JavaScript Objects in Detail

JAN. 27 2013 169

JavaScript's core—most often used and most fundamental—data type is the Object data type. JavaScript has one complex data type, the Object data type, and it has five simple data types: Number, String, Boolean, Undefined, and Null. Note that these simple (primitive) data types ar immutable (cannot be changed), while objects are mutable (can be changed).

menu

What is an Object
An object is an unordered list of primitive data types (and sometimes reference data types) that is stored as a series of name-value pairs. Each item in the list is called a *property* (functions are called *methods*).

Consider this simple object:

```
var myFirstObject = {firstName: "Richard", favoriteAuthor: "Conrad"};
```

Think of an object as a list that contains items, and each item (a property or a method) in the list is stored by a name-value pair. The property names in the example above are firstName and favoriteAuthor. And the values are “Richard” and “Conrad.”

Property names can be a string or a number, but if the property name is a number, it has to be accessed with the bracket notation. More on bracket notation later. Here is another example of objects with numbers as the property name:

```
 var ageGroup = {30: "Children", 100:"Very Old"};
console.log(ageGroup.30) // This will throw an error
// This is how you will access the value of the property 30, to get value "Children"
console.log(ageGroup["30"]); // Children

//It is best to avoid using numbers as property names.
```

As a JavaScript developer you will most often use the object data type, mostly for storing data and for creating your own custom methods and functions.

Reference Data Type and Primitive Data Types
One of the main differences between reference data type and primitive data types is reference data type’s value is stored as a reference, it is not stored directly on the variable, as a value, as the primitive data types are. For example:

```
// The primitive data type String is stored as a value
var person = "Kobe";
var anotherPerson = person; // anotherPerson = the value of person
person = "Bryant"; // value of person changed

console.log(anotherPerson); // Kobe
console.log(person); // Bryant
```

It is worth noting that even though we changed *person* to “Bryant,” the *anotherPerson* variable still retains the value that person had.

Compare the primitive data saved-as-value demonstrated above with the save-as-reference for objects:

```
var person = {name: "Kobe"};
var anotherPerson = person;
person.name = "Bryant";

console.log(anotherPerson.name); // Bryant
console.log(person.name); // Bryant
```

In this example, we copied the *person* object to *anotherPerson*, but because the value in person was stored as a reference and not an actual value, when we changed the person.name property to "Bryant" the anotherPerson reflected the change because it never stored an actual copy of it's own value of the person's properties, it only had a reference to it.

Object Data Properties Have Attributes
Each data property (object property that store data) has not only the name-value pair, but also 3 attributes (the three attributes are set to true by default):
— **Configurable Attribute:** Specifies whether the property can be deleted or changed.
— **Enumerable:** Specifies whether the property can be returned in a for/in loop.
— **Writable:** Specifies whether the property can be changed.

Note that ECMAScript 5 specifies accessor properties along with the data properties noted above. And the accessor properties are functions (getters and setters). We will learn more about ECMAScript 5 in an already-scheduled post slated for February 15.

Creating Objects
These are the two common ways to create objects:

1. **Object Literals**
   The most common and, indeed, the easiest way to create objects is with the object literal described here:

```
// This is an empty object initialized using the object literal notation
var myBooks = {};

// This is an object with 4 items, again using object literal
var mango = {
```

```
color: "yellow",
shape: "round",
sweetness: 8,

howSweetAmI: function () {
console.log("Hmm Hmm Good");
}
}
```

2. **Object Constructor**
   The second most common way to create objects is with Object constructor. A constructor is a function used for initializing new objects, and you use the new keyword to call the constructor.

```
var mango =  new Object ();
mango.color = "yellow";
mango.shape= "round";
mango.sweetness = 8;

mango.howSweetAmI = function () {
console.log("Hmm Hmm Good");
}
```

While you can use some reserved word such as “for” as property names in your objects, it is wise to avoid this altogether.

Objects can contain any other data type, including Numbers, Arrays, and even other Objects.

Practical Patterns for Creating Objects
For simple objects that may only ever be used once in your application to store data, the two methods used above would suffice for creating objects.

Imagine you have an application that displays fruits and detail about each fruit. All fruits in your application have these properties: color, shape, sweetness, cost, and a showName function. It would be quite tedious and counterproductive to type the following every time you want to create a new fruit object.

```
var mangoFruit = {
```

```
color: "yellow",
sweetness: 8,
fruitName: "Mango",
nativeToLand: ["South America", "Central America"],

showName: function () {
console.log("This is " + this.fruitName);
},
nativeTo: function () {
 this.nativeToLand.forEach(function (eachCountry)  {
            console.log("Grown in:" + eachCountry);
        });
}
}
```

If you have 10 fruits, you will have to add the **same** code 10 times. And what if you had to make a change to the nativeTo function? You will have to make the change in 10 different places. Now extrapolate this to adding objects for members on a website and suddenly you realized the manner in which we have created objects so far is not ideal objects that will have instances, particularly when developing large applications.

To solve these repetitive problems, software engineers have invented patterns (solutions for repetitive and common tasks) to make developing applications more efficient and streamlined.

Here are two common patterns for creating objects. If you have done the Learn JavaScript Properly course, you would have seen the lessons in the Code Academy used this first pattern frequently:

1. **Constructor Pattern for Creating Objects**

```
function Fruit (theColor, theSweetness, theFruitName, theNativeToLand) {

    this.color = theColor;
    this.sweetness = theSweetness;
    this.fruitName = theFruitName;
    this.nativeToLand = theNativeToLand;

    this.showName = function () {
        console.log("This is a " + this.fruitName);
    }

    this.nativeTo = function () {
    this.nativeToLand.forEach(function (eachCountry)  {
       console.log("Grown in:" + eachCountry);
        });
    }



}
```

With this pattern in place, it is very easy to create all sorts of fruits. Thus:

```
var mangoFruit = new Fruit ("Yellow", 8, "Mango", ["South America", "Central America", "West Africa
mangoFruit.showName(); // This is a Mango.
mangoFruit.nativeTo();
//Grown in:South America
// Grown in:Central America
// Grown in:West Africa

var pineappleFruit = new Fruit ("Brown", 5, "Pineapple", ["United States"]);
pineappleFruit.showName(); // This is a Pineapple.
```

If you had to change the fruitName function, you only had to do it in one location. The pattern encapsulates all the functionalities and characteristics of all the fruits in by making just the single Fruit function with inheritance.

Notes:
— An inherited property is defined on the object's prototype property. For example:
someObject.prototype.firstName = "rich";

— An own property is defined directly on the object itself, for example:
// Let's create an object first:
var aMango = new Fruit ();
// Now we define the mangoSpice property directly on the aMango object
// Because we define the mangoSpice property directly on the aMango object, it is an own property of aMango, not an inherited property.
aMango.mangoSpice = "some value";

— To access a property of an object, we use object.property, for example:
console.log(aMango.mangoSpice); // “some value”

— To invoke a method of an object, we use object.method(), for example:
// First, lets add a method
aMango.printStuff = function () {return “Printing”;}

// Now we can invoke the printStuff method:
aMango.printStuff ();

2. **Prototype Pattern for Creating Objects**

```
function Fruit () {

}

Fruit.prototype.color = "Yellow";
Fruit.prototype.sweetness = 7;
Fruit.prototype.fruitName = "Generic Fruit";
Fruit.prototype.nativeToLand = "USA";

Fruit.prototype.showName = function () {
console.log("This is a " + this.fruitName);
}

Fruit.prototype.nativeTo = function () {
            console.log("Grown in:" + this.nativeToLand);
}
```

And this is how we call the Fruit () constructor in this prototype pattern:

```
var mangoFruit = new Fruit ();
mangoFruit.showName(); //
mangoFruit.nativeTo();
// This is a Generic Fruit
// Grown in:USA
```

Further Reading
For a complete discussion on these two patterns and a thorough explanation of how each work and the disadvantages of each, read Chapter 6 of *Professional JavaScript for Web Developers*. You will also learn which pattern Zakas recommends as the best one to use (Hint: it is neither of the 2 above).

