# **BIND,APPLY & CALL IN JAVASCRIPT**

# 

# **Function.prototype.bind()**

The **bind()** method creates a new function that, when called, has its this keyword set to the provided value, with a given sequence of arguments preceding any provided when the new function is called.

## [**Syntax**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_objects/Function/bind#syntax)

bind(thisArg)

bind(thisArg, arg1)

bind(thisArg, arg1, arg2)

bind(thisArg, arg1, ... , argN)

[Parameters](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_objects/Function/bind#parameters)

**thisArg**

The value to be passed as the this parameter to the target function func when the bound function is called. The value is ignored if the bound function is constructed using the [new](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/new) operator. When using bind to create a function (supplied as a callback) inside a setTimeout, any primitive value passed as thisArg is converted to object. If no arguments are provided to bind , or if the thisArg is null or undefined, the this of the executing scope is treated as the thisArg for the new function.

**arg1, arg2, ...argN Optional**

Arguments to prepend to arguments provided to the bound function when invoking func.

### [Return value](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_objects/Function/bind#return_value)

A copy of the given function with the specified this value, and initial arguments (if provided).

## [**Description**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_objects/Function/bind#description)

The bind() function creates a new **bound function**, which is an *exotic function object* (a term from ECMAScript 2015) that wraps the original function object. Calling the bound function generally results in the execution of its wrapped function.

A bound function has the following internal properties:

**[[BoundTargetFunction]]**

The wrapped function object

**[[BoundThis]]**

The value that is always passed as this value when calling the wrapped function.

**[[BoundArguments]]**

A list of values whose elements are used as the first arguments to any call to the wrapped function.

**[[Call]]**

Executes code associated with this object. Invoked via a function call expression. The arguments to the internal method are a this value and a list containing the arguments passed to the function by a call expression.

When a bound function is called, it calls internal method [[Call]] on [[BoundTargetFunction]], with following arguments Call(boundThis, ...args). Where boundThis is [[BoundThis]], args is [[BoundArguments]], followed by the arguments passed by the function call.

A bound function may also be constructed using the [new](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/new) operator. Doing so acts as though the target function had instead been constructed. The provided this value is ignored, while prepended arguments are provided to the emulated function.

## [**Examples**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_objects/Function/bind#examples)

### [Creating a bound function](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_objects/Function/bind#creating_a_bound_function)

The simplest use of bind() is to make a function that, no matter how it is called, is called with a particular this value.

A common mistake for new JavaScript programmers is to extract a method from an object, then to later call that function and expect it to use the original object as its this (e.g., by using the method in callback-based code).

Without special care, however, the original object is usually lost. Creating a bound function from the function, using the original object, neatly solves this problem:

this.x = 9; // 'this' refers to global 'window' object here in a browser

const module = {

x: 81,

getX: function() { return this.x; }

};

module.getX();

// returns 81

const retrieveX = module.getX;

retrieveX();

// returns 9; the function gets invoked at the global scope

// Create a new function with 'this' bound to module

// New programmers might confuse the

// global variable 'x' with module's property 'x'

const boundGetX = retrieveX.bind(module);

boundGetX();

// returns 81

[Partially applied functions](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_objects/Function/bind#partially_applied_functions)

The next simplest use of bind() is to make a function with pre-specified initial arguments.

These arguments (if any) follow the provided this value and are then inserted at the start of the arguments passed to the target function, followed by whatever arguments are passed to the bound function at the time it is called.

function list() {

return Array.prototype.slice.call(arguments);

}

function addArguments(arg1, arg2) {

return arg1 + arg2;

}

const list1 = list(1, 2, 3);

// [1, 2, 3]

const result1 = addArguments(1, 2);

// 3

// Create a function with a preset leading argument

const leadingThirtysevenList = list.bind(null, 37);

// Create a function with a preset first argument.

const addThirtySeven = addArguments.bind(null, 37);

const list2 = leadingThirtysevenList();

// [37]

const list3 = leadingThirtysevenList(1, 2, 3);

// [37, 1, 2, 3]

const result2 = addThirtySeven(5);

// 37 + 5 = 42

const result3 = addThirtySeven(5, 10);

// 37 + 5 = 42

// (the second argument is ignored)

