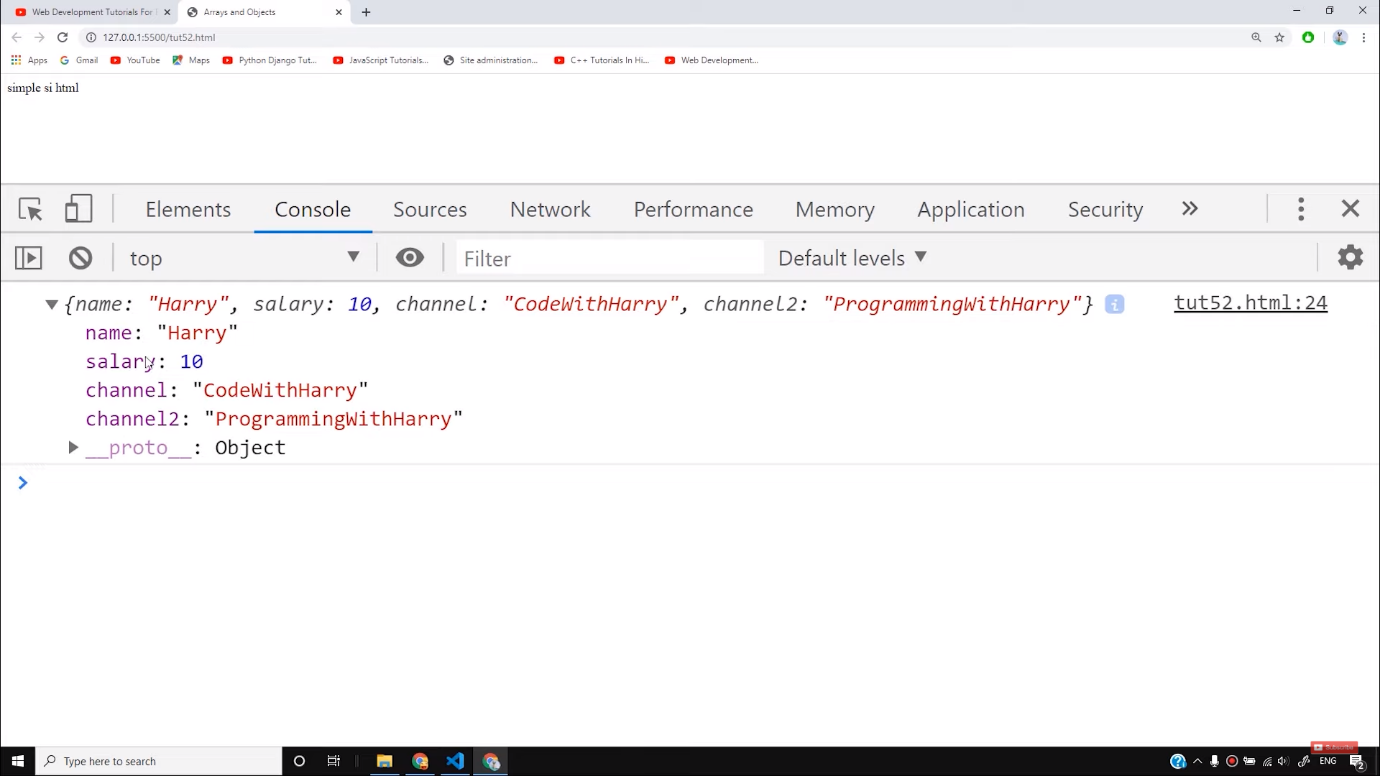
"channel 2": "ProgrammingWithHarry",

}

console.log(employee);

Copy

From the above code, the output will be generated as follows-



In this example, an **employee**is an object that containsthe***name, salary, channel, and channel 2*** inside it. This is an example, where we create objects manually by us.

There is a special kind of object known as **Arrays.** The JavaScript Array class is a global object that is used in the construction of arrays; which are high level, list-like objects. An array is a special variable, which can hold more than one value at a time.