How to Access Properties on an Object
The two primary ways of accessing properties of an object are with dot notation and bracket notation.

1. **Dot Notation**

```
// We have been using dot notation so far in the examples above, here is another example again:
var book = {title: "Ways to Go", pages: 280, bookMark1:"Page 20"};

// To access the properties of the book object with dot notation, you do this:
console.log ( book.title); // Ways to Go
console.log ( book.pages); // 280
```

2. **Bracket Notation**

```
// To access the properties of the book object with bracket notation, you do this:
console.log ( book["title"]); //Ways to Go
console.log ( book["pages"]); // 280

//Or, in case you have the property name in a variable:
var bookTitle = "title";
console.log ( book[bookTitle]); // Ways to Go
console.log (book["bookMark" + 1]); // Page 20
```

Accessing a property on an object that does not exist will result in *undefined.*

Own and Inherited Properties
Objects have inherited properties and own properties. The own properties are properties that were defined on the object, while the inherited properties were inherited from the object's Prototype object.

To find out if a property exists on an object (either as an inherited or an own property), you use the in operator:

```
// Create a new school object with a property name schoolName
var school = {schoolName:"MIT"};

// Prints true because schoolName is an own property on the school object
console.log("schoolName" in school);  // true

// Prints false because we did not define a schoolType property on the school object, and neither did tl
console.log("schoolType" in school);  // false

// Prints true because the school object inherited the toString method from Object.prototype.
console.log("toString" in school);  // true
```

hasOwnProperty

To find out if an object has a specific property as one of its own property, you use the hasOwnProperty method. This method is very useful because from time to time you need to enumerate an object and you want only the own properties, not the inherited ones.

```
// Create a new school object with a property name schoolName
var school = {schoolName:"MIT"};

// Prints true because schoolName is an own property on the school object
console.log(school.hasOwnProperty ("schoolName"));  // true

// Prints false because the school object inherited the toString method from Object.prototype, therefore
console.log(school.hasOwnProperty ("toString"));  // false
```

Accessing and Enumerating Properties on Objects

To access the enumerable (own and inherited) properties on objects, you use the for/in loop or a general for loop.

```
// Create a new school object with 3 own properties: schoolName, schoolAccredited, and schoolLocatior
var school = {schoolName:"MIT", schoolAccredited: true, schoolLocation:"Massachusetts"};

```

```
//Use of the for/in loop to access the properties in the school object
for (var eachItem in school) {
console.log(eachItem); // Prints schoolName, schoolAccredited, schoolLocation

}
```

Accessing Inherited Properties

Properties inherited from Object.prototype are not enumerable, so the for/in loop does not show them. However, inherited properties that are enumerable are revealed in the for/in loop iteration.

For example:

```
 //Use of the for/in loop to access the properties in the school object
for (var eachItem in school) {
console.log(eachItem); // Prints schoolName, schoolAccredited, schoolLocation

}

// Create a new HigherLearning function that the school object will inherit from.
/* SIDE NOTE: As Wilson (an astute reader) correctly pointed out in the comments below, the educatior
*/


function HigherLearning () {
this.educationLevel = "University";
}

// Implement inheritance with the HigherLearning constructor
var school = new HigherLearning ();
school.schoolName = "MIT";
school.schoolAccredited = true;
school.schoolLocation = "Massachusetts";


//Use of the for/in loop to access the properties in the school object
for (var eachItem in school) {
console.log(eachItem); // Prints educationLevel, schoolName, schoolAccredited, and schoolLocation
}
```

In the last example, note the educationLevel property that was defined on the HigherLearning function is listed as one of the school's properties, even though educationLevel is not an own property—it was inherited.

Object's Prototype Attribute and Prototype Property
The prototype attribute and prototype property of an object are critically important concepts to understand in JavaScript. Read my post JavaScript Prototype in Plain, Detailed Language for more.

Deleting Properties of an Object
To delete a property from an object, you use the delete operator. You cannot delete properties that were inherited, nor can you delete properties with their attributes set to configurable. You must delete the inherited properties on the prototype object (where the properties were defined). Also, you cannot delete properties of the global object, which were declared with the var keyword.

The delete operator returns true if the delete was successful. And surprisingly, it also returns true if the property to delete was nonexistent or the property could not be deleted (such as non-configurable or not owned by the object).

These examples illustrate:

```
var christmasList = {mike:"Book", jason:"sweater" }
delete christmasList.mike; // deletes the mike property

for (var people in christmasList) {
        console.log(people);
}
// Prints only jason
// The mike property was deleted

delete christmasList.toString; // returns true, but toString not deleted because it is an inherited method

// Here we call the toString method and it works just fine—wasn't deleted
christmasList.toString(); //"[object Object]"

// You can delete a property of an instance if the property is an own property of that instance.          r
```

```

console.log(school.hasOwnProperty("educationLevel")); true
// educationLevel is an own property on school, so we can delete it
delete school.educationLevel; true

// The educationLevel property was deleted from the school instance
console.log(school.educationLevel); undefined

// But the educationLevel property is still on the HigherLearning function
var newSchool = new HigherLearning ();
console.log(newSchool.educationLevel); // University

// If we had defined a property on the HigherLearning function's prototype, such as this educationLeve
HigherLearning.prototype.educationLevel2 = "University 2";

// Then the educationLevel2 property on the instances of HigherLearning would not be own property.

// The educationLevel2 property is not an own property on the school instance
console.log(school.hasOwnProperty("educationLevel2")); false
console.log(school.educationLevel2); // University 2

// Let's try to delete the inherited educationLevel2 property
delete school.educationLevel2; true (always returns true, as noted earlier)

// The inherited educationLevel2 property was not deleted
console.log(school.educationLevel2); University 2

```

Serialize and Deserialize Objects

To transfer your objects via HTTP or to otherwise convert it to a string, you will need to serialize it (convert it to a string); you can use the JSON.stringify function to serialize your objects. Note that prior to ECMAScript 5, you had to use a popular json2 library (by Douglas Crockford) to get the JSON.stringify function. It is now standardized in ECMAScript 5.

To Deserialize your object (convert it to an object from a string), you use the JSON.parse function from the same json2 library. This function too has been standardized by ECMAScript 5.

JSON.stringify Examples:

```
var christmasList = {mike:"Book", jason:"sweater", chelsea:"iPad" }
JSON.stringify (christmasList);
// Prints this string:
// "{"mike":"Book","jason":"sweater","chels":"iPad"}"

// To print a stringified object with formatting, add "null" and "4" as parameters:
JSON.stringify (christmasList, null, 4);
// "{
    "mike": "Book",
    "jason": "sweater",
    "chels": "iPad"
}"

// JSON.parse Examples \\
// The following is a JSON string, so we cannot access the properties with dot notation (like christmasL
var christmasListStr = '{"mike":"Book","jason":"sweater","chels":"iPad"}';

// Let's convert it to an object
var christmasListObj = JSON.parse (christmasListStr);

// Now that it is an object, we use dot notation
console.log(christmasListObj.mike); // Book
```

For more detailed coverage of JavaScript Objects, including ECMAScript 5 additions for dealing with objects, read chapter 6 of JavaScript: The Definitive Guide 6th Edition.

