

JavaScript Objects

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

In JavaScript, objects are king. If you understand objects, you understand JavaScript.

In JavaScript, almost "everything" is an object.

* Booleans can be objects (if defined with the new keyword)
* Numbers can be objects (if defined with the new keyword)
* Strings can be objects (if defined with the new keyword)
* Dates are always objects
* Maths are always objects
* Regular expressions are always objects
* Arrays are always objects
* Functions are always objects
* Objects are always objects

All JavaScript values, except primitives, are objects.

JavaScript Primitives

A **primitive value** is a value that has no properties or methods.

**3.14** is a primitive value

A **primitive data type** is data that has a primitive value.

JavaScript defines 7 types of primitive data types:

Examples

* string
* number
* boolean
* null
* undefined
* symbol
* bigint

Immutable

Primitive values are immutable (they are hardcoded and cannot be changed).

if x = 3.14, you can change the value of x, but you cannot change the value of 3.14.

|  |  |  |
| --- | --- | --- |
| **Value** | **Type** | **Comment** |
| "Hello" | string | "Hello" is always "Hello" |
| 3.14 | number | 3.14 is always 3.14 |
| true | boolean | true is always true |
| false | boolean | false is always false |
| null | null (object) | null is always null |
| undefined | undefined | undefined is always undefined |

Objects are Variables

JavaScript variables can contain single values:

Example

let person = "John Doe";

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

JavaScript variables can also contain many values.

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

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

Example

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

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

A JavaScript object is a collection of **named values**

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

Example

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

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

ADVERTISEMENT

Object Properties

The named values, in JavaScript objects, are called **properties**.

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

Objects written as name value pairs are similar to:

* Associative arrays in PHP
* Dictionaries in Python
* Hash tables in C
* Hash maps in Java
* Hashes in Ruby and Perl

Object Methods

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

Object properties can be both primitive values, other objects, and functions.

An **object method** is an object property containing a **function definition**.

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

JavaScript objects are containers for named values, called properties and methods.

You will learn more about methods in the next chapters.

Creating a JavaScript Object

With JavaScript, you can define and create your own objects.

There are different ways to create new objects:

* Create a single object, using an object literal.
* Create a single object, with the keyword new.
* Define an object constructor, and then create objects of the constructed type.
* Create an object using Object.create().

Using an Object Literal

This is the easiest way to create a JavaScript Object.

Using an object literal, you both define and create an object in one statement.

An object literal is a list of name:value pairs (like age:50) inside curly braces {}.

The following example creates a new JavaScript object with four properties:

Example

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

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_object_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_object_create_2)

This example creates an empty JavaScript object, and then adds 4 properties:

Example

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

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

Using the JavaScript Keyword new

The following example create a new JavaScript object using new Object(), and then adds 4 properties:

Example

const person = new Object();  
person.firstName = "John";  
person.lastName = "Doe";  
person.age = 50;  
person.eyeColor = "blue";

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

The examples above do exactly the same.

But there is no need to use new Object().

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

JavaScript Objects are Mutable

Objects are mutable: They are addressed by reference, not by value.

If person is an object, the following statement will not create a copy of person:

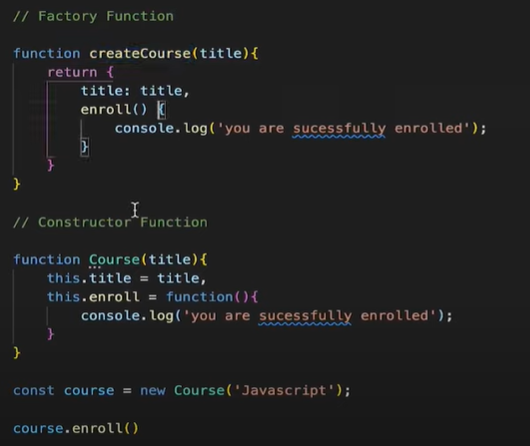
const x = person;  // Will not create a copy of person.

The object x is **not a copy** of person. It **is** person. Both x and person are the same object.

Any changes to x will also change person, because x and person are the same object.