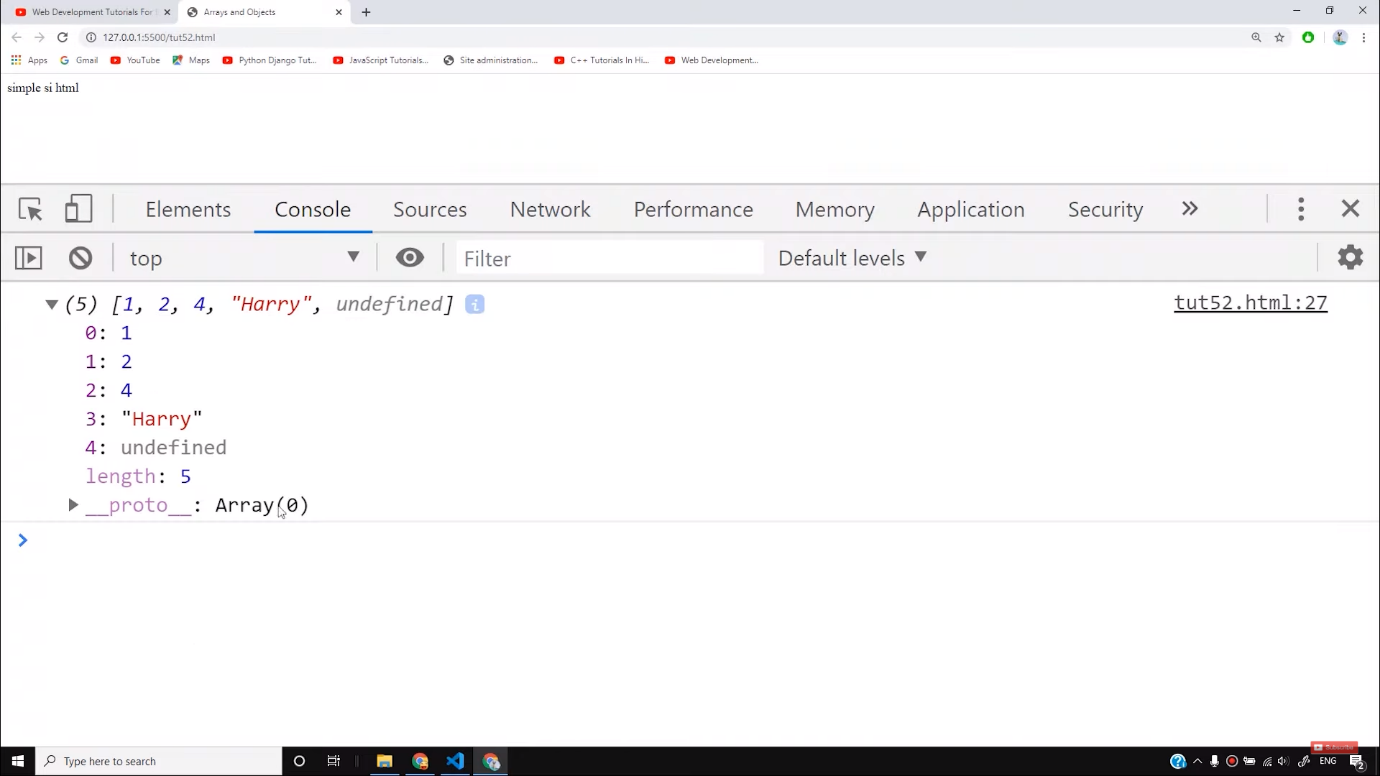
We can declare an array as follows-

let names = [1, 2, 4, "Harry", undefined];

console.log(names);

Copy

An array can contain any type of value in it whether it is a string, integer, or boolean. After writing the above code, you will get the output as follows-



To print the value present at any index number, we can write as follows-

console.log(names[1])

Copy

It will print the value present at index number 1. Arrays are important because they make it easier to iterate through each and every element present in DOM. We can also create arrays with the help of a **new**keyword as follows-

let names = new Array(23);

Copy

In this way, an array will be treated as an object. The new keyword is used to create a new object. To know the length of an array, we can use the **length** function as follows-

console.log(names.length);

Copy

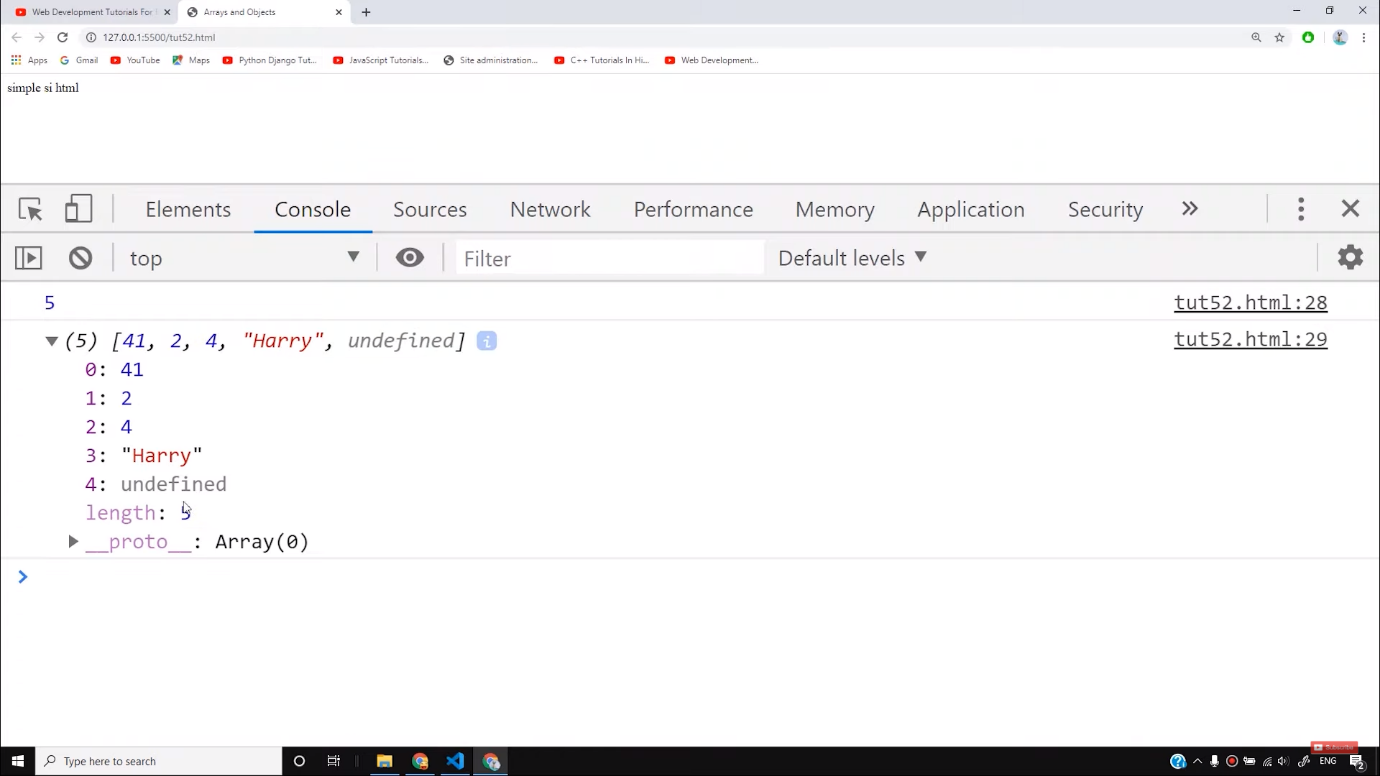
In this example, the output will be 5 because the array *names* contain 5 elements in it.

If we declare a new array as-

let names = new Array(41, 2, 4, "Harry", undefined);

Copy

It will be printed in the same order as follows-



However, to sort this array, we can use *sort()*function as follows-

names = names.sort();

Copy

This will sort all the elements in the array.

So I hope, you have understood the concepts of array and objects very well. Soon we will implement all these to make live websites. Till then keep practicing and learn new things.

