

# JavaScript Function Invocation

[< Previous](#)[Next >](#)

The code inside a JavaScript function will execute when "something" invokes it.

## Invoking a JavaScript Function

The code inside a function is not executed when the function is **defined**.

The code inside a function is executed when the function is **invoked**.

It is common to use the term "**call a function**" instead of "**invoke a function**".

It is also common to say "call upon a function", "start a function", or "execute a function".

In this tutorial, we will use **invoke**, because a JavaScript function can be invoked without being called.

## Invoking a Function as a Function

### Example

```
function myFunction(a, b) {  
    return a * b;  
}  
myFunction(10, 2);           // Will return 20
```

[Try it Yourself »](#)

The function above does not belong to any object. But in JavaScript there is always a default global object.

In HTML the default global object is the HTML page itself, so the function above "belongs" to the HTML page.

In a browser the page object is the browser window. The function above automatically becomes a window function.

myFunction() and window.myFunction() is the same function:

## Example

```
function myFunction(a, b) {  
    return a * b;  
}  
window.myFunction(10, 2);    // Will also return 20
```

[Try it Yourself »](#)

This is a common way to invoke a JavaScript function, but not a very good practice. Global variables, methods, or functions can easily create name conflicts and bugs in the global object.

## The ***this*** Keyword

In JavaScript, the thing called **this**, is the object that "owns" the current code.

The value of this, when used in a function, is the object that "owns" the function.

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

## The Global Object

When a function is called without an owner object, the value of **this** becomes the global object.

In a web browser the global object is the browser window.

This example returns the window object as the value of **this**:

## Example

```
var x = myFunction();           // x will be the window object

function myFunction() {
    return this;
}
```

[Try it Yourself »](#)

Invoking a function as a global function, causes the value of **this** to be the global object.

Using the window object as a variable can easily crash your program.

## Invoking a Function as a Method

In JavaScript you can define functions as object methods.

The following example creates an object (**myObject**), with two properties (**firstName** and **lastName**), and a method (**fullName**):

## Example

```
var myObject = {
    firstName: "John",
    lastName: "Doe",
    fullName: function () {
        return this.firstName + " " + this.lastName;
    }
}

myObject.fullName();           // Will return "John Doe"
```

[Try it Yourself »](#)

The **fullName** method is a function. The function belongs to the object. **myObject** is the owner of the function.

The thing called **this**, is the object that "owns" the JavaScript code. In this case the value of **this** is **myObject**.

Test it! Change the **fullName** method to return the value of **this**:

## Example

```
var myObject = {
  firstName: "John",
  lastName: "Doe",
  fullName: function () {
    return this;
  }
}
myObject.fullName();           // Will return [object Object] (the owner
                                object)
```

[Try it Yourself »](#)

Invoking a function as an object method, causes the value of **this** to be the object itself.

## Invoking a Function with a Function Constructor

If a function invocation is preceded with the **new** keyword, it is a constructor invocation.

It looks like you create a new function, but since JavaScript functions are objects you actually create a new object:

## Example

```
// This is a function constructor:
function myFunction(arg1, arg2) {
  this.firstName = arg1;
  this.lastName  = arg2;
```

```
}  
  
// This creates a new object  
var x = new myFunction("John", "Doe");  
x.firstName; // Will return "John"
```

Try it Yourself »

A constructor invocation creates a new object. The new object inherits the properties and methods from its constructor.

The **this** keyword in the constructor does not have a value.  
The value of **this** will be the new object created when the function is invoked.

[< Previous](#)

[Next >](#)

Copyright 1999-2018 by Refsnes Data. All Rights Reserved.