[With setTimeout()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_objects/Function/bind#with_settimeout)

By default within [window.setTimeout()](https://developer.mozilla.org/en-US/docs/Web/API/WindowOrWorkerGlobalScope/setTimeout), the this keyword will be set to the [window](https://developer.mozilla.org/en-US/docs/Web/API/Window) (or global) object. When working with class methods that require this to refer to class instances, you may explicitly bind this to the callback function, in order to maintain the instance.

function LateBloomer() {

this.petalCount = Math.floor(Math.random() \* 12) + 1;

}

// Declare bloom after a delay of 1 second

LateBloomer.prototype.bloom = function() {

window.setTimeout(this.declare.bind(this), 1000);

};

LateBloomer.prototype.declare = function() {

console.log(`I am a beautiful flower with ${this.petalCount} petals!`);

};

const flower = new LateBloomer();

flower.bloom();

// after 1 second, calls 'flower.declare()'

[Bound functions used as constructors](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_objects/Function/bind#bound_functions_used_as_constructors)

**Warning:** This section demonstrates JavaScript capabilities and documents some edge cases of the bind() method.

The methods shown below are not the best way to do things, and probably should not be used in any production environment.

Bound functions are automatically suitable for use with the [new](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/new) operator to construct new instances created by the target function. When a bound function is used to construct a value, the provided this is ignored.

However, provided arguments are still prepended to the constructor call:

function Point(x, y) {

this.x = x;

this.y = y;

}

Point.prototype.toString = function() {

return `${this.x},${this.y}`;

};

const p = new Point(1, 2);

p.toString();

// '1,2'

// not supported in the polyfill below,

// works fine with native bind:

const YAxisPoint = Point.bind(null, 0/\*x\*/);

const emptyObj = {};

const YAxisPoint = Point.bind(emptyObj, 0/\*x\*/);

const axisPoint = new YAxisPoint(5);

axisPoint.toString(); // '0,5'

axisPoint instanceof Point; // true

axisPoint instanceof YAxisPoint; // true

new YAxisPoint(17, 42) instanceof Point; // true

Note that you need not do anything special to create a bound function for use with [new](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/new).

The corollary is that you need not do anything special to create a bound function to be called plainly, even if you would rather require the bound function to only be called using [new](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/new).

// Example can be run directly in your JavaScript console

// ...continued from above

// Can still be called as a normal function

// (although usually this is undesired)

YAxisPoint(13);

`${emptyObj.x},${emptyObj.y}`;

// > '0,13'

If you wish to support the use of a bound function only using [new](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/new), or only by calling it, the target function must enforce that restriction.

### [Creating shortcuts](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_objects/Function/bind#creating_shortcuts)

bind() is also helpful in cases where you want to create a shortcut to a function which requires a specific this value.

Take [Array.prototype.slice()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/slice), for example, which you want to use for converting an array-like object to a real array. You could create a shortcut like this:

const slice = Array.prototype.slice;

// ...

slice.apply(arguments);

With bind(), this can be simplified.

In the following piece of code, slice() is a bound function to the [apply()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/apply) function of [Function](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function), with the this value set to the [slice()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/slice) function of Array.prototype. This means that additional apply() calls can be eliminated:

// same as "slice" in the previous example

const unboundSlice = Array.prototype.slice;

const slice = Function.prototype.apply.bind(unboundSlice);

// ...

slice(arguments);

[**Specifications**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_objects/Function/bind#specifications)

| **Specification** |
| --- |
| [ECMAScript Language Specification (ECMAScript)](https://tc39.es/ecma262/multipage/fundamental-objects.html#sec-function.prototype.bind)  [# sec-function.prototype.bind](https://tc39.es/ecma262/multipage/fundamental-objects.html#sec-function.prototype.bind) |

## [**Browser compatibility**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_objects/Function/bind#browser_compatibility)