Posted in: 16 Important JavaScript Concepts, JavaScript / Tagged: JavaScript, Learn JavaScript, Objects

Richard

Thanks for your time; please come back soon. Email me here: javascriptissexy at gmail email, or use the contact form.

169 Comments

moncler online shop

January 30, 2013 at 4:04 am / Reply

Good post, I always like them.

R Sturim

February 2, 2013 at 10:05 am / Reply

The “Fruit” constructor under the “Constructor Pattern for Creating Objects” has several coding errors. If you run the code through JSHint they’ll become very apparent. Properties should be ended with a semicolon (not a comma) and assignment should use the “=” sign, not the “:” — hope that helps.

Richard Bovell (Author)

February 2, 2013 at 1:31 pm / Reply

Thanks, R Sturim, for pointing out the errors. I just fixed the code.

I am not surprised there are errors in this particular post, because I didn’t get the chance to JSHint them and throughly review them for errors, which I am definitely doing for all the other example codes I post.

brandnew

February 4, 2013 at 6:07 pm / Reply

Thank you for Posts. i’m trying to get a grasp of JavaScript and i had a question related to this post in particular your example under, “Constructor Pattern for Creating Objects”.
When i type in the console (FireFox) mangoFruit.fruitName, output is “Mango”.
Question arises when i type mangoFruit.nativeTo, it returns:

```
(function () {
for (var eachCountry in this.nativeTo) {
console.log(“Grown in:” + eachCountry);
}
```

})
or when i type mangoFruit.nativeTo() it returns undefined.
i'm not certain why this is? How is this method called to output "eachCountry" variable?

thank you

Richard Bovell (Author)
February 4, 2013 at 7:23 pm / Reply

@brandnew
Update (you are correct):
There was an error in the code, the this.nativeTo call in the for loop was calling the this.nativeTo function. It was not calling the this.nativeTo variable (the array) we really wanted. I just fixed the error.
Here is the full code

———————————————

```
function Fruit (theColor, theSweetness, theFruitName,
theNativeToLand) {

this.color = theColor;
this.sweetness = theSweetness;
this.fruitName = theFruitName;
this.nativeToLand = theNativeToLand;

this.showName = function () {
console.log("This is a " + this.fruitName);
}

this.nativeTo = function () {

this.nativeToLand.forEach(function (eachCountry) {
console.log("Grown in:" + eachCountry);
});
}

}
```

var mangoFruit = new Fruit ("Yellow", 8, "Mango", ["South America", "Central America", "West Africa"]);
mangoFruit.showName(); // This is a Mango.
mangoFruit.nativeTo();
//Grown in:South America
// Grown in:Central America
// Grown in:West Africa

var pineappleFruit = new Fruit ("Brown", 5, "Pineapple", ["United States"]);
pineappleFruit.showName(); // This is a Pineapple.

____

brandnew

February 4, 2013 at 7:42 pm / Reply

thanks for the response: So if i just wanted to see the output of ["South America", "Central America", "West Africa"] from mangoFruit i would need to type in the entire code in the console?
i already have the entire code on a text file, i'm just running the firefox web console.

Richard Bovell (Author)

February 4, 2013 at 9:15 pm / Reply

Since you already have the Fruit function in a JS file, then you don't have to put the entire code in FireFox. I didn't know you had it in a JS file already.

M Dunn

February 12, 2013 at 8:15 pm / Reply

Excellent post! Two typos, I think:
1. To solve these repetitive problems, software engineers have invented patterns (clever was to solve common receptive tasks) to make developing applications more efficient.

– should read: (clever ways to solve common repetitive tasks)

2. In the last example, note the educationLevel property that was defined on the HigherLearning function is not listed as one of the school's properties, even though educationLevel is not an own property—it was inherited.
– should be: "… HigherLearning function is listed …"
At least that is what I am seeing.

Finally, I want to make sure I understand this part:
"To delete a property from an object, you use the delete operator. You cannot delete properties that were inherited, nor can you delete properties with their attributes set to configurable. "

It seems that I can delete the educationalLevel property from the school object (delete school.educationLevel) even though it is inherited from HigherLearning. Indeed, when that property is deleted, it is also deleted from HigherLearning. Am I seeing this right?

Thank you!

Richard Bovell (Author)
February 13, 2013 at 1:37 pm / Reply

@M Dunn
Thanks for pointing out the two typos; I just fixed them.

I updated the post with a detailed explanation of why we can delete the educationLevel property from the school object.

BTW, the educationLevel property was not deleted from the HigherLearning function. See the additional "delete" details I added to the post.

M Dunn

February 13, 2013 at 8:22 pm / Reply

Ah! I was faked out by the “true” returned when I deleted the inherited property and indeed, that property _was_ deleted (better to say “removed”?) from the child object. Silly me, I did not check to see that, of course, the prototype object still had the property. This makes sense now. Thank you.

BTW, you still have a “receptive” where there should be a “repetitive”.

Ariana A.

February 28, 2013 at 6:31 pm / Reply

I think you need to write a book. As someone totally new to programming I’ve had a hard time reading the books you’ve suggested. Some things are easy to grasp but once I got to Objects I wanted to bang my head against the wall. This post is so much easier to understand! I feel like I have a much better understanding now. You explained it so clearly and the examples were easy to follow. Thank you.

Richard Bovell (Author)

March 1, 2013 at 9:49 am / Reply

Thank you very much for the encouraging words, Ariana. You are too kind.

I am very happy to hear that the article helped you to better understand JavaScript objects.

antonio

March 29, 2013 at 8:19 pm / Reply

These are great Tuts, thank you Richard!
I took your advice and work through almost everything here and elsewhere (in the console, editor etc.)…
Question: This may be pretty basic but given the fact you can change variables at will, doesn't that make them mutable? I mean if I declare…

var starship = "enterprise";
var starship = "reliant"; // won't starship now be reliant?

Richard Bovell (Author)

April 1, 2013 at 9:23 pm / Reply

I think this is a great question.

When you change a variable to a new string, you are **redefining** the variable and giving it a new value. Technically, this means that in the JS engine (behind the scenes), the first variable is destroyed and it is redefined with a new string. So you didn't actually change the variable; the original string was literally destroyed, then the new string was created and assigned to the same variable.

Here is a quick example to demonstrate that strings are immutable (cannot be changed):

```

var myName = "Antonio";
console.log (myName.toLowerCase()); // antonio
console.log (myName.replace("An", "An-")); // An-tonio
console.log(myName); // "Antonio"

```

The string was never changed, because strings in JavaScript are immutable—they cannot be changed. It appears that behind the scenes a new string is being created and returned every time we execute one of the string methods like toLowerCase().

antonio

April 5, 2013 at 11:26 am / Reply

Thank you Richard!!
After reading Chapter 4(JS.FWD), this become much clearer!
One last thing though regarding this;

```
function setName(obj){
obj.name = “Nicholas”;
obj = new Object();
obj.name = “Greg”;
}

var person = new Object();
setName(person);
alert(person.name); //”Nicholas”
```

I understand the point, this chapter was trying to convey with primitive vs. reference values, and I understand when “person” was passed in “setName” it picks up the value of “NIcholas” (it appears the argument “person” is going down the body of the function).
So is the creation of “obj = new Object() in effect acting as a “break” in a way to stop the argument from becoming “Greg?”

