


# Function.prototype.call()

Jump to: [Syntax](#) [Description](#) [Examples](#) [Specifications](#) [Browser compatibility](#) [See also](#)

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

 **Note:** While the syntax of this function is almost identical to that of `apply()`, the fundamental difference is that `call()` accepts an **argument list**, while `apply()` accepts a **single array of arguments**.

JavaScript Demo: Function.call()

```
1 function Product(name, price) {
2   this.name = name;
3   this.price = price;
4 }
5
6 function Food(name, price) {
7   Product.call(this, name, price);
8   this.category = 'food';
9 }
10
11 console.log(new Food('cheese', 5).name);
12 // expected output: "cheese"
13
```

Run ›

Reset

## Syntax

```
function.call(thisArg, arg1, arg2, ...)
```

### Parameters

#### thisArg

Optional. The value of `this` provided for the call to a *function*. Note that `this` may not be the actual value seen by the method: if the method is a function in non-strict mode, `null` and `undefined` will be replaced with the global object and primitive values will be converted to objects.

#### arg1, arg2, ...

Optional. Arguments for the function.

### Return value

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

# Description

A different `this` object can be assigned when calling an existing function. `this` refers to the current object, the calling object. 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.

---

## Examples

### 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` and `name` and `price`. `Product` initializes the properties `name` and `price`, both specialized functions define the `category`.

```
1 function Product(name, price) {
2   this.name = name;
3   this.price = price;
4 }
5
6 function Food(name, price) {
7   Product.call(this, name, price);
8   this.category = 'food';
9 }
10
11 function Toy(name, price) {
12   Product.call(this, name, price);
13   this.category = 'toy';
14 }
15
16 var cheese = new Food('feta', 5);
17 var fun = new Toy('robot', 40);
```

### Using `call` to invoke an anonymous function

In this purely constructed 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 right index of the object in the array. Passing the object as `this` value was not strictly necessary, but is done for explanatory purpose.

```
1 var animals = [
2   { species: 'Lion', name: 'King' },
3   { species: 'Whale', name: 'Fail' }
4 ];
5
6 for (var i = 0; i < animals.length; i++) {
7   (function(i) {
8     this.print = function() {
9       console.log('#' + i + ' ' + this.species
10         + ': ' + this.name);
11     }
12     this.print();
13   }).call(animals[i], i);
14 }
```

### 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`.

```
1 function greet() {
2   var reply = [this.animal, 'typically sleep between', this.sleepDuration].join(' ');
3   console.log(reply);
4 }
5
6 var obj = {
7   animal: 'cats', sleepDuration: '12 and 16 hours'
8 };
9
10 greet.call(obj);
```









```
9 |
10 | greet.call(obj); // cats typically sleep between 12 and 16 hours
```

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.




















```
1 | var sData = 'Wisen';
2 |
3 | function display(){
4 |   console.log('sData value is %s ', this.sData);
5 | }
6 |
7 | display.call(); // sData value is Wisen
```

## Specifications

Specification	Status	Comment
 ECMAScript 1st Edition (ECMA-262)	 <b>ST</b> Standard	Initial definition. Implemented in JavaScript 1.3.
 ECMAScript 5.1 (ECMA-262) The definition of 'Function.prototype.call' in that specification.	 <b>ST</b> Standard	
 ECMAScript 2015 (6th Edition, ECMA-262) The definition of 'Function.prototype.call' in that specification.	 <b>ST</b> Standard	
 ECMAScript Latest Draft (ECMA-262) The definition of 'Function.prototype.call' in that specification.	 <b>D</b> Draft	

## Browser compatibility

New compatibility tables are in beta ▾

														
														
Basic support														
Yes	Yes	1	Yes	Yes	Yes	Yes	Yes	Yes	4	Yes	Yes	Yes	Yes	Yes
	Full support													

## See also

- `Function.prototype.bind()`
- `Function.prototype.apply()`
- Introduction to Object-Oriented JavaScript