Example

const person = {  
  firstName:"John",  
  lastName:"Doe",  
  age:50, eyeColor:"blue"  
}  
  
const x = person;  
x.age = 10;      // Will change both x.age and person.age



# JavaScript Object Properties

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

Properties are the most important part of any JavaScript object.

## JavaScript Properties

Properties are the values associated with a JavaScript object.

A JavaScript object is a collection of unordered properties.

Properties can usually be changed, added, and deleted, but some are read only.

## Accessing JavaScript Properties

The syntax for accessing the property of an object is:

*objectName.property*// person.age

or

*objectName*["*property*"]   // person["age"]

or

*objectName*[*expression*]   // x = "age"; person[x]

The expression must evaluate to a property name.

### Example 1

person.firstname + " is " + person.age + " years old.";

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_object_properties1)

### Example 2

person["firstname"] + " is " + person["age"] + " years old.";

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_object_properties2)

## JavaScript for...in Loop

The JavaScript for...in statement loops through the properties of an object.

### Syntax

for (let *variable* in *object*) {  
*// code to be executed*  
}

The block of code inside of the for...in loop will be executed once for each property.

Looping through the properties of an object:

### Example

const person = {  
  fname:" John",  
  lname:" Doe",  
  age: 25  
};  
  
for (let x in person) {  
  txt += person[x];  
}

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

## Adding New Properties

You can add new properties to an existing object by simply giving it a value.

Assume that the person object already exists - you can then give it new properties:

### Example

person.nationality = "English";

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_object_properties3)

## Deleting Properties

The delete keyword deletes a property from an object:

### Example

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

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_object_properties4)

or delete person["age"];

### Example

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

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_object_properties5)

The delete keyword deletes both the value of the property and the property itself.

After deletion, the property cannot be used before it is added back again.

The delete operator is designed to be used on object properties. It has no effect on variables or functions.

The delete operator should not be used on predefined JavaScript object properties. It can crash your application.

## Nested Objects

Values in an object can be another object:

### Example

myObj = {  
  name:"John",  
  age:30,  
  cars: {  
    car1:"Ford",  
    car2:"BMW",  
    car3:"Fiat"  
  }  
}

You can access nested objects using the dot notation or the bracket notation:

### Example

myObj.cars.car2;

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

or:

### Example

myObj.cars["car2"];

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_object_nested2)

or:

### Example

myObj["cars"]["car2"];

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_object_nested3)

or:

### Example

let p1 = "cars";  
let p2 = "car2";  
myObj[p1][p2];

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_object_nested4)

## Nested Arrays and Objects

Values in objects can be arrays, and values in arrays can be objects:

### Example

const myObj = {  
  name: "John",  
  age: 30,  
  cars: [  
    {name:"Ford", models:["Fiesta", "Focus", "Mustang"]},  
    {name:"BMW", models:["320", "X3", "X5"]},  
    {name:"Fiat", models:["500", "Panda"]}  
  ]  
}

To access arrays inside arrays, use a for-in loop for each array:

### Example

for (let i in myObj.cars) {  
  x += "<h1>" + myObj.cars[i].name + "</h1>";  
  for (let j in myObj.cars[i].models) {  
    x += myObj.cars[i].models[j];  
  }  
}

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

## Property Attributes

All properties have a name. In addition they also have a value.

The value is one of the property's attributes.

Other attributes are: enumerable, configurable, and writable.

These attributes define how the property can be accessed (is it readable?, is it writable?)

In JavaScript, all attributes can be read, but only the value attribute can be changed (and only if the property is writable).

( ECMAScript 5 has methods for both getting and setting all property attributes)

# JavaScript Object Methods

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

### Example

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

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

## 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)

## JavaScript Methods

JavaScript methods are actions that can be performed on objects.

A JavaScript **method** is a property containing a **function definition**.

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

Methods are functions stored as object properties.

## Accessing Object Methods

You access an object method with the following syntax:

*objectName.methodName()*

You will typically describe fullName() as a method of the person object, and fullName as a property.

The fullName property will execute (as a function) when it is invoked with ().

