JavaScript and Chill

An introduction to a cool language

JavaScript, not to be confused with Java, was created in 10 days in May 1995 by Brendan Eich.

… and some would say it shows

JavaScript is one of the world's most popular programming languages due to its role as the scripting language of the WWW.

Despite its popularity, few know that JavaScript is a dynamic object-oriented general-purpose programming language

Whaattt??

My First JavaScript

Click me to display Date and Time.

Mon Jan 11 2016 22:10:13 GMT-0400 (AST)

ALRIGHT

JS

JavaScript has three simple types:

number:	double-precision 64-bit format IEEE 754
boolean:	truthy value or falsey value
string:	sequences of Unicode characters (UTF-16 code units where each code unit is represented by a 16-bit number)

JavaScript numbers

Every number in JavaScript is a floating-point value--there are no integers.

2 + 2 = 3.999999999

JavaScript booleans

A boolean is either true or false. All values in JavaScript can be converted to a boolean.

All values are either truthy or falsey

JavaScript strings

There is no character type in JavaScript--only strings of length 1.

Strings are delimited with pairs of either the ‘ or “ character. Either is okay, but they must be in pairs

Implicit type conversion

Javascript will convert between types to make every effort to execute a statement.

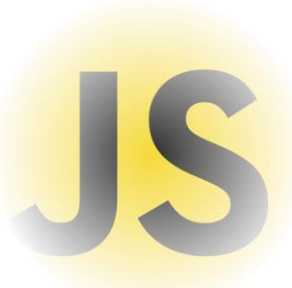
These conversions are not always what you think.

5 + 7 = 12

'5' + 7 = '57'

Original	to Number	to String	to Boolean
false	0	"false"	false
true	1	"true"	true
0	0	"0"	false
1	1	"1"	true
"0"	0	"0"	true
"1"	1	"1"	true
NaN	NaN	"NaN"	false
Infinity	Infinity	"Infinity"	true
-Infinity	-Infinity	"-Infinity"	true
""	0	""	false
"20"	20	"20"	true

Original	to Number	to String	to Boolean
false	0	"false"	
	false		
true	1	"true"	
	true		
0	0		"0"
		false	
1	1		"1"
		true	
"0"	0		"0"
		true	
"1"	1		"1"
		true	
NaN	NaN		"NaN"
	false		
Infinity	Infinity	"Infinity"	
	true		
-Infinity	-Infinity	"-Infinity"	
	true		
""	0		""
		false	
"20"	20	"20"	



JavaScript has another very important type: **Object**

"An implementation may provide other types, such as Dates and Regular Expressions, but these are really just objects. Everything else is just objects."

- Douglas Crockford.

Object: An object is a (referenceable) container of name / value pairs.

The names are strings (or other elements such as numbers that are converted to strings).

The values can be any of the data types including other objects.

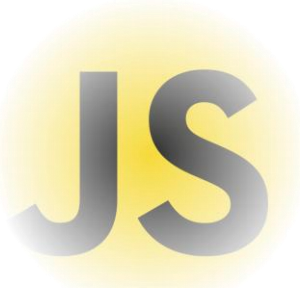
Creating a new object

(both are the same):

```
var myObject = {};  //use this one.
var myObject = new Object();
```

Creation and initialization:

```
var mascot = {
     name: 'Porkchop',
     age: {years: 3, days:124}
};
```



Assigning object properties

```
var obj = {name: 'Porkchop', 'weight': 22};
obj.colour = 'kinda red';
obj['ears'] = 'floppy';

var walker = 'legs';
obj[walker] = 4;
```

Accessing object properties

```
var myObj = {age: 10};
```

Dot notation requires a string constant.

```
var myAge = myObj.age;
```

Array notation uses any expression that resolves to a string.

```
var myAge = myObj['homage'.substring(3)];
```


DIVERSION

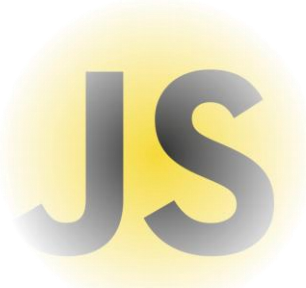
Defaults and guards

You can use `||` for assigning default values:

```
var name = myObj.name || '<none>';
```

You can guard against TypeError exception with `&&`

```
var name = myObj && myObj.name;
```



`&&` guard gotcha

Testing for the existence of a property using `myObj && myObj.myProperty` is BAD.