**Code as described/written in the video**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Arrays and Objects</title>

</head>

<body>

<div class="container">simple si html</div>

<script>

let myVar = 34;

let myVar2 = "string";

let myVar3 = true;

let myVar4 = null;

let myVar5 = undefined;

// let employee = {

// name: "Harry",

// salary: 10,

// channel: "CodeWithHarry",

// "channel 2": "ProgrammingWithHarry",

// }

// console.log(employee);

// let names = [1, 2, 4, "Harry", undefined];

// let names = new Array(41, 2, 4, "Harry", undefined);

let names = new Array(23);

console.log(names.length);

names = names.sort();

names.push("this is pushed");

console.log(names);

</script>

</body>

</html>

# JavaScript Objects

**Real Life Objects, Properties, and Methods**

In real life, a car is an **object**.

A car has **properties** like weight and color, and **methods** like start and stop:

|  |  |  |
| --- | --- | --- |
| **Object** | **Properties** | **Methods** |
|  | car.name = Fiat  car.model = 500  car.weight = 850kg  car.color = white | car.start()  car.drive()  car.brake()  car.stop() |

All cars have the same **properties**, but the property **values** differ from car to car.

All cars have the same **methods**, but the methods are performed **at different times**.

## **JavaScript Objects**

You have already learned that JavaScript variables are containers for data values.

This code assigns a **simple value** (Fiat) to a **variable** named car:

let car = "Fiat";

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_objects_variable)

Objects are variables too. But objects can contain many values.

This code assigns **many values** (Fiat, 500, white) to a **variable** named car:

const car = {type:"Fiat", model:"500", color:"white"};

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_objects_object)

The values are written as **name:value** pairs (name and value separated by a colon).

It is a common practice to declare objects with the const keyword.

Learn more about using const with objects in the chapter: [JS Const](https://www.w3schools.com/js/js_const.asp).

ADVERTISEMENT

## **Object Definition**

You define (and create) a JavaScript object with an object literal:

### **Example**

const person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_objects_create_1)

Spaces and line breaks are not important. An object definition can span multiple lines:

### **Example**

const person = {  
  firstName: "John",  
  lastName: "Doe",  
  age: 50,  
  eyeColor: "blue"  
};

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_objects_create_2)

## **Object Properties**

The **name:values** pairs in JavaScript objects are called **properties**:

|  |  |
| --- | --- |
| **Property** | **Property Value** |
| firstName | John |
| lastName | Doe |
| age | 50 |
| eyeColor | blue |

## **Accessing Object Properties**

You can access object properties in two ways:

*objectName.propertyName*

or

*objectName["propertyName"]*

### **Example1**

person.lastName;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_objects_properties_1)

### **Example2**

person["lastName"];

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_objects_properties_2)

JavaScript objects are containers for **named values**called properties.

## **Object Methods**

Objects can also have **methods**.

Methods are **actions** that can be performed on objects.

Methods are stored in properties as **function definitions**.

|  |  |
| --- | --- |
| **Property** | **Property Value** |
| firstName | John |
| lastName | Doe |
| age | 50 |
| eyeColor | blue |
| fullName | function() {return this.firstName + " " + this.lastName;} |

A method is a function stored as a property.

### **Example**

const person = {  
  firstName: "John",  
  lastName : "Doe",  
  id       : 5566,  
  fullName : function() {  
    return this.firstName + " " + this.lastName;  
  }  
};

In the example above, this refers to the **person object**.

I.E. **this.firstName** means the **firstName** property of **this**.

I.E. **this.firstName** means the **firstName** property of **person**.

## **What is**this**?**

In JavaScript, the this keyword refers to an **object**.

**Which** object depends on how this is being invoked (used or called).

The this keyword refers to different objects depending on how it is used:

|  |
| --- |
| In an object method, this refers to the **object**. |
| Alone, this refers to the **global object**. |
| In a function, this refers to the **global object**. |
| In a function, in strict mode, this is undefined. |
| In an event, this refers to the **element** that received the event. |
| Methods like call(), apply(), and bind() can refer this to **any object**. |

## **Note**

this is not a variable. It is a keyword. You cannot change the value of this.

## **See Also:**

[The JavaScript **this** Tutorial](https://www.w3schools.com/js/js_this.asp)

## **The this Keyword**

In a function definition, this refers to the "owner" of the function.

In the example above, this is the **person object** that "owns" the fullName function.

In other words, this.firstName means the firstName property of **this object**.

Learn more about this in [The JavaScript this Tutorial](https://www.w3schools.com/js/js_this.asp).

## **Accessing Object Methods**

You access an object method with the following syntax:

*objectName.methodName()*

### **Example**

name = person.fullName();

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_objects_method)

If you access a method **without** the () parentheses, it will return the **function definition**:

### **Example**

name = person.fullName;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_objects_function)

## **Do Not Declare Strings, Numbers, and Booleans as Objects!**

When a JavaScript variable is declared with the keyword "new", the variable is created as an object:

x = new String();        // Declares x as a String object  
y = new Number();        // Declares y as a Number object  
z = new Boolean();       // Declares z as a Boolean object

Avoid String, Number, and Boolean objects. They complicate your code and slow down execution speed.

# JavaScript Arrays

[❮ Previous](https://www.w3schools.com/js/js_number_methods.asp)[Next ❯](https://www.w3schools.com/js/js_array_methods.asp)

An array is a special variable, which can hold more than one value:

const cars = ["Saab", "Volvo", "BMW"];

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array)

## **Why Use Arrays?**

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

let car1 = "Saab";  
let car2 = "Volvo";  
let car3 = "BMW";

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is an array!

An array can hold many values under a single name, and you can access the values by referring to an index number.

## **Creating an Array**

Using an array literal is the easiest way to create a JavaScript Array.

Syntax:

const array\_name = [item1, item2, ...];

It is a common practice to declare arrays with the const keyword.

Learn more about const with arrays in the chapter: [JS Array Const](https://www.w3schools.com/js/js_array_const.asp).

### **Example**

const cars = ["Saab", "Volvo", "BMW"];

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array)

Spaces and line breaks are not important. A declaration can span multiple lines:

### **Example**

const cars = [  
  "Saab",  
  "Volvo",  
  "BMW"  
];

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_newlines)

You can also create an array, and then provide the elements:

### **Example**

const cars = [];  
cars[0]= "Saab";  
cars[1]= "Volvo";  
cars[2]= "BMW";

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_add_elements)

## **Using the JavaScript Keyword new**

The following example also creates an Array, and assigns values to it:

### **Example**

const cars = new Array("Saab", "Volvo", "BMW");

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_new)

The two examples above do exactly the same.