[Report problems with this compatibility data on GitHub](https://github.com/mdn/browser-compat-data/issues/new?body=%3C%21--+Tips%3A+where+applicable%2C+specify+browser+name%2C+browser+version%2C+and+mobile+operating+system+version+--%3E%0A%0A%23%23%23%23+What+information+was+incorrect%2C+unhelpful%2C+or+incomplete%3F%0A%0A%23%23%23%23+What+did+you+expect+to+see%3F%0A%0A%23%23%23%23+Did+you+test+this%3F+If+so%2C+how%3F%0A%0A%0A%3C%21--+Do+not+make+changes+below+this+line+--%3E%0A%3Cdetails%3E%0A%3Csummary%3EMDN+page+report+details%3C%2Fsummary%3E%0A%0A*+Query%3A+%60javascript.builtins.Function.bind%60%0A*+MDN+URL%3A+https%3A%2F%2Fdeveloper.mozilla.org%2Fen-US%2Fdocs%2FWeb%2FJavaScript%2FReference%2FGlobal_objects%2FFunction%2Fbind%0A*+Report+started%3A+2021-08-13T17%3A42%3A23.387Z%0A%0A%3C%2Fdetails%3E&title=javascript.builtins.Function.bind+-+%3CPUT+TITLE+HERE%3E)

|  | **desktop** | | | | | | **mobile** | | | | | | **server** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Chrome | Edge | Firefox | Internet Explorer | Opera | Safari | WebView Android | Chrome Android | Firefox for Android | Opera Android | Safari on iOS | Samsung Internet | Node.js |
| bind | Full support  7 | Full support  12 | Full support  4 | Full support  9 | Full support  11.6 | Full support  5.1 | Full support  4 | Full support  18 | Full support  4 | Full support  12 | Full support  6 | Full support  1.0 | Full support  0.10.0 |

### Legend

**Full support**

Full support

# 

# **Function.prototype.apply()**

The **apply()** method calls a function with a given this value, and arguments provided as an array (or an [array-like object](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Indexed_collections#working_with_array-like_objects)).

## [**Syntax**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/apply#syntax)

apply(thisArg)

apply(thisArg, argsArray)

### [Parameters](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/apply#parameters)

**thisArg**

The value of this provided for the call to func.

Note that this may not be the actual value seen by the method: if the method is a function in [non-strict mode](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Strict_mode) code, [null](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/null) and [undefined](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/undefined) will be replaced with the global object, and primitive values will be boxed. This argument is required.

**argsArray Optional**

An array-like object, specifying the arguments with which func should be called, or [null](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/null) or [undefined](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/undefined) if no arguments should be provided to the function.

Starting with ECMAScript 5 these arguments can be a generic array-like object instead of an array. See below for [browser compatibility](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/apply#browser_compatibility) information.

### [Return value](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/apply#return_value)

The result of calling the function with the specified **this** value and arguments.

## [**Description**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/apply#description)

**Note:** While the syntax of this function is almost identical to that of [call()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/call), the fundamental difference is that call() accepts an **argument list**, while apply() accepts a **single array of arguments**.

**Note:** When the first argument is undefined or null a similar outcome can be achieved using the array [spread syntax](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread_syntax).

You can assign a different this object when calling an existing function. this refers to the current object (the calling object). With apply, you can write a method once, and then inherit it in another object, without having to rewrite the method for the new object.

apply is very similar to [call()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/call), except for the type of arguments it supports. You use an arguments array instead of a list of arguments (parameters). With apply, you can also use an array literal, for example, func.apply(this, ['eat', 'bananas']), or an [Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array) object, for example, func.apply(this, new Array('eat', 'bananas')).

You can also use [arguments](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/arguments) for the argsArray parameter. [arguments](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/arguments) is a local variable of a function. It can be used for all unspecified arguments of the called object. Thus, you do not have to know the arguments of the called object when you use the apply method. You can use arguments to pass all the arguments to the called object. The called object is then responsible for handling the arguments.

Since ECMAScript 5th Edition, you can also use any kind of object which is array-like. In practice, this means it's going to have a length property, and integer ("index") properties in the range (0..length - 1). For example, you could use a [NodeList](https://developer.mozilla.org/en-US/docs/Web/API/NodeList), or a custom object like { 'length': 2, '0': 'eat', '1': 'bananas' }.

**Note:** Many older browsers—including Chrome <17 and Internet Explorer <9—don't accept array-like objects, and will throw an exception.