What happens when `myProperty` is falsey like when it equals `0`?

Use `myObj.hasOwnProperty('myProperty')` instead.


Diversion
End
2CAULDRONS

Adding and removing object properties

Objects are dynamic. You can add and remove properties at runtime.

If you assign to an object property it

- updates the property if it exists, or
- creates the property (augments the object) if it does not.

Delete a property using the delete operator

```
delete myObj.age;
```

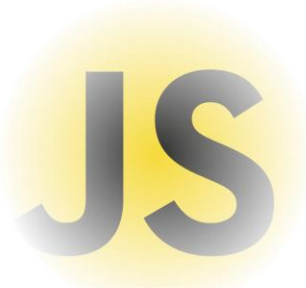
Iterating objects

```
var obj = {name: 'Porkchop', 'weight': 22};
for (var prop in obj) {
     console.log(prop + ' is ' + obj[prop]);
}
```

```
name is Porkchop
weight is 22
```

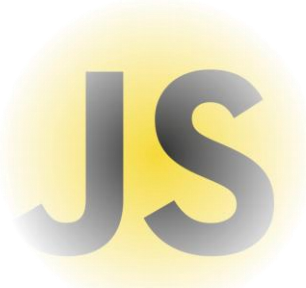
Objects are references

```
var objA = {age: 10};
var objB = objA;
// objA and objB refer to the same object

objB.color = 'green';
console.log(objA);

{ age: 10, color: 'green' }
```



Objects are references

```
var a = {}, b = {}, c = {};
        // a, b, and c each refer to a
        // different empty object

    a = b = c = {};
        // a, b, and c all refer to
        // the same empty object
```



Arrays

JavaScript arrays are objects that have a `length` property.

`length` is one higher than the largest index.

It is up to you to make sure the properties are numbers.

Creating arrays

```
var a = new Array();
a[0] = "dog";
a[1] = "cat";
a[2] = "hen";
a.length; // 3

var a = ["dog", "cat", "hen"];
a.length; // 3
```

Array length

```
var a = ["dog", "cat", "hen"];
a[100] = "fox";
a.length; // 101

a.length = 2;  // a = ["dog", "cat"];
```

Iterating arrays

```
for (var i = 0, len = a.length; i < len; i++)
{
  // Do something with a[i]
}
```

and, because Arrays are Objects

```
for (i in a) {        // i may not be in order
  // Do something with a[i]
}
```

Manipulating arrays

It is usually best to use the built-in array methods like

`push` to add an item to the end of an array

`pop` to remove and return the last item

`slice` to return a sub array

`concat` to combine arrays, and so on.

Functions

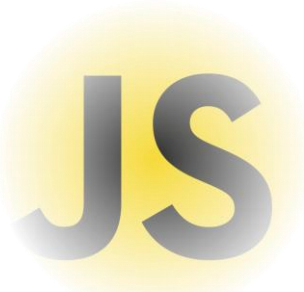
JavaScript functions work like you would expect them to.

However, a function is also an Object with extra abilities so this means that:

- A function can have properties
- You can store the function in a variable
- You can pass the function as a parameter to another function
- You can return the function from a function

Some examples using functions

```
function add(x, y) {
  return x + y;
}

add(3, 4)       // 7
add(4)              // NaN -> y is undefined
add(3, 4, 5)    // 7 -> 5 is ignored
add("3", 4)     // "34" -> 4 becomes "4"
```



Functions as variables

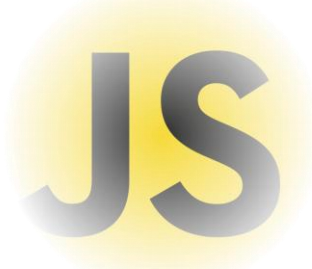
```
var adder = function(x, y) {
  return x + y;
}

// similar to the previous usage
adder(3, 4)          // 7
```

Functions as parameters

```
function doit(x, y, applyfn) {
  return applyfn(x, y)
}
```

Now, this is new ...

```
doit(3, 4, adder);
doit(3, 4, function(x, y) { return x - y; });
```