This example accesses the fullName() **method** of a person object:

### Example

name = person.fullName();

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

If you access the fullName **property**, without (), it will return the **function definition**:

### Example

name = person.fullName;

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

ADVERTISEMENT

## Adding a Method to an Object

Adding a new method to an object is easy:

### Example

person.name = function () {  
  return this.firstName + " " + this.lastName;  
};

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

## Using Built-In Methods

This example uses the toUpperCase() method of the String object, to convert a text to uppercase:

let message = "Hello world!";  
let x = message.toUpperCase();

The value of x, after execution of the code above will be:

HELLO WORLD!

### Example

person.name = function () {  
  return (this.firstName + " " + this.lastName).toUpperCase();  
};

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

JavaScript Display Objects

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

How to Display JavaScript Objects?

Displaying a JavaScript object will output **[object Object]**.

Example

const person = {  
  name: "John",  
  age: 30,  
  city: "New York"  
};  
  
document.getElementById("demo").innerHTML = person;

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

Some common solutions to display JavaScript objects are:

* Displaying the Object Properties by name
* Displaying the Object Properties in a Loop
* Displaying the Object using Object.values()
* Displaying the Object using JSON.stringify()

Displaying Object Properties

The properties of an object can be displayed as a string:

Example

const person = {  
  name: "John",  
  age: 30,  
  city: "New York"  
};  
  
document.getElementById("demo").innerHTML =  
person.name + "," + person.age + "," + person.city;

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

Displaying the Object in a Loop

The properties of an object can be collected in a loop:

Example

const person = {  
  name: "John",  
  age: 30,  
  city: "New York"  
};  
  
let txt = "";  
for (let x in person) {  
txt += person[x] + " ";  
};  
  
document.getElementById("demo").innerHTML = txt;

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

You must use **person[x]** in the loop.

**person.x** will not work (Because **x** is a variable).

Using Object.values()

Any JavaScript object can be converted to an array using Object.values():

const person = {  
  name: "John",  
  age: 30,  
  city: "New York"  
};  
  
const myArray = Object.values(person);

myArray is now a JavaScript array, ready to be displayed:

Example

const person = {  
  name: "John",  
  age: 30,  
  city: "New York"  
};  
  
const myArray = Object.values(person);  
document.getElementById("demo").innerHTML = myArray;

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

Object.values() is supported in all major browsers since 2016.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| 54 (2016) | 14 (2016) | 47 (2016) | 10 (2016) | 41 (2016) |

ADVERTISEMENT

Using JSON.stringify()

Any JavaScript object can be stringified (converted to a string) with the JavaScript function JSON.stringify():

const person = {  
  name: "John",  
  age: 30,  
  city: "New York"  
};  
  
let myString = JSON.stringify(person);

myString is now a JavaScript string, ready to be displayed:

Example

const person = {  
  name: "John",  
  age: 30,  
  city: "New York"  
};  
  
let myString = JSON.stringify(person);  
document.getElementById("demo").innerHTML = myString;

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

The result will be a string following the JSON notation:

{"name":"John","age":50,"city":"New York"}

JSON.stringify() is included in JavaScript and supported in all major browsers.

Stringify Dates

JSON.stringify converts dates into strings:

Example

const person = {  
  name: "John",  
  today: new Date()  
};  
  
let myString = JSON.stringify(person);  
document.getElementById("demo").innerHTML = myString;

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

Stringify Functions

JSON.stringify will not stringify functions:

Example

const person = {  
  name: "John",  
  age: function () {return 30;}  
};  
  
let myString = JSON.stringify(person);  
document.getElementById("demo").innerHTML = myString;

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

This can be "fixed" if you convert the functions into strings before stringifying.

Example

const person = {  
  name: "John",  
  age: function () {return 30;}  
};  
person.age = person.age.toString();  
  
let myString = JSON.stringify(person);  
document.getElementById("demo").innerHTML = myString;

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

Stringify Arrays

It is also possible to stringify JavaScript arrays:

Example

const arr = ["John", "Peter", "Sally", "Jane"];  
  
let myString = JSON.stringify(arr);  
document.getElementById("demo").innerHTML = myString;

JavaScript Object Accessors

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

JavaScript Accessors (Getters and Setters)

ECMAScript 5 (ES5 2009) introduced Getter and Setters.