There is no need to use new Array().

For simplicity, readability and execution speed, use the array literal method.

ADVERTISEMENT

## **Accessing Array Elements**

You access an array element by referring to the **index number**:

const cars = ["Saab", "Volvo", "BMW"];  
let car = cars[0];

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_element)

**Note:** Array indexes start with 0.

[0] is the first element. [1] is the second element.

## **Changing an Array Element**

This statement changes the value of the first element in cars:

cars[0] = "Opel";

### **Example**

const cars = ["Saab", "Volvo", "BMW"];  
cars[0] = "Opel";

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_element_change)

## **Access the Full Array**

With JavaScript, the full array can be accessed by referring to the array name:

### **Example**

const cars = ["Saab", "Volvo", "BMW"];  
document.getElementById("demo").innerHTML = cars;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_full)

## **Arrays are Objects**

Arrays are a special type of objects. The typeof operator in JavaScript returns "object" for arrays.

But, JavaScript arrays are best described as arrays.

Arrays use **numbers** to access its "elements". In this example, person[0] returns John:

### **Array:**

const person = ["John", "Doe", 46];

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_array)

Objects use **names** to access its "members". In this example, person.firstName returns John:

### **Object:**

const person = {firstName:"John", lastName:"Doe", age:46};

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_object)

## **Array Elements Can Be Objects**

JavaScript variables can be objects. Arrays are special kinds of objects.

Because of this, you can have variables of different types in the same Array.

You can have objects in an Array. You can have functions in an Array. You can have arrays in an Array:

myArray[0] = Date.now;  
myArray[1] = myFunction;  
myArray[2] = myCars;

## **Array Properties and Methods**

The real strength of JavaScript arrays are the built-in array properties and methods:

cars.length   // Returns the number of elements  
cars.sort()   // Sorts the array

Array methods are covered in the next chapters.

## **The length Property**

The length property of an array returns the length of an array (the number of array elements).

### **Example**

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

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_length)

The length property is always one more than the highest array index.

## **Accessing the First Array Element**

### **Example**

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
let fruit = fruits[0];

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_first)

## **Accessing the Last Array Element**

### **Example**

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
let fruit = fruits[fruits.length - 1];

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_last)

## **Looping Array Elements**

One way to loop through an array, is using a for loop:

### **Example**

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
let fLen = fruits.length;  
  
let text = "<ul>";  
for (let i = 0; i < fLen; i++) {  
  text += "<li>" + fruits[i] + "</li>";  
}  
text += "</ul>";

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_loop)

You can also use the Array.forEach() function:

### **Example**

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
  
let text = "<ul>";  
fruits.forEach(myFunction);  
text += "</ul>";  
  
function myFunction(value) {  
  text += "<li>" + value + "</li>";  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_loop_foreach)

## **Adding Array Elements**

The easiest way to add a new element to an array is using the push() method:

### **Example**

const fruits = ["Banana", "Orange", "Apple"];  
fruits.push("Lemon");  // Adds a new element (Lemon) to fruits

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_add_push)

New element can also be added to an array using the length property:

### **Example**

const fruits = ["Banana", "Orange", "Apple"];  
fruits[fruits.length] = "Lemon";  // Adds "Lemon" to fruits

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_add)

**WARNING !**

Adding elements with high indexes can create undefined "holes" in an array:

### **Example**

const fruits = ["Banana", "Orange", "Apple"];  
fruits[6] = "Lemon";  // Creates undefined "holes" in fruits

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_holes)

## **Associative Arrays**

Many programming languages support arrays with named indexes.

Arrays with named indexes are called associative arrays (or hashes).

JavaScript does **not** support arrays with named indexes.

In JavaScript, **arrays** always use **numbered indexes**.

### **Example**

const person = [];  
person[0] = "John";  
person[1] = "Doe";  
person[2] = 46;  
person.length;    // Will return 3  
person[0];        // Will return "John"

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_associative_1)

**WARNING !!**  
If you use named indexes, JavaScript will redefine the array to an object.

After that, some array methods and properties will produce **incorrect results**.

### **Example:**

const person = [];  
person["firstName"] = "John";  
person["lastName"] = "Doe";  
person["age"] = 46;  
person.length;     // Will return 0  
person[0];         // Will return undefined

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_associative_2)

## **The Difference Between Arrays and Objects**

In JavaScript, **arrays** use **numbered indexes**.

In JavaScript, **objects** use **named indexes**.

Arrays are a special kind of objects, with numbered indexes.

## **When to Use Arrays. When to use Objects.**

* JavaScript does not support associative arrays.
* You should use **objects** when you want the element names to be **strings (text)**.
* You should use **arrays** when you want the element names to be **numbers**.

## **JavaScript new Array()**

JavaScript has a built in array constructor new Array().

But you can safely use [] instead.

These two different statements both create a new empty array named points:

const points = new Array();  
const points = [];

These two different statements both create a new array containing 6 numbers:

const points = new Array(40, 100, 1, 5, 25, 10);  
const points = [40, 100, 1, 5, 25, 10];

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_literal)

The new keyword can produce some unexpected results:

// Create an array with three elements:  
const points = new Array(40, 100, 1);

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_new_array_three)

// Create an array with two elements:  
const points = new Array(40, 100);

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_new_array_two)

// Create an array with one element ???  
const points = new Array(40);

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_new_error)

### **A Common Error**

const points = [40];

is not the same as:

const points = new Array(40);

// Create an array with one element:  
const points = [40];

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_one)

// Create an array with 40 undefined elements:  
const points = new Array(40);

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_new_error2)

## **How to Recognize an Array**

A common question is: How do I know if a variable is an array?

The problem is that the JavaScript operator typeof returns "object":

const fruits = ["Banana", "Orange", "Apple"];  
let type = typeof fruits;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_typeof)

The typeof operator returns object because a JavaScript array is an object.