Adding Functions to Objects

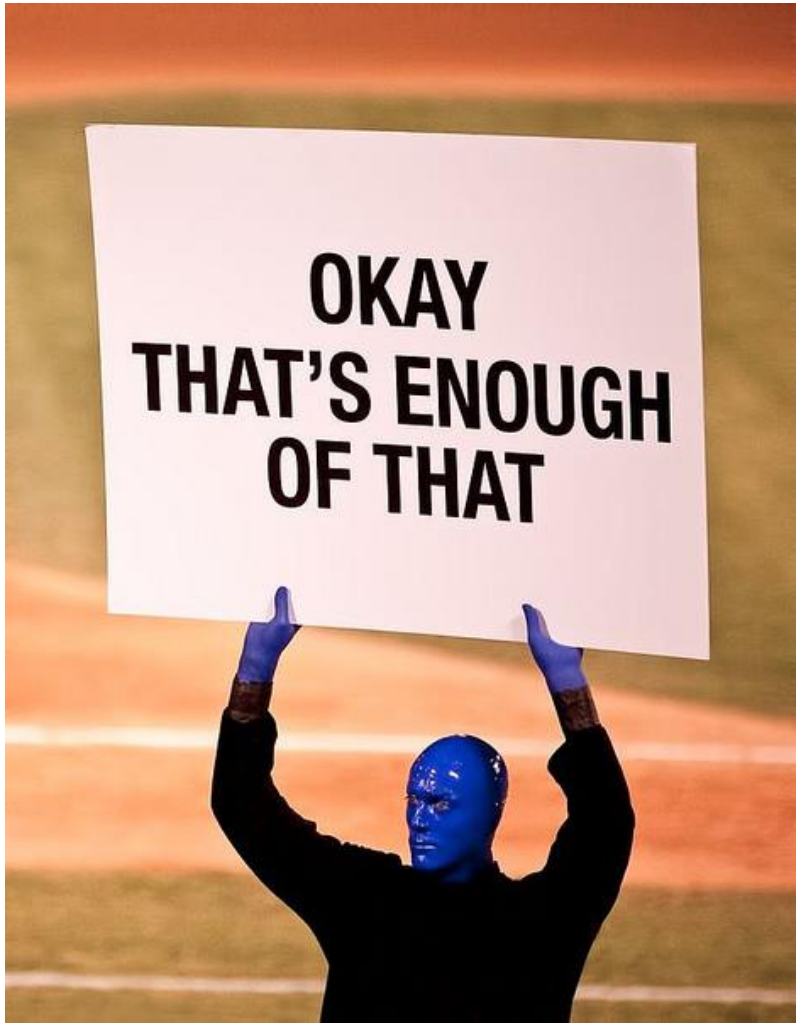
```
var student = {
        total: 0,
        grades: [],
    recordMark: function(mark) {
        this.grades.push(mark);
        this.total += mark;
        console.log(this.total / this.grades.length);
    }
};
student.recordMark(90);          // 90
student.recordMark(80);          // 85
```

Adding properties to Functions

```
var runningAverage = function(mark) {
    this.grades.push(mark);
    this.total += mark;
    console.log(this.total / this.grades.length);
};

runningAverage.total = 0;
runningAverage.grades = [];

runningAverage(90);         // 90
runningAverage(80);         // 85
```

Adding properties to Functions

```
var runningAverage = function(mark) {
        this.total = this.total || 0;
        this.grades = this.grades || [];
    this.grades.push(mark);
    this.total += mark;
    console.log(this.total / this.grades.length);
};

runningAverage(90);          // 90
runningAverage(80);          // 85
```