Getters and setters allow you to define Object Accessors (Computed Properties).

JavaScript Getter (The get Keyword)

This example uses a lang property to get the value of the language property.

Example

// Create an object:  
const person = {  
  firstName: "John",  
  lastName: "Doe",  
  language: "en",  
  get lang() {  
    return this.language;  
  }  
};  
  
// Display data from the object using a getter:  
document.getElementById("demo").innerHTML = person.lang;

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

JavaScript Setter (The set Keyword)

This example uses a lang property to set the value of the language property.

Example

const person = {  
  firstName: "John",  
  lastName: "Doe",  
  language: "",  
  set lang(lang) {  
    this.language = lang;  
  }  
};  
  
// Set an object property using a setter:  
person.lang = "en";  
  
// Display data from the object:  
document.getElementById("demo").innerHTML = person.language;

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

ADVERTISEMENT

JavaScript Function or Getter?

What is the differences between these two examples?

Example 1

const person = {  
  firstName: "John",  
  lastName: "Doe",  
  fullName: function() {  
    return this.firstName + " " + this.lastName;  
  }  
};  
  
// Display data from the object using a method:  
document.getElementById("demo").innerHTML = person.fullName();

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

Example 2

const person = {  
  firstName: "John",  
  lastName: "Doe",  
  get fullName() {  
    return this.firstName + " " + this.lastName;  
  }  
};  
  
// Display data from the object using a getter:  
document.getElementById("demo").innerHTML = person.fullName;

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

Example 1 access fullName as a function: person.fullName().

Example 2 access fullName as a property: person.fullName.

The second example provides a simpler syntax.

Data Quality

JavaScript can secure better data quality when using getters and setters.

Using the lang property, in this example, returns the value of the language property in upper case:

Example

// Create an object:  
const person = {  
  firstName: "John",  
  lastName: "Doe",  
  language: "en",  
  get lang() {  
    return this.language.toUpperCase();  
  }  
};  
  
// Display data from the object using a getter:  
document.getElementById("demo").innerHTML = person.lang;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_object_accessors_get2)

Using the lang property, in this example, stores an upper case value in the language property:

Example

const person = {  
  firstName: "John",  
  lastName: "Doe",  
  language: "",  
  set lang(lang) {  
    this.language = lang.toUpperCase();  
  }  
};  
  
// Set an object property using a setter:  
person.lang = "en";  
  
// Display data from the object:  
document.getElementById("demo").innerHTML = person.language;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_object_accessors_set2)

Why Using Getters and Setters?

* It gives simpler syntax
* It allows equal syntax for properties and methods
* It can secure better data quality
* It is useful for doing things behind-the-scenes

Object.defineProperty()

The Object.defineProperty() method can also be used to add Getters and Setters:

A Counter Example

// Define object  
const obj = {counter : 0};  
  
// Define setters and getters  
Object.defineProperty(obj, "reset", {  
  get : function () {this.counter = 0;}  
});  
Object.defineProperty(obj, "increment", {  
  get : function () {this.counter++;}  
});  
Object.defineProperty(obj, "decrement", {  
  get : function () {this.counter--;}  
});  
Object.defineProperty(obj, "add", {  
  set : function (value) {this.counter += value;}  
});  
Object.defineProperty(obj, "subtract", {  
  set : function (value) {this.counter -= value;}  
});  
  
// Play with the counter:  
obj.reset;  
obj.add = 5;  
obj.subtract = 1;  
obj.increment;  
obj.decrement;

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_object_accessors_set4)

# JavaScript Object Constructors

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

### Example

function Person(first, last, age, eye) {  
  this.firstName = first;  
  this.lastName = last;  
  this.age = age;  
  this.eyeColor = eye;  
}

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

## Notes

It is considered good practice to name constructor functions with an upper-case first letter.

## About this

In a constructor function this does not have a value. It is a substitute for the new object. The value of this will become the new object when a new object is created.