I modified the code to comment out the new obj:

```
function setName(obj){
obj.name = “Nicholas”;
//obj = new Object();
obj.name = “Greg”;
}
```

```
var person = new Object();
setName(person);
alert(person.name); //now “Greg”
```

Thanks!!!!

Yo

December 5, 2014 at 6:38 am / Reply

I tried some combinations and I think it is due the scope and preference of the Object.

tomas

April 3, 2013 at 1:45 pm / Reply

hi,
in this code:

```
var mangoFruit = {
color: “yellow”,
sweetness: 8,
fruitName: “Mango”,
nativeToLand: [South America, Central America],

showName: function () {
console.log(“This is ” + fruitName);
},
nativeTo: function () {
nativeToLand.forEach(function (eachCountry) {
console.log(“Grown in:” + eachCountry);
});
}
}
```

i’m getting this error:

Syntax error at line 5 while loading: syntax error
ativeToLand: [South America, Central Ame
_______________^

why is it?
thanks

Richard Bovell (Author)
April 4, 2013 at 10:41 pm / Reply

You are correct, Tomas. I forgot to wrap the strings (inside the *nativeToLand* array) in quotes. I fixed it above.

This **incorrect** line:
nativeToLand: [South America, Central America],

Should be this:

```
nativeToLand: ["South America", "Central America"],
```

Thanks for bringing this to my attention.

Soumen
July 23, 2013 at 10:04 am / Reply

var mangoFruit = {
color: “yellow”,
sweetness: 8,
fruitName: “Mango”,
nativeToLand: [“South America”, “Central America”],

showName: function () {
console.log(“This is ” + this.fruitName);
},

nativeTo: function () {
nativeToLand.forEach(function (eachCountry) {
console.log("Grown in:" + eachCountry);
});
}
};
mangoFruit.nativeTo();

This gives 'ReferenceError: nativeToLand is not defined' in jsfiddle and firebug. Where am I wrong?

Richard Bovell (Author)
July 31, 2013 at 1:52 am / Reply

It was an error in my code. I just fixed it. I forgot the "this" keyword. It should be:

```

// Note the "this.nativeToLand"
this.nativeToLand.forEach(function (eachCountry) {
console.log("Grown in:" + eachCountry);
});

```

I just created a JSbin with the example:
http://jsbin.com/anuyoc/1/edit

Gmoney
April 18, 2013 at 1:19 pm / Reply

Richard,
Went and read chapter 6 of 3rd Ed of Zakas book and I have a question.

Under the Parasitic Constr Patt and the Durable Constructor Pattern are these things true:

the whole premise of those patterns is IF you want to override the value returned (by using the return keyword)
AND both patterns "hide" what created it and any information about its type other then its prototype is the native Object. It also means each property that is a function is in the global namespace too?

Is that correct ?

I see some ppl argue/debate that you should ALWAYS hide information about an objects implemention.
Basically saying to always use the factory pattern

Richard Bovell (Author)
April 25, 2013 at 8:13 pm / Reply

There is no single pattern that should ALWAYS be used for every circumstance, but there are a couple of patterns that are more optimal, more well conceived than the others.

In my post "OOP in JavaScript: What You NEED to Know," I discuss the two best patterns for OOP in JS: 1. Combination Constructor/ Prototype Pattern: Best for Encapsulation

2. Parasitic Combination Inheritance Pattern" Best for Inheritance.

sajay
April 29, 2013 at 5:20 am / Reply

You Rocks…Love your posts…

Richard Bovell (Author)
April 29, 2013 at 6:47 am / Reply

Thanks much, Sajay. With a cool name like Sajay, you rock too 🙂

sajay

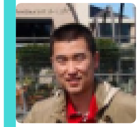
April 29, 2013 at 7:43 am / Reply

Richard, good to hear from you, I am taking my JS skills to advance level. I find your articles very well written, easy to follow. Keep up the good work. Thanks 🙂

Richard Bovell (Author)

April 30, 2013 at 11:24 pm / Reply

Thank you for the wonderful words, Sajay. I am very happy to hear my blog is helping you, and that you are taking your JS skills to the advanced level.



Wang Lei

May 2, 2013 at 6:44 am / Reply

Really good website for studying javascript

Richard Bovell (Author)

May 2, 2013 at 11:06 am / Reply

Thanks for the feedback, Wang.

Willson

May 14, 2013 at 5:11 pm / Reply

Hi Richard,

I'm a little confused. Here's the example where I'm confused:

function HigherLearning () {
this.educationLevel = "University";

}

var school = new HigherLearning ();
school.schoolName = "MIT";
school.schoolAccredited = true;
school.schoolLocation = "Massachusetts";

In this example, you described "educationLevel" as an inherited property. However, my understanding is that this isn't accurate (although I could totally be wrong here). First, school.hasOwnProperty("educationLevel") returns true which implies that this is an own property on the school object. Second, my understanding is that when the new keyword is used with a constructor function, the invocation context is the newly created object. Therefore, the "this" keyword in the HigherLearning function refers to the newly created object (i.e. this.educationLevel refers to school.educationLevel).

Is my understanding accurate?

Thanks,
Willson

Richard Bovell (Author)
May 15, 2013 at 12:06 am / Reply

> *In this example, you described "educationLevel" as an inherited property.*

You are correct, technically 🙂

So, this is the technical explanation (as you correctly observed): Since the "this" keyword in the Constructor points to the newly created object, in this case the *school* object, and therefore the *educationLevel* property will be created on the *school* object, then it follows that educationLevel is not really inherited from the

Constructor, but rather, it is a property that is actually created (new property) on the school object upon invocation of said school object.

Willson

May 15, 2013 at 12:14 am / Reply

Thanks for being so prompt with your responses, and helping to clarify my confusion!

Richard Bovell (Author)

May 15, 2013 at 12:19 am / Reply

I just updated the post and give you credit for your astute observation. 🙂

Keep pointing out errors whenever you find them. I am happy to update the articles and rid them of all typos, grammar mistakes, errors, technical missinformation, and the like.

Thanks again.

A N Sinha

May 15, 2013 at 3:23 pm / Reply

Hi Richard,

Thanks for posting the Excellent Topics!, and the way of clearing the concept. this article help me how to write code in advance level. It really helpful for programmer.

basically, I am upgrading the skills set in advance level.

thanks your very much 🙂

A.N. Sinha 🙂

peacelion

June 2, 2013 at 2:30 pm / Reply

there is an error with mangoFruit.showName().

showName should be:

showName: function () {
console.log("This is " + this.fruitName);
}

Richard Bovell (Author)

June 5, 2013 at 10:57 pm / Reply

The showName method added on the Constructor in the code in the article is correct.

peacelion

June 6, 2013 at 5:16 am / Reply

Correct. But the typo is just before, in the mangoFruit object definition…

Richard Bovell (Author)

June 6, 2013 at 8:50 pm / Reply

I am looking for the error, but I am not seeing it, so I need your help to find it 🙂 .