### **Solution 1:**

To solve this problem ECMAScript 5 (JavaScript 2009) defined a new method Array.isArray():

Array.isArray(fruits);

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_isarray_method)

### **Solution 2:**

The instanceof operator returns true if an object is created by a given constructor:

const fruits = ["Banana", "Orange", "Apple"];  
  
fruits instanceof Array;

# JavaScript Array Methods

[❮ Previous](https://www.w3schools.com/js/js_arrays.asp)[Next ❯](https://www.w3schools.com/js/js_array_sort.asp)

## **Converting Arrays to Strings**

The JavaScript method toString() converts an array to a string of (comma separated) array values.

### **Example**

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
document.getElementById("demo").innerHTML = fruits.toString();

Result:

Banana,Orange,Apple,Mango

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_tostring)

The join() method also joins all array elements into a string.

It behaves just like toString(), but in addition you can specify the separator:

### **Example**

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
document.getElementById("demo").innerHTML = fruits.join(" \* ");

Result:

Banana \* Orange \* Apple \* Mango

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_join)

## **Popping and Pushing**

When you work with arrays, it is easy to remove elements and add new elements.

This is what popping and pushing is:

Popping items **out** of an array, or pushing items **into** an array.

ADVERTISEMENT

## **JavaScript Array pop()**

The pop() method removes the last element from an array:

### **Example**

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

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_pop)

The pop() method returns the value that was "popped out":

### **Example**

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

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_pop_out)

## **JavaScript Array push()**

The push() method adds a new element to an array (at the end):

### **Example**

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

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_push)

The push() method returns the new array length:

### **Example**

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

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_push_length)

## **Shifting Elements**

Shifting is equivalent to popping, but working on the first element instead of the last.

## **JavaScript Array shift()**

The shift() method removes the first array element and "shifts" all other elements to a lower index.

### **Example**

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

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_shift)

The shift() method returns the value that was "shifted out":

### **Example**

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

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_shift_return)

## **JavaScript Array unshift()**

The unshift() method adds a new element to an array (at the beginning), and "unshifts" older elements:

### **Example**

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

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_unshift)

The unshift() method returns the new array length.

### **Example**

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

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_unshift_return)

## **Changing Elements**

Array elements are accessed using their **index number**:

Array **indexes** start with 0:

[0] is the first array element  
[1] is the second  
[2] is the third ...

### **Example**

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits[0] = "Kiwi";

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_change)

## **JavaScript Array length**

The length property provides an easy way to append a new element to an array:

### **Example**

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

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_change_add)

## **JavaScript Array delete()**

### **Warning !**

Array elements can be deleted using the JavaScript operator delete.

Using delete leaves undefined holes in the array.

Use pop() or shift() instead.

### **Example**

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
delete fruits[0];

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_delete)

## **Merging (Concatenating) Arrays**

The concat() method creates a new array by merging (concatenating) existing arrays:

### **Example (Merging Two Arrays)**

const myGirls = ["Cecilie", "Lone"];  
const myBoys = ["Emil", "Tobias", "Linus"];  
  
const myChildren = myGirls.concat(myBoys);

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_concat)

The concat() method does not change the existing arrays. It always returns a new array.

The concat() method can take any number of array arguments:

### **Example (Merging Three Arrays)**

const arr1 = ["Cecilie", "Lone"];  
const arr2 = ["Emil", "Tobias", "Linus"];  
const arr3 = ["Robin", "Morgan"];  
const myChildren = arr1.concat(arr2, arr3);

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_concat2)

The concat() method can also take strings as arguments:

### **Example (Merging an Array with Values)**

const arr1 = ["Emil", "Tobias", "Linus"];  
const myChildren = arr1.concat("Peter");

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_concat3)

## **Splicing and Slicing Arrays**

The splice() method adds new items to an array.

The slice() method slices out a piece of an array.

## **JavaScript Array splice()**

The splice() method can be used to add new items to an array:

### **Example**

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

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_splice)

The first parameter (2) defines the position **where** new elements should be **added** (spliced in).

The second parameter (0) defines **how many** elements should be **removed**.

The rest of the parameters ("Lemon" , "Kiwi") define the new elements to be **added**.

The splice() method returns an array with the deleted items:

### **Example**

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

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_splice_return)

## **Using splice() to Remove Elements**

With clever parameter setting, you can use splice() to remove elements without leaving "holes" in the array:

### **Example**

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.splice(0, 1);

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_remove)

The first parameter (0) defines the position where new elements should be **added** (spliced in).

The second parameter (1) defines **how many** elements should be **removed**.

The rest of the parameters are omitted. No new elements will be added.

## **JavaScript Array slice()**

The slice() method slices out a piece of an array into a new array.

This example slices out a part of an array starting from array element 1 ("Orange"):

### **Example**

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

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_slice1)

## **Note**

The slice() method creates a new array.

The slice() method does not remove any elements from the source array.

This example slices out a part of an array starting from array element 3 ("Apple"):

### **Example**

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

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_slice3)

The slice() method can take two arguments like slice(1, 3).

The method then selects elements from the start argument, and up to (but not including) the end argument.

### **Example**

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

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_slice)

If the end argument is omitted, like in the first examples, the slice() method slices out the rest of the array.

### **Example**

const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];  
const citrus = fruits.slice(2);

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_slice2)

## **Automatic toString()**

JavaScript automatically converts an array to a comma separated string when a primitive value is expected.

This is always the case when you try to output an array.

These two examples will produce the same result:

### **Example**

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
document.getElementById("demo").innerHTML = fruits.toString();

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_tostring)

### **Example**

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
document.getElementById("demo").innerHTML = fruits;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_automatic)

# JavaScript Sorting Arrays

**Sorting an Array**

The sort() method sorts an array alphabetically:

### **Example**

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

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_sort)

## **Reversing an Array**

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

You can use it to sort an array in descending order:

### **Example**

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

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_sort_reverse)

## **Numeric Sort**

By default, the sort() function sorts values as **strings**.

This works well for strings ("Apple" comes before "Banana").

However, if numbers are sorted as strings, "25" is bigger than "100", because "2" is bigger than "1".

Because of this, the sort() method will produce incorrect result when sorting numbers.

You can fix this by providing a **compare function**:

### **Example**

const points = [40, 100, 1, 5, 25, 10];  
points.sort(function(a, b){return a - b});

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_sort2)

Use the same trick to sort an array descending:

### **Example**

const points = [40, 100, 1, 5, 25, 10];  
points.sort(function(a, b){return b - a});

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_sort3)

ADVERTISEMENT

## **The Compare Function**

The purpose of the compare function is to define an alternative sort order.