## [**Examples**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/apply#examples)

### [Using apply to append an array to another](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/apply#using_apply_to_append_an_array_to_another)

You can use push to append an element to an array. And, because push accepts a variable number of arguments, you can also push multiple elements at once.

But, if you pass an array to push, it will actually add that array as a single element, instead of adding the elements individually. So you end up with an array inside an array.

What if that is not what you want? concat does have the desired behavior in this case, but it does not append to the *existing* array—it instead creates and returns a new array.

But you wanted to append to the existing array... So what now? Write a loop? Surely not?

apply to the rescue!

const array = ['a', 'b'];

const elements = [0, 1, 2];

array.push.apply(array, elements);

console.info(array); // ["a", "b", 0, 1, 2]

### [Using apply and built-in functions](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/apply#using_apply_and_built-in_functions)

Clever usage of apply allows you to use built-in functions for some tasks that would probably have otherwise been written by looping over the array values.

As an example, here are Math.max/Math.min, used to find out the maximum/minimum value in an array.

// min/max number in an array

const numbers = [5, 6, 2, 3, 7];

// using Math.min/Math.max apply

let max = Math.max.apply(null, numbers);

// This about equal to Math.max(numbers[0], ...)

// or Math.max(5, 6, ...)

let min = Math.min.apply(null, numbers);

// vs. simple loop based algorithm

max = -Infinity, min = +Infinity;

for (let i = 0; i < numbers.length; i++) {

if (numbers[i] > max) {

max = numbers[i];

}

if (numbers[i] < min) {

min = numbers[i];

}

}

But beware: by using apply this way, you run the risk of exceeding the JavaScript engine's argument length limit. The consequences of applying a function with too many arguments (that is, more than tens of thousands of arguments) varies across engines. (The JavaScriptCore engine has hard-coded [argument limit of 65536](https://bugs.webkit.org/show_bug.cgi?id=80797).

This is because the limit (and indeed, even the nature of any excessively-large-stack behavior) is unspecified. Some engines will throw an exception. More perniciously, others will arbitrarily limit the number of arguments actually passed to the applied function. To illustrate this latter case: if such an engine had a limit of four arguments (actual limits are of course significantly higher), it would be as if the arguments 5, 6, 2, 3 had been passed to apply in the examples above, rather than the full array.

If your value array might grow into the tens of thousands, use a hybrid strategy: apply your function to chunks of the array at a time:

function minOfArray(arr) {

let min = Infinity;

let QUANTUM = 32768;

for (var i = 0, len = arr.length; i < len; i += QUANTUM) {

var submin = Math.min.apply(null,

arr.slice(i, Math.min(i+QUANTUM, len)));

min = Math.min(submin, min);

}

return min;

}

let min = minOfArray([5, 6, 2, 3, 7]);

### [Using apply to chain constructors](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/apply#using_apply_to_chain_constructors)

You can use apply to chain [constructors](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/new) for an object (similar to Java).

In the following example we will create a global [Function](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function) method called construct, which will enable you to use an array-like object with a constructor instead of an arguments list.

Function.prototype.construct = function(aArgs) {

let oNew = Object.create(this.prototype);

this.apply(oNew, aArgs);

return oNew;

};

Example usage:

function MyConstructor() {

for (let nProp = 0; nProp < arguments.length; nProp++) {

this['property' + nProp] = arguments[nProp];

}

}

let myArray = [4, 'Hello world!', false];

let myInstance = MyConstructor.construct(myArray);

console.log(myInstance.property1); // logs 'Hello world!'

console.log(myInstance instanceof MyConstructor); // logs 'true'

console.log(myInstance.constructor); // logs 'MyConstructor'

**Note:** This non-native Function.construct method will not work with some native constructors; like [Date](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date), for example. In these cases you have to use the [Function.prototype.bind](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/bind) method.

For example, imagine having an array like the following, to be used with [Date](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date) constructor: [2012, 11, 4]; in this case you have to write something like: new (Function.prototype.bind.apply(Date, [null].concat([2012, 11, 4])))().

This is not the best way to do things, and probably not to be used in any production environment.