## See Also:

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

## Object Types (Blueprints) (Classes)

The examples from the previous chapters are limited. They only create single objects.

Sometimes we need a "**blueprint**" for creating many objects of the same "type".

The way to create an "object type", is to use an **object constructor function**.

In the example above, function Person() is an object constructor function.

Objects of the same type are created by calling the constructor function with the new keyword:

const myFather = new Person("John", "Doe", 50, "blue");  
const myMother = new Person("Sally", "Rally", 48, "green");

[Try it yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_object_constructor1)

ADVERTISEMENT

## 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)

## Adding a Property to an Object

Adding a new property to an existing object is easy:

### Example

myFather.nationality = "English";

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_object_constructor2)

The property will be added to myFather. Not to myMother. (Not to any other person objects).

## Adding a Method to an Object

Adding a new method to an existing object is easy:

### Example

myFather.name = function () {  
  return this.firstName + " " + this.lastName;  
};

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_object_constructor3)

The method will be added to myFather. Not to myMother. (Not to any other person objects).

## Adding a Property to a Constructor

You cannot add a new property to an object constructor the same way you add a new property to an existing object:

### Example

Person.nationality = "English";

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_object_constructor4)

To add a new property to a constructor, you must add it to the constructor function:

### Example

function Person(first, last, age, eyecolor) {  
  this.firstName = first;  
  this.lastName = last;  
  this.age = age;  
  this.eyeColor = eyecolor;  
  this.nationality = "English";  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_object_constructor5)

This way object properties can have default values.

## Adding a Method to a Constructor

Your constructor function can also define methods:

### Example

function Person(first, last, age, eyecolor) {  
  this.firstName = first;  
  this.lastName = last;  
  this.age = age;  
  this.eyeColor = eyecolor;  
  this.name = function() {  
    return this.firstName + " " + this.lastName;  
  };  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_object_constructor6)

You cannot add a new method to an object constructor the same way you add a new method to an existing object.

Adding methods to an object constructor must be done inside the constructor function:

### Example

function Person(firstName, lastName, age, eyeColor) {  
  this.firstName = firstName;   
  this.lastName = lastName;  
  this.age = age;  
  this.eyeColor = eyeColor;  
  this.changeName = function (name) {  
    this.lastName = name;  
  };  
}

The changeName() function assigns the value of name to the person's lastName property.

### Now You Can Try:

myMother.changeName("Doe");

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_object_constructor7)

JavaScript knows which person you are talking about by "substituting" **this** with **myMother**.

## Built-in JavaScript Constructors

JavaScript has built-in constructors for native objects:

new String()    // A new String object  
new Number()    // A new Number object  
new Boolean()   // A new Boolean object  
new Object()    // A new Object object  
new Array()     // A new Array object  
new RegExp()    // A new RegExp object  
new Function()  // A new Function object  
new Date()      // A new Date object

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

The Math() object is not in the list. Math is a global object. The new keyword cannot be used on Math.

## Did You Know?

As you can see above, JavaScript has object versions of the primitive data types String, Number, and Boolean. But there is no reason to create complex objects. Primitive values are much faster:

Use string literals "" instead of new String().

Use number literals 50 instead of new Number().

Use boolean literals true / false instead of new Boolean().

Use object literals {} instead of new Object().

Use array literals [] instead of new Array().

Use pattern literals /()/ instead of new RegExp().

Use function expressions () {} instead of new Function().

### Example

let x1 = "";             // new primitive string  
let x2 = 0;              // new primitive number  
let x3 = false;          // new primitive boolean  
  
const x4 = {};           // new Object object  
const x5 = [];           // new Array object  
const x6 = /()/          // new RegExp object  
const x7 = function(){}; // new function

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_object_constructor_builtin1)

## String Objects

Normally, strings are created as primitives: firstName = "John"

But strings can also be created as objects using the new keyword:  
firstName = new String("John")

Learn why strings should not be created as object in the chapter [JS Strings](https://www.w3schools.com/js/js_strings.asp).