The compare function should return a negative, zero, or positive value, depending on the arguments:

function(a, b){return a - b}

When the sort() function compares two values, it sends the values to the compare function, and sorts the values according to the returned (negative, zero, positive) value.

If the result is negative a is sorted before b.

If the result is positive b is sorted before a.

If the result is 0 no changes are done with the sort order of the two values.

**Example:**

The compare function compares all the values in the array, two values at a time (a, b).

When comparing 40 and 100, the sort() method calls the compare function(40, 100).

The function calculates 40 - 100 (a - b), and since the result is negative (-60),  the sort function will sort 40 as a value lower than 100.

You can use this code snippet to experiment with numerically and alphabetically sorting:

<button onclick="myFunction1()">Sort Alphabetically</button>  
<button onclick="myFunction2()">Sort Numerically</button>  
  
<p id="demo"></p>  
  
<script>  
const points = [40, 100, 1, 5, 25, 10];  
document.getElementById("demo").innerHTML = points;  
  
function myFunction1() {  
  points.sort();  
  document.getElementById("demo").innerHTML = points;  
}  
  
function myFunction2() {  
  points.sort(function(a, b){return a - b});  
  document.getElementById("demo").innerHTML = points;  
}  
</script>

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_sort_alpha)

## **Sorting an Array in Random Order**

### **Example**

const points = [40, 100, 1, 5, 25, 10];  
points.sort(function(a, b){return 0.5 - Math.random()});

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_sort_random)

## **The Fisher Yates Method**

The above example, array.sort(), is not accurate, it will favor some numbers over the others.

The most popular correct method, is called the Fisher Yates shuffle, and was introduced in data science as early as 1938!

In JavaScript the method can be translated to this:

### **Example**

const points = [40, 100, 1, 5, 25, 10];  
  
for (let i = points.length -1; i > 0; i--) {  
  let j = Math.floor(Math.random() \* i)  
  let k = points[i]  
  points[i] = points[j]  
  points[j] = k  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_sort_random2)

## **Find the Highest (or Lowest) Array Value**

There are no built-in functions for finding the max or min value in an array.

However, after you have sorted an array, you can use the index to obtain the highest and lowest values.

Sorting ascending:

### **Example**

const points = [40, 100, 1, 5, 25, 10];  
points.sort(function(a, b){return a - b});  
// now points[0] contains the lowest value  
// and points[points.length-1] contains the highest value

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_sort_low)

Sorting descending:

### **Example**

const points = [40, 100, 1, 5, 25, 10];  
points.sort(function(a, b){return b - a});  
// now points[0] contains the highest value  
// and points[points.length-1] contains the lowest value

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_sort_high)

Sorting a whole array is a very inefficient method if you only want to find the highest (or lowest) value.

## **Using Math.max() on an Array**

You can use Math.max.apply to find the highest number in an array:

### **Example**

function myArrayMax(arr) {  
  return Math.max.apply(null, arr);  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_sort_math_max)

Math.max.apply(null, [1, 2, 3]) is equivalent to Math.max(1, 2, 3).

## **Using Math.min() on an Array**

You can use Math.min.apply to find the lowest number in an array:

### **Example**

function myArrayMin(arr) {  
  return Math.min.apply(null, arr);  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_sort_math_min)

Math.min.apply(null, [1, 2, 3]) is equivalent to Math.min(1, 2, 3).

## **My Min / Max JavaScript Methods**

The fastest solution is to use a "home made" method.

This function loops through an array comparing each value with the highest value found:

### **Example (Find Max)**

function myArrayMax(arr) {  
  let len = arr.length;  
  let max = -Infinity;  
  while (len--) {  
    if (arr[len] > max) {  
      max = arr[len];  
    }  
  }  
  return max;  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_sort_max)

This function loops through an array comparing each value with the lowest value found:

### **Example (Find Min)**

function myArrayMin(arr) {  
  let len = arr.length;  
  let min = Infinity;  
  while (len--) {  
    if (arr[len] < min) {  
      min = arr[len];  
    }  
  }  
  return min;  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_sort_min)

## **Sorting Object Arrays**

JavaScript arrays often contain objects:

### **Example**

const cars = [  
  {type:"Volvo", year:2016},  
  {type:"Saab", year:2001},  
  {type:"BMW", year:2010}  
];

Even if objects have properties of different data types, the sort() method can be used to sort the array.

The solution is to write a compare function to compare the property values:

### **Example**

cars.sort(function(a, b){return a.year - b.year});

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_sort_object1)

Comparing string properties is a little more complex:

### **Example**

cars.sort(function(a, b){  
  let x = a.type.toLowerCase();  
  let y = b.type.toLowerCase();  
  if (x < y) {return -1;}  
  if (x > y) {return 1;}  
  return 0;  
});

# JavaScript Array Iteration

[❮ Previous](https://www.w3schools.com/js/js_array_sort.asp)[Next ❯](https://www.w3schools.com/js/js_array_const.asp)

Array iteration methods operate on every array item.

## **JavaScript Array forEach()**

The forEach() method calls a function (a callback function) once for each array element.

### **Example**

const numbers = [45, 4, 9, 16, 25];  
let txt = "";  
numbers.forEach(myFunction);  
  
function myFunction(value, index, array) {  
  txt += value + "<br>";  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_foreach)

Note that the function takes 3 arguments:

* The item value
* The item index
* The array itself

The example above uses only the value parameter. The example can be rewritten to:

### **Example**

const numbers = [45, 4, 9, 16, 25];  
let txt = "";  
numbers.forEach(myFunction);  
  
function myFunction(value) {  
  txt += value + "<br>";  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_foreach_2)

## **JavaScript Array map()**

The map() method creates a new array by performing a function on each array element.

The map() method does not execute the function for array elements without values.