## [**Specifications**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/apply#specifications)

| **Specification** |
| --- |
| [ECMAScript Language Specification (ECMAScript)](https://tc39.es/ecma262/multipage/fundamental-objects.html#sec-function.prototype.apply)  [# sec-function.prototype.apply](https://tc39.es/ecma262/multipage/fundamental-objects.html#sec-function.prototype.apply) |

## [**Browser compatibility**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/apply#browser_compatibility)

[Report problems with this compatibility data on GitHub](https://github.com/mdn/browser-compat-data/issues/new?body=%3C%21--+Tips%3A+where+applicable%2C+specify+browser+name%2C+browser+version%2C+and+mobile+operating+system+version+--%3E%0A%0A%23%23%23%23+What+information+was+incorrect%2C+unhelpful%2C+or+incomplete%3F%0A%0A%23%23%23%23+What+did+you+expect+to+see%3F%0A%0A%23%23%23%23+Did+you+test+this%3F+If+so%2C+how%3F%0A%0A%0A%3C%21--+Do+not+make+changes+below+this+line+--%3E%0A%3Cdetails%3E%0A%3Csummary%3EMDN+page+report+details%3C%2Fsummary%3E%0A%0A*+Query%3A+%60javascript.builtins.Function.apply%60%0A*+MDN+URL%3A+https%3A%2F%2Fdeveloper.mozilla.org%2Fen-US%2Fdocs%2FWeb%2FJavaScript%2FReference%2FGlobal_Objects%2FFunction%2Fapply%0A*+Report+started%3A+2021-08-13T17%3A45%3A54.048Z%0A%0A%3C%2Fdetails%3E&title=javascript.builtins.Function.apply+-+%3CPUT+TITLE+HERE%3E)

|  | **desktop** | | | | | | **mobile** | | | | | | **server** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Chrome | Edge | Firefox | Internet Explorer | Opera | Safari | WebView Android | Chrome Android | Firefox for Android | Opera Android | Safari on iOS | Samsung Internet | Node.js |
| apply | Full support  1 | Full support  12 | Full support  1 | Full support  5.5 | Full support  4 | Full support  1 | Full support  1 | Full support  18 | Full support  4 | Full support  10.1 | Full support  1 | Full support  1.0 | Full support  0.10.0 |
| ES 5.1: generic array-like object as arguments | Full support  17 | Full support  12 | Full support  4 | Full support  9 | Full support  5 | Full support  6 | Full support  37 | Full support  18 | Full support  4 | Full support  10.1 | Full support  6 | Full support  1.0 | Full support  0.10.0 |

### Legend

**Full support**

Full support

# 

# 

# 

# **Function.prototype.call()**

The **call()** method calls a function with a given this value and arguments provided individually.

## [**Syntax**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/call#syntax)

call()

call(thisArg)

call(thisArg, arg1)

call(thisArg, arg1, arg2)

call(thisArg, arg1, ... , argN)

[Parameters](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/call#parameters)

**thisArg Optional**

The value to use as this when calling func.

**Note:** In certain cases, thisArg may not be the actual value seen by the method.

If the method is a function in [non-strict mode](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Strict_mode), [null](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/null) and [undefined](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/undefined) will be replaced with the global object, and primitive values will be converted to objects.

**arg1, arg2, ...argN Optional**

Arguments for the function.

### [Return value](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/call#return_value)

The result of calling the function with the specified **this** value and arguments.

## [**Description**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/call#description)

The call() allows for a function/method belonging to one object to be assigned and called for a different object.

call() provides a new value of this to the function/method. With call(), you can write a method once and then inherit it in another object, without having to rewrite the method for the new object.

**Note:** While the syntax of this function is almost identical to that of [apply()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/apply), the fundamental difference is that call() accepts an **argument list**, while apply() accepts a **single array of arguments**.

## [**Examples**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/call#examples)

### [Using call to chain constructors for an object](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/call#using_call_to_chain_constructors_for_an_object)

You can use call to chain constructors for an object (similar to Java).

In the following example, the constructor for the Product object is defined with two parameters: name and price.

Two other functions, Food and Toy, invoke Product, passing this, name, and price. Product initializes the properties name and price, both specialized functions define the category.

function Product(name, price) {

this.name = name;

this.price = price;

}

function Food(name, price) {

Product.call(this, name, price);

this.category = 'food';

}

function Toy(name, price) {

Product.call(this, name, price);

this.category = 'toy';

}

const cheese = new Food('feta', 5);

const fun = new Toy('robot', 40);

[Using call to invoke an anonymous function](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/call#using_call_to_invoke_an_anonymous_function)

In this example, we create an anonymous function and use call to invoke it on every object in an array.