## Number Objects

Normally, numbers are created as primitives: x = 30

But numbers can also be created as objects using the new keyword:  
x = new Number(30)

Learn why numbers should not be created as object in the chapter [JS Numbers](https://www.w3schools.com/js/js_numbers.asp).

## Boolean Objects

Normally, booleans are created as primitives: x = false

But booleans can also be created as objects using the new keyword:  
x = new Boolean(false)

JavaScript Object Prototypes

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

All JavaScript objects inherit properties and methods from a prototype.

In the previous chapter we learned how to use an **object constructor**:

Example

function Person(first, last, age, eyecolor) {  
  this.firstName = first;  
  this.lastName = last;  
  this.age = age;  
  this.eyeColor = eyecolor;  
}  
  
const myFather = new Person("John", "Doe", 50, "blue");  
const myMother = new Person("Sally", "Rally", 48, "green");

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_object_prototype1)

We also learned that you can **not** add a new property to an existing object constructor:

Example

Person.nationality = "English";

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_object_prototype3)

To add a new property to a constructor, you must add it to the constructor function:

Example

function Person(first, last, age, eyecolor) {  
  this.firstName = first;  
  this.lastName = last;  
  this.age = age;  
  this.eyeColor = eyecolor;  
  this.nationality = "English";  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_object_prototype4)

ADVERTISEMENT

Prototype Inheritance

All JavaScript objects inherit properties and methods from a prototype:

* Date objects inherit from Date.prototype
* Array objects inherit from Array.prototype
* Person objects inherit from Person.prototype

The Object.prototype is on the top of the prototype inheritance chain:

Date objects, Array objects, and Person objects inherit from Object.prototype.

Adding Properties and Methods to Objects

Sometimes you want to add new properties (or methods) to all existing objects of a given type.

Sometimes you want to add new properties (or methods) to an object constructor.

Using the **prototype** Property

The JavaScript prototype property allows you to add new properties to object constructors:

Example

function Person(first, last, age, eyecolor) {  
  this.firstName = first;  
  this.lastName = last;  
  this.age = age;  
  this.eyeColor = eyecolor;  
}  
  
Person.prototype.nationality = "English";

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_object_prototype5)

The JavaScript prototype property also allows you to add new methods to objects constructors:

Example

function Person(first, last, age, eyecolor) {  
  this.firstName = first;  
  this.lastName = last;  
  this.age = age;  
  this.eyeColor = eyecolor;  
}  
  
Person.prototype.name = function() {  
  return this.firstName + " " + this.lastName;  
};

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_object_prototype6)

JavaScript Iterables

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

Iterable objects are objects that can be iterated over with for..of.

Technically, iterables must implement the Symbol.iterator method.

Iterating Over a String

You can use a for..of loop to iterate over the elements of a string:

Example

for (const x of "W3Schools") {  
  // code block to be executed  
}

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

Iterating Over an Array

You can use a for..of loop to iterate over the elements of an Array:

Example

for (const x of [1,2,3,4,5]) {  
  // code block to be executed  
}

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

JavaScript Iterators

The **iterator protocol** defines how to produce a **sequence of values** from an object.

An object becomes an **iterator** when it implements a next() method.

The next() method must return an object with two properties:

* value (the next value)
* done (true or false)

|  |  |
| --- | --- |
| **value** | The value returned by the iterator (Can be omitted if done is true) |
| **done** | *true* if the iterator has completed *false* if the iterator has produced a new value |

ADVERTISEMENT

Home Made Iterable

This iterable returns never ending: 10,20,30,40,.... Everytime next() is called:

Example

// Home Made Iterable  
function myNumbers() {  
  let n = 0;  
  return {  
    next: function() {  
      n += 10;  
      return {value:n, done:false};  
    }  
  };  
}  
  
// Create Iterable  
const n = myNumbers();  
n.next(); // Returns 10  
n.next(); // Returns 20  
n.next(); // Returns 30

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

The problem with a home made iterable:

It does not support the JavaScript for..of statement.

A JavaScript iterable is an object that has a **Symbol.iterator**.

The Symbol.iterator is a function that returns a next() function.