Below is the full code that I copied and pasted from the article above. You can run it in JSBin or your console and you will see that it runs just fine, with the expected results.

```

function Fruit (theColor, theSweetness, theFruitN

    this.color = theColor;
    this.sweetness = theSweetness;
    this.fruitName = theFruitName;
    this.nativeToLand = theNativeToLand;

    this.showName = function () {
        console.log("This is a " + this.fruitName);
    }

    this.nativeTo = function () {
    this.nativeToLand.forEach(function (eachCount
       console.log("Grown in:" + eachCountry);
        });
    }


}

var mangoFruit = new Fruit ("Yellow", 8, "Mango"
mangoFruit.showName(); // This is a Mango.
mangoFruit.nativeTo();

```

peacelion

June 7, 2013 at 5:26 am /

just a few lines before in the post:

var mangoFruit = {
color: “yellow”,

sweetness: 8,
fruitName: “Mango”,
nativeToLand: [“South America”, “Central America”],

showName: function () {
console.log(“This is ” + fruitName);
},
nativeTo: function () {
nativeToLand.forEach(function (eachCountry) {
console.log(“Grown in:” + eachCountry);
});
}
}
—
console.log(“This is ” + this.fruitName);
—
the “this” is missing

Richard Bovell (Author)
June 7, 2013 at 2:49 pm /

Thanks very much for your persistent, peacelion, I just found the error and I fixed it. The code was indeed missing the “this.” Good catch.

peacelion
June 7, 2013 at 5:27 am /

And by the way thanx a lot for your tutorials !!! IMHO the best about on internet.

Richard Bovell (Author)

June 7, 2013 at 2:50 pm /

Thank you, I am very happy that the blog is proving to be helpful and instructive. I really appreciate your kind words.

Anand Kumar

July 10, 2013 at 5:01 pm / Reply

Thanks a lot for this article and others on js. I have been struggling to learn the OOP concepts in javascript and has been too lazy to go through voluminous books but I do have spent time on google and youtubes reading articles and watching videos and then I suddenly stumbled upon your blog. You have a great simple way of explaining such complex topics. I am planning to read most of your posts and I hope you keep adding content on the website. I will refer this to all my friends who want to deep dive into the OOP concents of JS.

Richard Bovell (Author)

July 11, 2013 at 8:01 pm / Reply

Thank you very much, Anand.

Gracie

July 12, 2013 at 2:24 am / Reply

Hey! Found another error at the very beginning. Yours says:

var ageGroup = {30: “Children”, 100:”Very Old”};
console.log(ageGroup.30) // This will throw an error

// This is how you will access the value of the property 30, to get value "Children"
console.log(ageGroup["30"]}; // Children

console.log's end parenthesis is replaced with an end curly brace instead. Thanks for the post! The track is helping me a lot so far!

Richard Bovell (Author)
July 17, 2013 at 3:05 pm / Reply

Thanks very much, Gracie.

Good catch. I just fixed the typo.

Gracie Gutman
July 12, 2013 at 2:25 am / Reply

Hey! Found another error at the very beginning. Yours says:

var ageGroup = {30: "Children", 100:"Very Old"};
console.log(ageGroup.30) // This will throw an error
// This is how you will access the value of the property 30, to get value "Children"
console.log(ageGroup["30"]}; // Children

console.log's end parenthesis is replaced with an end curly brace instead. Thanks for the post! The track is helping me a lot so far.

Ehsan
July 28, 2013 at 11:48 am / Reply

Hi
Thanks a lot for your great posts
I have a question what is the usage of prototype pattern in your example?
The only usage I have in my mind is this:

We have one object and we don't make its properties with prototype pattern. Then we make another object that inherits the properties of the first object now we use prototype pattern to say the second object inherits all the properties of the first one. Am I right?

Richard Bovell (Author)

July 31, 2013 at 2:46 am / Reply

Yep, you are correct, Ehsan.

Sergi

July 31, 2013 at 12:41 pm / Reply

Richard, i know you appreciate typo notifications, so here's one for you: in the article above, the word RETAIN needs to be plural.

e.g., the anotherPerson variable still retain the value > INCORRECT
vs
the anotherPerson variable still retains the value > CORRECT

Richard Bovell (Author)

July 31, 2013 at 2:01 pm / Reply

Very good catch, Sergi. Thanks a lot. I just fixed it.

And you are correct that I do really appreciate typos, bugs, grammatical errors, and all such notifications.

Tim

August 8, 2013 at 4:02 pm / Reply

What an awesome read! Helped demistify the subject completely...and the subject on prototyping was awesome.

Thanks for the great article! I'm on week three of your javascript program and I'm learning much more than my previously failed attempts using other methods.

Richard Bovell (Author)
August 9, 2013 at 8:07 pm / Reply

Wonderful!.

Ramesh Kunhiraman
August 9, 2013 at 7:35 am / Reply

The best article I came across on JavaScript. Excellent piece of work

Richard Bovell (Author)
August 9, 2013 at 8:18 pm / Reply

Those are wonderful words, Ramesh :). Thank you very much.

antonio
August 9, 2013 at 11:40 am / Reply

Hey Richard,
Just reread this post and now I totally get it! I actually reread the post about 'passing by value vs. reference'
Thanks,
antonio

Richard Bovell (Author)
August 9, 2013 at 8:19 pm / Reply

I am glad to hear your JS skills are improving. Keep up the good thetrystero

September 15, 2013 at 2:26 am / Reply

howSweetAmI: function () {
console.log("Hmm Hmm Good");
}
}

Was there a typo here? there are two closing braces, but only one opening brace.

Richard Bovell (Author)

September 16, 2013 at 4:47 am / Reply

No, there was not a typo in that code (although I do make mistakes often 🙂).

Here is the full code:

```

// This is an object with 4 items, again using object literal
var mango = {
color: "yellow",
shape: "round",
sweetness: 8,
howSweetAmI: function () {
console.log("Hmm Hmm Good");
}

} // this last closing bracket belongs to the <em>mango</em> object

```

Foysal Mamun

September 30, 2013 at 7:43 am / Reply

Thanks for another good post.

Deepak

October 3, 2013 at 6:58 am / Reply

Thanks so much very useful.

Mohammed

October 6, 2013 at 5:49 pm / Reply

Great post!

Alessandro

October 7, 2013 at 12:37 pm / Reply

Great article, very well explained. I am a little bit more confident on objects now.

Doni Noeit

November 11, 2013 at 6:02 pm / Reply

"JavaScript has one complex data type, the Object data type, and it has five simple data types: Number, String, Boolean, Undefined, and Null. Note that these simple (primitive) data types are immutable, they cannot be changed, while objects are mutable."

[1] So if the String datatype is immutable, what does this do?

```
String.prototype.hasStuff = function (s) {
return Boolean(this.length > 0);
}
```

```
>> "tommy".hasStuff()
true
```

[2] So if,

>>typeof(“tommy”)
“string”
>>typeof(4)
“number”
>>typeof(true)
“boolean”
>>typeof(undefined)
“undefined”

and if

>>var bands = [“The Clash”, “Davis Bowie”, “Pink Floyd”]
>>typeof(bands)
“object”

>>typeof(null)
“object”

then null is a datatype how?

Richard Of Stanley (Author)
November 22, 2013 at 7:18 am / Reply

1. What you have done is add a method to the String prototype, which you can do for any of the data type.

By “immutable”, we mean you cannot change the instance of the data type: If you create a string instance, you cannot change it. You can alter a copy of it and save the copy to another variable, but the original string will not be changed.