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

This example multiplies each array value by 2:

### **Example**

const numbers1 = [45, 4, 9, 16, 25];  
const numbers2 = numbers1.map(myFunction);  
  
function myFunction(value, index, array) {  
  return value \* 2;  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_map)

Note that the function takes 3 arguments:

* The item value
* The item index
* The array itself

When a callback function uses only the value parameter, the index and array parameters can be omitted:

### **Example**

const numbers1 = [45, 4, 9, 16, 25];  
const numbers2 = numbers1.map(myFunction);  
  
function myFunction(value) {  
  return value \* 2;  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_map_2)

ADVERTISEMENT

## **JavaScript Array filter()**

The filter() method creates a new array with array elements that passes a test.

This example creates a new array from elements with a value larger than 18:

### **Example**

const numbers = [45, 4, 9, 16, 25];  
const over18 = numbers.filter(myFunction);  
  
function myFunction(value, index, array) {  
  return value > 18;  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_filter)

Note that the function takes 3 arguments:

* The item value
* The item index
* The array itself

In the example above, the callback function does not use the index and array parameters, so they can be omitted:

### **Example**

const numbers = [45, 4, 9, 16, 25];  
const over18 = numbers.filter(myFunction);  
  
function myFunction(value) {  
  return value > 18;  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_filter_2)

## **JavaScript Array reduce()**

The reduce() method runs a function on each array element to produce (reduce it to) a single value.

The reduce() method works from left-to-right in the array. See also reduceRight().

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

This example finds the sum of all numbers in an array:

### **Example**

const numbers = [45, 4, 9, 16, 25];  
let sum = numbers.reduce(myFunction);  
  
function myFunction(total, value, index, array) {  
  return total + value;  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_reduce)

Note that the function takes 4 arguments:

* The total (the initial value / previously returned value)
* The item value
* The item index
* The array itself

The example above does not use the index and array parameters. It can be rewritten to:

### **Example**

const numbers = [45, 4, 9, 16, 25];  
let sum = numbers.reduce(myFunction);  
  
function myFunction(total, value) {  
  return total + value;  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_reduce_2)

The reduce() method can accept an initial value:

### **Example**

const numbers = [45, 4, 9, 16, 25];  
let sum = numbers.reduce(myFunction, 100);  
  
function myFunction(total, value) {  
  return total + value;  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_reduce_initial)

## **JavaScript Array reduceRight()**

The reduceRight() method runs a function on each array element to produce (reduce it to) a single value.

The reduceRight() works from right-to-left in the array. See also reduce().

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

This example finds the sum of all numbers in an array:

### **Example**

const numbers = [45, 4, 9, 16, 25];  
let sum = numbers.reduceRight(myFunction);  
  
function myFunction(total, value, index, array) {  
  return total + value;  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_reduce_right)

Note that the function takes 4 arguments:

* The total (the initial value / previously returned value)
* The item value
* The item index
* The array itself

The example above does not use the index and array parameters. It can be rewritten to:

### **Example**

const numbers = [45, 4, 9, 16, 25];  
let sum = numbers.reduceRight(myFunction);  
  
function myFunction(total, value) {  
  return total + value;  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_reduce_right_2)

## **JavaScript Array every()**

The every() method check if all array values pass a test.

This example check if all array values are larger than 18:

### **Example**

const numbers = [45, 4, 9, 16, 25];  
let allOver18 = numbers.every(myFunction);  
  
function myFunction(value, index, array) {  
  return value > 18;  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_every)

Note that the function takes 3 arguments:

* The item value
* The item index
* The array itself

When a callback function uses the first parameter only (value), the other parameters can be omitted:

### **Example**

const numbers = [45, 4, 9, 16, 25];  
let allOver18 = numbers.every(myFunction);  
  
function myFunction(value) {  
  return value > 18;  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_every_2)

## **JavaScript Array some()**

The some() method check if some array values pass a test.

This example check if some array values are larger than 18:

### **Example**

const numbers = [45, 4, 9, 16, 25];  
let someOver18 = numbers.some(myFunction);  
  
function myFunction(value, index, array) {  
  return value > 18;  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_some)

Note that the function takes 3 arguments:

* The item value
* The item index
* The array itself

## **JavaScript Array indexOf()**

The indexOf() method searches an array for an element value and returns its position.

**Note:** The first item has position 0, the second item has position 1, and so on.

### **Example**

Search an array for the item "Apple":

const fruits = ["Apple", "Orange", "Apple", "Mango"];  
let position = fruits.indexOf("Apple") + 1;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_indexof)

### **Syntax**

*array*.indexOf(item, start)

|  |  |
| --- | --- |
| item | Required. The item to search for. |
| start | Optional. Where to start the search. Negative values will start at the given position counting from the end, and search to the end. |

Array.indexOf() returns -1 if the item is not found.

If the item is present more than once, it returns the position of the first occurrence.

## **JavaScript Array lastIndexOf()**

Array.lastIndexOf() is the same as Array.indexOf(), but returns the position of the last occurrence of the specified element.

### **Example**

Search an array for the item "Apple":

const fruits = ["Apple", "Orange", "Apple", "Mango"];  
let position = fruits.lastIndexOf("Apple") + 1;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_lastindexof)

### **Syntax**

*array*.lastIndexOf(item, start)

|  |  |
| --- | --- |
| item | Required. The item to search for |
| start | Optional. Where to start the search. Negative values will start at the given position counting from the end, and search to the beginning |

## **JavaScript Array find()**

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

This example finds (returns the value of) the first element that is larger than 18:

### **Example**

const numbers = [4, 9, 16, 25, 29];  
let first = numbers.find(myFunction);  
  
function myFunction(value, index, array) {  
  return value > 18;  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_find)

Note that the function takes 3 arguments:

* The item value
* The item index
* The array itself

## **Browser Support**

find() is an ES6 feature (JavaScript 2015).

It is supported in all modern browsers:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| Chrome | Edge | Firefox | Safari | Opera |
| Yes | Yes | Yes | Yes | Yes |

find() is not supported in Internet Explorer.

## **JavaScript Array findIndex()**

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

This example finds the index of the first element that is larger than 18:

### **Example**

const numbers = [4, 9, 16, 25, 29];  
let first = numbers.findIndex(myFunction);  
  
function myFunction(value, index, array) {  
  return value > 18;  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_find_index)

Note that the function takes 3 arguments:

* The item value
* The item index
* The array itself

## **Browser Support**

findIndex() is an ES6 feature (JavaScript 2015).