An iterable can be iterated over with the code: for (const x of iterable) { }

Example

// Create an Object  
myNumbers = {};  
  
// Make it Iterable  
myNumbers[Symbol.iterator] = function() {  
  let n = 0;  
  done = false;  
  return {  
    next() {  
      n += 10;  
      if (n == 100) {done = true}  
      return {value:n, done:done};  
    }  
  };  
}

Now you can use for..of

for (const num of myNumbers) {  
  // Any Code Here  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_iterate_homemade2)

The Symbol.iterator method is called automatically by for..of.

But we can also do it "manually":

Example

let iterator = myNumbers[Symbol.iterator]();  
  
while (true) {  
  const result = iterator.next();  
  if (result.done) break;  
  // Any Code Here  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_iterate_homemade3)

JavaScript Sets

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

A JavaScript Set is a collection of unique values.

Each value can only occur once in a Set.

A Set can hold any value of any data type.

Set Methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| new Set() | Creates a new Set |
| add() | Adds a new element to the Set |
| delete() | Removes an element from a Set |
| has() | Returns true if a value exists |
| clear() | Removes all elements from a Set |
| forEach() | Invokes a callback for each element |
| values() | Returns an Iterator with all the values in a Set |
| keys() | Same as values() |
| entries() | Returns an Iterator with the [value,value] pairs from a Set |

|  |  |
| --- | --- |
| **Property** | **Description** |
| size | Returns the number elements in a Set |

How to Create a Set

You can create a JavaScript Set by:

* Passing an Array to new Set()
* Create a new Set and use add() to add values
* Create a new Set and use add() to add variables

The new Set() Method

Pass an Array to the new Set() constructor:

Example

// Create a Set  
const letters = new Set(["a","b","c"]);

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

Create a Set and add literal values:

Example

// Create a Set  
const letters = new Set();  
  
// Add Values to the Set  
letters.add("a");  
letters.add("b");  
letters.add("c");

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

Create a Set and add variables:

Example

// Create Variables  
const a = "a";  
const b = "b";  
const c = "c";  
  
// Create a Set  
const letters = new Set();  
  
// Add Variables to the Set  
letters.add(a);  
letters.add(b);  
letters.add(c);

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

The add() Method

Example

letters.add("d");  
letters.add("e");

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

If you add equal elements, only the first will be saved:

Example

letters.add("a");  
letters.add("b");  
letters.add("c");  
letters.add("c");  
letters.add("c");  
letters.add("c");  
letters.add("c");  
letters.add("c");

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

ADVERTISEMENT

The forEach() Method

The forEach() method invokes a function for each Set element:

Example

// Create a Set  
const letters = new Set(["a","b","c"]);  
  
// List all entries  
let text = "";  
letters.forEach (function(value) {  
  text += value;  
})

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

The values() Method

The values() method returns an Iterator object containing all the values in a Set:

Example

letters.values()   // Returns [object Set Iterator]

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

Now you can use the Iterator object to access the elements:

Example

// Create an Iterator  
const myIterator = letters.values();  
  
// List all Values  
let text = "";  
for (const entry of myIterator) {  
  text += entry;  
}

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

The keys() Method

A Set has no keys.

keys() returns the same as values().

This makes Sets compatible with Maps.

Example

letters.keys()   // Returns [object Set Iterator]

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

The entries() Method

A Set has no keys.

entries() returns [value,value] pairs instead of [key,value] pairs.

This makes Sets compatible with Maps:

Example

// Create an Iterator  
const myIterator = letters.entries();  
  
// List all Entries  
let text = "";  
for (const entry of myIterator) {  
  text += entry;  
}

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

Sets are Objects

For a Set, typeof returns object:

typeof letters;      // Returns object

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

For a Set, instanceof Set returns true:

letters instanceof Set;  // Returns true

# JavaScript Maps

[❮ Previous](https://www.w3schools.com/js/js_object_sets.asp)[Next ❯](https://www.w3schools.com/js/js_object_es5.asp)

A Map holds key-value pairs where the keys can be any datatype.

A Map remembers the original insertion order of the keys.