For example (Immutable Data Type):

```
var originalStr = "Hello There";
originalStr.length = 0;
```

```
// The string did not change
console.log(originalStr); // "Hello There"

var newStr = originalStr.toLowerCase();
console.log(newStr); // "hello there"

// The original string variable never changed
console.log(originalStr); // "Hello There"
```

On the other hand, if we make any changes to an Object data type, such as an array, look what happens:
(Mutable Data Types)

```
var originalArr =["Hello There"];
originalArr.length = 0;
// The array changed; it is now empty
console.log(originalArr); []

originalArr.push("How are you?");
// It changed again
console.log(originalArr); // ["How are you?"]
```

2. Yes, Null is a data type, in fact.

Elhadi

December 15, 2013 at 10:08 pm / Reply

Thank you very much, the mentionned book is excellent also.

syed shaik shavali

January 7, 2014 at 11:55 pm / Reply

Thank you Richard.
Its good learning blog.

Syed.

Alex Andrew

January 12, 2014 at 1:12 pm / Reply

nice articels!

sanjeev dhakal

January 29, 2014 at 3:54 am / Reply

i had learned javascript for past two weeks and wile printing output i used document.write but now onward while i am going to learn more advance javascript i find console.log instead of document.write i cannot diffirentiate their working mechanism.

neelesh jain

March 5, 2014 at 6:26 am / Reply

Hey,
console.log("xyz"); – prints "xyz" on the console of the browser, not on the browser HTML page. only the developer can see this, not the user of the website and hence, it is mainly used for debugging.
while,
document.write("xyz"); – will print "xyz" on the webpage itself.

neelesh jain

March 5, 2014 at 6:23 am / Reply

Thanks for the post. It was very useful for me as i am just beginning on the JS journey, it cleared quite a few concepts.
Thankyou Again

Pravin Waychal

March 7, 2014 at 2:23 am / Reply

Hi Richard,

Will you please add example which shows how to get values of property in object.
or maybe be its allready written and I couldn't able to find it out.

var Fruit = function(color){
this.color = color
}

var mango = new Fruit("yellow");
mango.taste = "sweet";

for(var prop in mango){
console.log(mango[prop]); // Prints Yellow, Sweet
}

and thanks so much for making this complex subject so easy 🙂

Mike K

March 7, 2014 at 7:52 pm / Reply

Great runthrough of objects in JS, thanks very much!

Small typo/problem:
"To delete a property from an object, you use the delete operator. You cannot delete properties that were inherited, nor can you delete properties with their attributes set to configurable."

Did you mean "… attributes set to NOT configurable"?

Chandra

March 26, 2014 at 2:51 am / Reply

Hello Richard,
Good information and site for javascript lovers.
Is there any such website which is informative and easy to understand for jQuery, HTML5, CSS3.

Tarak

May 5, 2014 at 3:20 am / Reply

Hi Richard,

Thanks for posting the Excellent Topics!. Your explanation is also crystal clear.

-Thanx,
Tarak

Rai Butera

May 14, 2014 at 10:36 am / Reply

I found the sections "Accessing and Enumerating Properties on Objects" and "Accessing Inherited Properties" highly confusing as you don't seem to go over exclusively enumerating own properties, rather you seem to demonstrate inheritance in two ways.

Shaikh Siraj

July 8, 2014 at 4:26 am / Reply

Hi Richard

Very Very good example indeed, i know quit a lot about javascript, but still i learnt so much, especially foreach, delete property good

Keep going on..

MaC

July 29, 2014 at 2:24 am / Reply

Hey kudos to the nice article. Thank you very much. Really refresh javascript

Anthony Wijnen

July 30, 2014 at 7:05 am / Reply

Hey Richard!

First off, great stuff. I'm really enjoying your tutorials.

I was wondering about the following. You mentioned that using the constructor pattern for creating objects, you can easily make changes ones which permeate everywhere: "If you had to change the fruitName function, you only had to do it in one location. The pattern encapsulates all the functionalities and characteristics of all the fruits in by making just the single Fruit function with inheritance."

How would I do that? Let's say I created a couple of fruits with the Fruit() constructors and now I want them all to have a different showName() method. How would I accomplish that?

Thanks a lot!
Anthony

Benicio

August 20, 2014 at 9:39 am / Reply

A typo: In the section that reads " it never stored an actual copy of it's own value", should be " it never stored an actual copy of its own value".

Sorry, I like your posts. I go ADD when I see "it's" when it should be "its", I stop reading and wonder why that happens and if the code might have a typo as well.

Juan Llosas

September 8, 2014 at 3:07 pm / Reply

Man the guys got gypped that Christmas.. unless Mike got Professional JavaScript for Developers ofcourse.

asif

September 16, 2014 at 10:53 am / Reply

why there is need for creating copy of built in object,
ex:- var now = new Date();
why we can not use Date() object directly.,

ShafiShawon

September 24, 2014 at 2:13 pm / Reply

Really It's a great post. You deserve (y). I have many confusions before reading this.

Kris

December 5, 2014 at 7:19 pm / Reply

Great post! Under "Deleting Properties of an Object," you've written.. "nor can you delete properties with their attributes set to configurable."

That should be "non-configurable" or configurable set to false right?

Andrew

January 11, 2015 at 6:31 pm / Reply

I've been following through your advanced web developers track, and it's been great so far, but I've hit a wall on objects. I don't understand getters and setters (and accessor/data properties in general), which you don't seem to mention, but chapter 6 does in detail.

What is the value of using a 'value' accessor property over simply assigning a value to an attribute normally?
Also, what purpose do get and set fill? Get runs a function when that property is accessed (in the example, "year"), and set runs a function when that property is written to? Am I understanding correctly? I still feel shaky on objects in general, but I feel like I have a foundation to work on. I don't feel remotely that way about these properties.

Andrew

January 11, 2015 at 6:36 pm / Reply

If I understand correctly, get lays out the behavior for what the defined property should do when it's called. In the case of the example in the book, on page 177, that is simply to return the same value that _year holds. (But why not just define year to begin with and skip the _year assignment? What do we gain from that?)

Set, on the other hand, lays out the behavior for what year should do when it's written to. In the case of this example, it updates its own value (and _year) as well as updating the edition accordingly.

Am I on the right track? I'm still fuzzy on the real-world utility of Object.defineProperty().

Andrew

January 11, 2015 at 6:40 pm / Reply

Apologies to deluge with comments. I'm just questioning the real world utility of defineProperty(). For example, doesn't this code snippet accomplish the same thing as the example on page 177, without using this property? It just seems like needless complication.

```
var bookTest = {
year: 2004,
edition: 1,

updateEdition: function(newYear) {
if (newYear > 2004) {
this.year = newYear;
this.edition += newYear – 2004;
}
}
};

bookTest.updateEdition(2010);
alert(bookTest.edition);
```

Gytis

January 30, 2015 at 3:24 pm / Reply

I found the use of functions together with "new" keyword interesting. Maybe you should a a bit explaining how/why this works in JS.

avoaja

February 10, 2015 at 8:44 am / Reply

Very good post. Written 2 years ago but still fresh.

Ajay

February 17, 2015 at 6:34 am / Reply

Hi,
In the "Constructor pattern for objects" section, there needs to be a semi-colon at the end of the braces
here:

this.showName = function () {
console.log("This is a " + this.fruitName);
};

this.nativeTo = function () {
this.nativeToLand.forEach(function (eachCountry) {
console.log("Grown in:" + eachCountry);
});
};

Google Chrome's console throws an ILLEGAL token exception without the semicolon.