It is supported in all modern browsers:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| Chrome | Edge | Firefox | Safari | Opera |
| Yes | Yes | Yes | Yes | Yes |

findIndex() is not supported in Internet Explorer.

## **JavaScript Array.from()**

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

### **Example**

Create an Array from a String:

Array.from("ABCDEFG");

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_from)

## **Browser Support**

from() is an ES6 feature (JavaScript 2015).

It is supported in all modern browsers:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| Chrome | Edge | Firefox | Safari | Opera |
| Yes | Yes | Yes | Yes | Yes |

from() is not supported in Internet Explorer.

## **JavaScript Array Keys()**

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

### **Example**

Create an Array Iterator object, containing the keys of the array:

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

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_keys)

## **Browser Support**

keys() is an ES6 feature (JavaScript 2015).

It is supported in all modern browsers:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| Chrome | Edge | Firefox | Safari | Opera |
| Yes | Yes | Yes | Yes | Yes |

keys() is not supported in Internet Explorer.

## **Array entries()**

### **Example**

Create an Array Iterator, and then iterate over the key/value pairs:

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

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_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.

## **Browser Support**

entries() is an ES6 feature (JavaScript 2015).

It is supported in all modern browsers:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| Chrome | Edge | Firefox | Safari | Opera |
| Yes | Yes | Yes | Yes | Yes |

entries() is not supported in Internet Explorer.

## **JavaScript Array includes()**

ECMAScript 2016 introduced Array.includes() to arrays. This allows us to check if an element is present in an array (including NaN, unlike indexOf).

### **Example**

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
  
fruits.includes("Mango"); // is true

# JavaScript Array Const

[❮ Previous](https://www.w3schools.com/js/js_array_iteration.asp)[Next ❯](https://www.w3schools.com/js/js_dates.asp)

## **ECMAScript 2015 (ES6)**

in 2015, JavaScript introduced an important new keyword: const.

It has become a common practice to declare arrays using const:

### **Example**

const cars = ["Saab", "Volvo", "BMW"];

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_const_array_use)

## **Cannot be Reassigned**

An array declared with const cannot be reassigned:

### **Example**

const cars = ["Saab", "Volvo", "BMW"];  
cars = ["Toyota", "Volvo", "Audi"];    // ERROR

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_const_array_assign)

## **Arrays are Not Constants**

The keyword const is a little misleading.

It does NOT define a constant array. It defines a constant reference to an array.

Because of this, we can still change the elements of a constant array.

## **Elements Can be Reassigned**

You can change the elements of a constant array:

### **Example**

// You can create a constant array:  
const cars = ["Saab", "Volvo", "BMW"];  
  
// You can change an element:  
cars[0] = "Toyota";  
  
// You can add an element:  
cars.push("Audi");

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_const_array)

## **Browser Support**

The const keyword is not supported in Internet Explorer 10 or earlier.

The following table defines the first browser versions with full support for the const keyword:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| Chrome 49 | IE 11 / Edge | Firefox 36 | Safari 10 | Opera 36 |
| Mar, 2016 | Oct, 2013 | Feb, 2015 | Sep, 2016 | Mar, 2016 |

## **Assigned when Declared**

JavaScript const variables must be assigned a value when they are declared:

Meaning: An arrays declared with const must be initialized when it is declared.

Using const without initializing the array is a syntax error:

### **Example**

This will not work:

const cars;  
cars = ["Saab", "Volvo", "BMW"];

Arrays declared with var can be initialized at any time.

You can even use the array before it is declared:

### **Example**

This is OK:

cars = ["Saab", "Volvo", "BMW"];  
var cars;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_hoisting_var)

## **Const Block Scope**

An array declared with const has **Block Scope**.

An array declared in a block is not the same as an array declared outside the block:

### **Example**

const cars = ["Saab", "Volvo", "BMW"];  
// Here cars[0] is "Saab"  
{  
  const cars = ["Toyota", "Volvo", "BMW"];  
  // Here cars[0] is "Toyota"  
}  
// Here cars[0] is **"Saab"**

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_const)

An array declared with var does not have block scope:

### **Example**

var cars = ["Saab", "Volvo", "BMW"];  
// Here cars[0] is "Saab"  
{  
  var cars = ["Toyota", "Volvo", "BMW"];  
  // Here cars[0] is "Toyota"  
}  
// Here cars[0] is **"Toyota"**

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_array_var)

You can learn more about Block Scope in the chapter: [JavaScript Scope](https://www.w3schools.com/js/js_scope.asp).

ADVERTISEMENT

## **Redeclaring Arrays**

Redeclaring an array declared with var is allowed anywhere in a program:

### **Example**

var cars = ["Volvo", "BMW"];   // Allowed  
var cars = ["Toyota", "BMW"];  // Allowed  
cars = ["Volvo", "Saab"];      // Allowed

Redeclaring or reassigning an array to const, in the same scope, or in the same block, is not allowed:

### **Example**

var cars = ["Volvo", "BMW"];     // Allowed  
const cars = ["Volvo", "BMW"];   // Not allowed  
{  
  var cars = ["Volvo", "BMW"];   // Allowed  
  const cars = ["Volvo", "BMW"]; // Not allowed  
}

Redeclaring or reassigning an existing const array, in the same scope, or in the same block, is not allowed:

### **Example**

const cars = ["Volvo", "BMW"];   // Allowed  
const cars = ["Volvo", "BMW"];   // Not allowed  
var cars = ["Volvo", "BMW"];     // Not allowed  
cars = ["Volvo", "BMW"];         // Not allowed  
  
{  
  const cars = ["Volvo", "BMW"]; // Allowed  
  const cars = ["Volvo", "BMW"]; // Not allowed  
  var cars = ["Volvo", "BMW"];   // Not allowed  
  cars = ["Volvo", "BMW"];       // Not allowed  
}

Redeclaring an array with const, in another scope, or in another block, is allowed:

### **Example**

const cars = ["Volvo", "BMW"];   // Allowed  
{  
  const cars = ["Volvo", "BMW"]; // Allowed  
}  
{  
  const cars = ["Volvo", "BMW"]; // Allowed  
}