The main purpose of the anonymous function here is to add a print function to every object, which is able to print the correct index of the object in the array.

**Note:** Passing the object as this value is not strictly necessary, but is done for explanatory purpose.

const animals = [

{ species: 'Lion', name: 'King' },

{ species: 'Whale', name: 'Fail' }

];

for (let i = 0; i < animals.length; i++) {

(function(i) {

this.print = function() {

console.log('#' + i + ' ' + this.species

+ ': ' + this.name);

}

this.print();

}).call(animals[i], i);

}

[Using call to invoke a function and specifying the context for 'this'](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/call#using_call_to_invoke_a_function_and_specifying_the_context_for_this)

In the example below, when we call greet, the value of this will be bound to object obj.

function greet() {

const reply = [this.animal, 'typically sleep between', this.sleepDuration].join(' ');

console.log(reply);

}

const obj = {

animal: 'cats', sleepDuration: '12 and 16 hours'

};

greet.call(obj); // cats typically sleep between 12 and 16 hours

[Using call to invoke a function and without specifying the first argument](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/call#using_call_to_invoke_a_function_and_without_specifying_the_first_argument)

In the example below, we invoke the display function without passing the first argument. If the first argument is not passed, the value of this is bound to the global object.

var sData = 'Wisen';

function display() {

console.log('sData value is %s ', this.sData);

}

display.call(); // sData value is Wisen

**Note:** In strict mode, the value of this will be undefined. See below.

'use strict';

var sData = 'Wisen';

function display() {

console.log('sData value is %s ', this.sData);

}

display.call(); // Cannot read the property of 'sData' of undefined

## [**Specifications**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/call#specifications)

| **Specification** |
| --- |
| [ECMAScript Language Specification (ECMAScript)](https://tc39.es/ecma262/multipage/fundamental-objects.html#sec-function.prototype.call)  [# sec-function.prototype.call](https://tc39.es/ecma262/multipage/fundamental-objects.html#sec-function.prototype.call) |

## [**Browser compatibility**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/call#browser_compatibility)

[Report problems with this compatibility data on GitHub](https://github.com/mdn/browser-compat-data/issues/new?body=%3C%21--+Tips%3A+where+applicable%2C+specify+browser+name%2C+browser+version%2C+and+mobile+operating+system+version+--%3E%0A%0A%23%23%23%23+What+information+was+incorrect%2C+unhelpful%2C+or+incomplete%3F%0A%0A%23%23%23%23+What+did+you+expect+to+see%3F%0A%0A%23%23%23%23+Did+you+test+this%3F+If+so%2C+how%3F%0A%0A%0A%3C%21--+Do+not+make+changes+below+this+line+--%3E%0A%3Cdetails%3E%0A%3Csummary%3EMDN+page+report+details%3C%2Fsummary%3E%0A%0A*+Query%3A+%60javascript.builtins.Function.call%60%0A*+MDN+URL%3A+https%3A%2F%2Fdeveloper.mozilla.org%2Fen-US%2Fdocs%2FWeb%2FJavaScript%2FReference%2FGlobal_Objects%2FFunction%2Fcall%0A*+Report+started%3A+2021-08-13T17%3A47%3A52.437Z%0A%0A%3C%2Fdetails%3E&title=javascript.builtins.Function.call+-+%3CPUT+TITLE+HERE%3E)

|  | **desktop** | | | | | | **mobile** | | | | | | **server** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Chrome | Edge | Firefox | Internet Explorer | Opera | Safari | WebView Android | Chrome Android | Firefox for Android | Opera Android | Safari on iOS | Samsung Internet | Node.js |
| call | Full support  1 | Full support  12 | Full support  1 | Full support  5.5 | Full support  4 | Full support  1 | Full support  1 | Full support  18 | Full support  4 | Full support  10.1 | Full support  1 | Full support  1.0 | Full support  0.10.0 |

### Legend

**Full support**

Full support