Thank you for the explanation. It helped me a lot.

gheorghe
March 14, 2015 at 8:34 am / Reply

Hi, I really love your posts, I need to say that I love to follow you, but I have one question. I saw that you recommend several books, like "JavaScript.The Definitive Guide" or Professional JavaScript for Web Developers" ....so I took the decision to stard reading one of these, but I must choose only one, not both because there is no time for both, so why would you think is better?

gheorghe
March 14, 2015 at 8:39 am / Reply

sorry, I`ve made a mistake at the end:
.... so, WHAT(not why) would you think is better?

some
May 17, 2015 at 2:23 pm / Reply

Hi,
Could you tell what is the diffrence between using:
this.property (or method) in the constructor and :
Constructor.prototype.property (or method).

I will be very happy if you find time and explain me that 🙂

Anand Rajput

May 29, 2015 at 4:11 pm / Reply

Great inline explanation of Object..you are shakespeare of programming.!!! Great Post .

snehashis

June 28, 2015 at 6:58 pm / Reply

Really Good

PiotrG

June 30, 2015 at 8:08 am / Reply

I read all the article and comments and I;m still little confused about this.
You write:
"In the last example, note the educationLevel property that was defined on the HigherLearning function is listed as one of the school's properties, even though educationLevel is not an own property—it was inherited."

and near below you write:
"console.log(school.hasOwnProperty("educationLevel")); true
// educationLevel is an own property on school, so we can delete it"

I saw the note where you add that the educationLevel property is not actually inherited by objects that use the HigherLearning constructor, so I think I get it.

But this text is confusing “educationLevel is not an own property—it was inherited” and below ” educationLevel is an own property on school, so we can delete it”

And the second ..
Please explain me this, because I think I don’t understand this the way I should.
“Properties inherited from Object.prototype are not enumerable, so the for/in loop does not show them”

So why the for in loop below show me print function and author property ?

```
function PrintStuff (myDocuments) {
this.documents = myDocuments;
}

PrintStuff.prototype.print = function () {
console.log(this.documents);
}

PrintStuff.prototype.author = “Stephen King”;
var newObj = new PrintStuff();

for (x in newObj) {
console.log(x);
}

//documents, print, author.
```

vasanth

July 9, 2015 at 12:52 am / Reply

In the last example, note the educationLevel property that was defined on the HigherLearning function is listed as one of the school’s properties, even though educationLevel is not an own property—it was inherited.

console.log(school.hasOwnProperty(“educationLevel”)); true
// educationLevel is an own property on school, so we can delete it

delete school.educationLevel; true

In the first condition you said that educationLevel is not the own property of school it's inherited. In second instance while deleting you said it has own property..Little bit confused.

Saras Arya

September 10, 2015 at 6:14 pm / Reply

I have two questions regarding delete.
First You cannot delete if property of object is set to Configurable. In the post you said the three attributes i.e configurable, enumerable and writable are set to true by default. If yes then how can you delete properties of any objects without setting configurable to false. which I don't see in any of your examples

Second You said properties of any object with var keyword cannot be deleted while you have the same in the following example. What is correct or what am I missing ? Kindly Explain?

Saras Arya

September 10, 2015 at 6:18 pm / Reply

I have two questions regarding delete.
First You cannot delete if property of object is set to Configurable. In the post you said the three attributes i.e configurable, enumerable and writable are set to true by default. If yes then how can you delete properties of any objects without setting configurable to false. which I don't see in any of your examples

Second You said properties of any object with var keyword cannot be deleted while you have the same in the following example. What is correct or what am I missing ? Kindly Explain

Alessandro

September 15, 2015 at 7:01 am / Reply

Thank you 🙂

DFF

October 7, 2015 at 3:15 pm / Reply

Just came across this article. It's a great resource for learning JS Objects

王如锵

October 13, 2015 at 9:28 pm / Reply

You said that "ducationLevel is not an own property—it was inherited".

And you said that "you cannot delete properties that were inherited".

So I am confused why that:
console.log(school.hasOwnProperty("educationLevel")); true
// educationLevel is an own property on school, so we can delete it
delete school.educationLevel; true

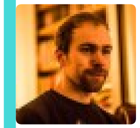
王如锵

October 13, 2015 at 9:28 pm / Reply

You said that "educationLevel is not an own property—it was inherited".

And you said that "you cannot delete properties that were inherited".

So I am confused why that:
console.log(school.hasOwnProperty("educationLevel")); true
// educationLevel is an own property on school, so we can delete it
delete school.educationLevel; true



Bart Kleijngeld

October 30, 2015 at 8:46 pm / Reply

TL;DR The second time the name property is set, the obj variable points to another -the newly created- object than the first time, which dies after the function execution finishes.

Within the `setName' function, let's focus on the line:
obj = new Object();

Here a new object is created and the obj variable is reassigned to point to it. When the name property is set on the line following, it is thus set on that object. Once the function finishes execution, because of its scope this object dies. Back in the global scope, the person variable points to the original object again, which had its name property set to'Nicholas'.

Vinay

November 5, 2015 at 2:37 am / Reply

Hello Richard, great explanation even a 6 year old can understand easily.

kindly write a article regarding mongoose explanation as it is frequently used in node js and regular expressions if possible.

thanks , you are my mentor 🙂

James

November 30, 2015 at 6:52 am / Reply

// Prints false because we did not define a schoolType property on the school object, and neither did the object inherit a schoolType property from its prototype object Object.prototype.
console.log("schoolType" in school); // false

Can you elaborate when you referring to "neither did the object inherit schoolType ……from its prototype object" What is object Object.prototype. Where do we find this in the example

SAUMYA RANJAN SAHU

December 3, 2015 at 2:35 pm / Reply

nice post!! good to know the basics

Jirayu Limjinda

December 21, 2015 at 4:41 am / Reply

So you make me clear with javascript object
thank you so much.

Khim

December 25, 2015 at 12:24 pm / Reply

Your JSON.stringfy example has an error that say it will print:
// "{"mike":"Book","jason":"sweater","chels":"iPad"}"
The "chels" should be "chelsea". Just thought it will help.

Kelvin

January 13, 2016 at 5:35 am / Reply

What can I say, you just explained the topic better than Nicholas C, Zakas

Tarun Nagpal

February 12, 2016 at 10:01 am / Reply

Awesome

Mahesh

February 25, 2016 at 4:27 am / Reply

Hello,

I am beginner to javascript. may i know from which post should i start to learn full JS.

Mahesh

February 25, 2016 at 4:30 am / Reply

Hello,

which is the best way to learn full JS and also please suggest me the study material

Rahul

March 1, 2016 at 8:59 am / Reply

Great tutorial , this really helped me clear my basics of object finally .

Thanks a ton !!!!

Ajay

March 9, 2016 at 5:12 am / Reply

It is good article sir, but there is correction in first example that native place of "Mango" is India…not an America, the scientific word of mango is "Mangifera indica". Do search on mango on internet you will find:
"The common mango species, or Mangifera indica, grows on the largest fruit tree in the world. Mango trees are native to northeast India, where Hindu writings about mangoes date back to 4000 B.C. Mangoes were domesticated in India, and spread to East Asia around 400 B.C. By the 10th century, mangoes were cultivated in East Africa, followed by the Philippines in the 15th century. Portuguese traders brought mango seeds along their route, spreading the trees to their African and Brazilian colonies in the 16th century."

kindlly, update your example sir.