A Map has a property that represents the size of the map.

## Map Methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| new Map() | Creates a new Map object |
| set() | Sets the value for a key in a Map |
| get() | Gets the value for a key in a Map |
| clear() | Removes all the elements from a Map |
| delete() | Removes a Map element specified by a key |
| has() | Returns true if a key exists in a Map |
| forEach() | Invokes a callback for each key/value pair in a Map |
| entries() | Returns an iterator object with the [key, value] pairs in a Map |
| keys() | Returns an iterator object with the keys in a Map |
| values() | Returns an iterator object of the values in a Map |

|  |  |
| --- | --- |
| **Property** | **Description** |
| size | Returns the number of Map elements |

## How to Create a Map

You can create a JavaScript Map by:

* Passing an Array to new Map()
* Create a Map and use Map.set()

## new Map()

You can create a Map by passing an Array to the new Map() constructor:

### Example

// Create a Map  
const fruits = new Map([  
  ["apples", 500],  
  ["bananas", 300],  
  ["oranges", 200]  
]);

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

## Map.set()

You can add elements to a Map with the set() method:

### Example

// Create a Map  
const fruits = new Map();  
  
// Set Map Values  
fruits.set("apples", 500);  
fruits.set("bananas", 300);  
fruits.set("oranges", 200);

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

The set() method can also be used to change existing Map values:

### Example

fruits.set("apples", 500);

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

## Map.get()

The get() method gets the value of a key in a Map:

### Example

fruits.get("apples");    // Returns 500

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

ADVERTISEMENT

## Map.size

The size property returns the number of elements in a Map:

### Example

fruits.size;

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

## Map.delete()

The delete() method removes a Map element:

### Example

fruits.delete("apples");

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

## Map.clear()

The clear() method removes all the elements from a Map:

### Example

fruits.clear();

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

## Map.has()

The has() method returns true if a key exists in a Map:

### Example

fruits.has("apples");

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

### Try This:

fruits.delete("apples");  
fruits.has("apples");

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

## Maps are Objects

typeof returns object:

### Example

// Returns object:  
typeof fruits;

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

instanceof Map returns true:

### Example

// Returns true:  
fruits instanceof Map;

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

## JavaScript Objects vs Maps

#### **Differences between JavaScript Objects and Maps:**

|  |  |
| --- | --- |
| **Object** | **Map** |
| Not directly iterable | Directly iterable |
| Do not have a size property | Have a size property |
| Keys must be Strings (or Symbols) | Keys can be any datatype |
| Keys are not well ordered | Keys are ordered by insertion |
| Have default keys | Do not have default keys |

## Map.forEach()

The forEach() method invokes a callback for each key/value pair in a Map:

### Example

// List all entries  
let text = "";  
fruits.forEach (function(value, key) {  
  text += key + ' = ' + value;  
})

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

## Map.entries()

The entries() method returns an iterator object with the [key,values] in a Map:

### Example

// List all entries  
let text = "";  
for (const x of fruits.entries()) {  
  text += x;  
}

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

## Map.keys()

The keys() method returns an iterator object with the keys in a Map:

### Example

// List all keys  
let text = "";  
for (const x of fruits.keys()) {  
  text += x;  
}

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

## Map.values()

The values() method returns an iterator object with the values in a Map:

### Example

// List all values  
let text = "";  
for (const x of fruits.values()) {  
  text += x;  
}

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

You can use the values() method to sum the values in a Map:

### Example

// Sum all values  
let total = 0;  
for (const x of fruits.values()) {  
  total += x;  
}

[Try it Yourself »](https://www.w3schools.com/js/tryit.asp?filename=tryjs_map_values1)

## Objects as Keys

Being able to use objects as keys is an important Map feature.

### Example

// Create Objects  
const apples = {name: 'Apples'};  
const bananas = {name: 'Bananas'};  
const oranges = {name: 'Oranges'};  
  
// Create a Map  
const fruits = new Map();  
  
// Add new Elements to the Map  
fruits.set(apples, 500);  
fruits.set(bananas, 300);  
fruits.set(oranges, 200);

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