Venkatesh

March 10, 2016 at 1:53 am / Reply

Hi,

I have little confusion in accessing the property in the object basically through bracket notation without quotes " "

//Or, in case you have the property name in a variable:
var bookTitle = "title";
console.log (book[bookTitle]); // Ways to Go
console.log (book["bookMark" + 1]); // Page 20

In this how can we create a " var bookTitle = "title"; " inside a constructor functions so that we can access the property/variable without using notation like this book[bookTitle]

Sorry if my question is too basic 🙂 However im a starter right now in javascript 🙂

Idris

March 28, 2016 at 10:35 am / Reply

Good one .. ease to learn

Jatni

April 15, 2016 at 6:06 am / Reply

Very nice post in not so comlex term.
Thank for posting it.

viet

April 15, 2016 at 11:49 am / Reply

Iam sorry SIr, I've changed the code a little bit:

I wrote:

```
this.nativeToLand.forEach(function (eachCountry) {
console.log("Grown in:" + eachCountry);
}
```

instead of

```
this.nativeTo = function () {
this.nativeToLand.forEach(function (eachCountry) {
console.log("Grown in:" + eachCountry);
});
```

then it returned: this.nativeToLand is not defined. Why did it happens

viet

April 15, 2016 at 11:49 am / Reply

Iam sorry SIr, I've changed the code a little bit:

I wrote:

```
this.nativeToLand.forEach(function (eachCountry) {
console.log("Grown in:" + eachCountry);
}
```

instead of

```
this.nativeTo = function () {
this.nativeToLand.forEach(function (eachCountry) {
console.log("Grown in:" + eachCountry);
});
```

then it returned: this.nativeToLand is not defined. Why did it happen

viet

April 15, 2016 at 12:20 pm / Reply

Oops, i've made a mistake. I reviewed my code again and found that. Sorry for that. Anyway, your blog is awesome and help me a lot. Thank u!

NITESH

April 16, 2016 at 2:26 am / Reply

Nice post about object and very helpfully .thanks for building my concept.

FlyingGambit

April 18, 2016 at 11:42 pm / Reply

An example I tried to see if it deletes inherited properties

var Person = function(name){this.name = name;}
var john = new Person("John");
Person.prototype.race = "Human";
console.log(john); // shows {name: "John"}
john.hasOwnProperty("race"); // shows false
john.race // shows "Human"
delete john.race // shows true
john.race // shows "Human"

FlyingGambit

April 18, 2016 at 11:44 pm / Reply

What is the difference between object property and object attribute ?

FlyingGambit

April 18, 2016 at 11:48 pm / Reply

Your articles are great, if you can then plz include summary sections at the end, some bullet points for quick referencing. I often find myself having to read the entire article again.

Jacek J.

April 27, 2016 at 7:38 pm / Reply

"nor can you delete properties with their attributes set to configurable" ——> should be changed to:
" "nor can you delete properties with their attributes set to NON – configurable"

I know you don't update it any longer (which is REALLY sad…), but the page is a really helpful and I hope you will change this so other people won't get confused.

ilhan

April 28, 2016 at 1:54 pm / Reply

Hi great article!

I have a comment on the following example:

```
function HigherLearning () {
this.educationLevel = "University";
}

// Implement inheritance with the HigherLearning constructor
var school = new HigherLearning ();
school.schoolName = "MIT";
school.schoolAccredited = true;
school.schoolLocation = "Massachusetts";


//Use of the for/in loop to access the properties in the school object
for (var eachItem in school) {
console.log(eachItem); // Prints educationLevel, schoolName, schoolAccredited, and schoolLocation
}
```

In the last example, note the educationLevel property that was defined on the HigherLearning function is listed as one of the school's properties, even though educationLevel is not an own property—it was inherited.

My question:
Are you sure “educationLevel property” is not an own property?
I think it would not be an own property if it was defined on HigherLearning.prototype, e.g HigherLearning.prototype.educationLevel.
But in this case I believe it is an own property.

You can use to test it yourself.
for (prop in school) {
if (school.hasOwnProperty(prop)) {
console.log(“prop: ” + prop)
}
}

I am looking forward to your reply

giorgi

October 12, 2016 at 3:51 am / Reply

Below is bad example even if string were reference type still anotherPerson would have Kobe, please correct
—-
// The primitive data type String is stored as a value
var person = “Kobe”;
var anotherPerson = person; // anotherPerson = the value of person
person = “Bryant”; // value of person changed

console.log(anotherPerson); // Kobe
console.log(person); // Bryant

giorgi

October 12, 2016 at 6:12 am / Reply

Also when showing two ways to create objects using constructor function and prototypes you don’t highlight what are differences with these approaches, when there are, e.g. prototype stuff is shared

Amit

October 19, 2016 at 1:23 pm / Reply

I like your post

Trackbacks for this post

1. JavaScript Prototype in Plain, Detailed Language | JavaScript is Sexy
2. 16 JavaScript Concepts You Must Know Well | JavaScript is Sexy
3. Learn Node.js Completely and with Confidence | JavaScript is Sexy
4. Learn Backbone.js Completely | JavaScript is Sexy
5. Javascript | Pearltrees
6. How to Learn JavaScript Properly | JavaScript is Sexy
7. OOP In JavaScript: What You NEED to Know | JavaScript is Sexy
8. Learning JS Properly – Week 2 | coding(isBananas);
9. JS is Sexy Week 2- A bit on Method and Properties | B's Blog
10. Javascript is Sexy Week 1&2- Structure at Last, I Found a Study Group | B's Blog
11. Abhinav (archmonk) | Pearltrees
12. Apply, Call, and Bind Methods are Essential for JavaScript Professionals | JavaScript is Sexy
13. A Simple, Comprehensive Overview of Javascript | BetterExplained
14. Beautiful JavaScript: Easily Create Chainable (Cascading) Methods for Expressiveness | JavaScript is Sexy
15. How to Learn JavaScript Properly - Ray Sinlao
16. JavaScript对象详解 - javascript - 开发者第2317377个问答
17. JavaScript对象详解 - javascript教程 - 开发者
18. 如何正确学习JavaScript | 应酷爱网页设计
19. sexy.com | JavaScript is Sexy
20. 如何正确学习JavaScript | 产品汪一技术百科
21. 「译」如何正确学习JavaScript – 千月殿 -
22. 如何正确学习JavaScript - javascript - 嗅探实事
23. 如何正确学习Javascript | 17khba
24. [译] 如何恰当地学习 JavaScript — 好JSER
25. Askerov Javid | Web development
26. javascript学习路线 | w3croad

27. JavaScript Concepts we should know well |
28. Program Outline | Tilly Codes
29. 「译」如何正确学习JavaScript - YelloWish
30. 「译」如何正确学习JavaScript | 大前端
31. jQuery & other updates | Alexa Weidinger
32. Javascript – Objects | Majesty
33. Learning MEAN Stack | Mandal's Musings
34. 如何正确学习JavaScript | 前端控
35. Code Craft – Embedding C++: Classes | Hackaday
36. JavaScript | Pearltrees
37. [퍼온글] 자바스크립트 배우기 - hodoogwaja
38. JavaScript Objects Overview – Tutorialspoint – sous forme d'optimisation avec la forme en ligne builder
39. JavaScript Objects Overview – Tutorialspoint – singlepointofoptimization
40. Object Expressions and Declarations – Kotlin Programming Language | launchformsinhighstyle

Leave a Reply

Name

Email (hidden)

Website

Comment:

Submit Comment

Notify me of follow-up comments by email.

Notify me of new posts by email.