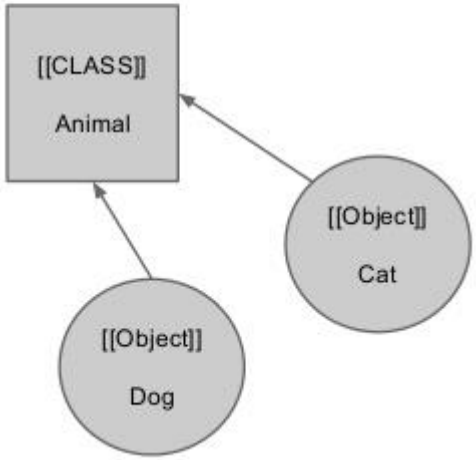

OKAY
THAT'S ENOUGH
OF THAT

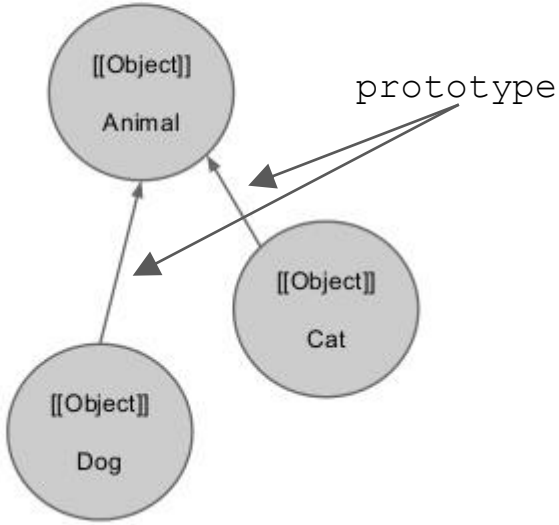

BUT WAIT,
THERE'S MORE!
made on imgur

There is much more to JavaScript

- prototypical inheritance
- real use of “this” in functions
- function bind apply
- closures (we saw just a little)
- use of JavaScript in the browser
- the many, many useful javaScript libraries
- constructors

Classical vs Prototypal Inheritance





JavaScript

JavaScript `prototype`

JavaScript objects have a special (read-only) property called `prototype` that points to a “parent” object. The `prototype` is assigned up when the object is created.

There are no classes in JavaScript, only objects.

`prototype` points to an object.

Object creation (with prototype)

I said to use this for a new Object

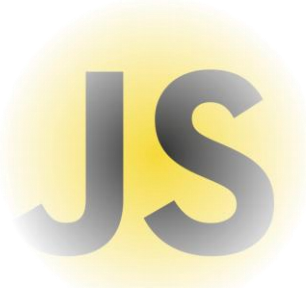
```
var myObj = {};
```

To instantiate an object with a specific prototype use:

```
var myObj = Object.create(<prototype object>);
```

For an object with no prototype:

```
var myObj = Object.create(null);
```

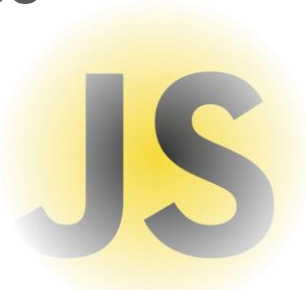


Default object `prototype`

There is a default `prototype` object in JavaScript that is assigned by default when you create an object.

This object has a number of properties and methods that are common to all objects.

The only way to get rid of the default prototype is to use `Object.create(null);`



The prototype chain

A request is made for a property of an object:

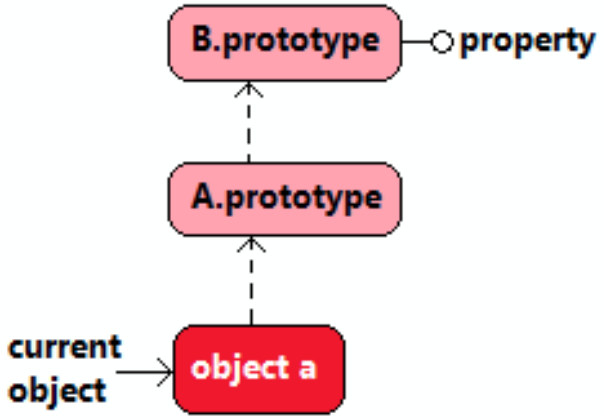
```
var = theObj.theProp;
```

If the object has a property called `theProp`, it is returned.

If it does not have the property `theProp` AND it has a prototype AND the prototype has the property `theProp`, the prototype's `theProp` is returned.

The prototype chain

The search for properties in prototypes continues until no more prototypes exist.



Some References

“A Short History of JavaScript” - https://www.w3.org/community/webed/wiki/A_Short_History_of_JavaScript

“A re-introduction to JavaScript (JS tutorial)” - https://developer.mozilla.org/en-US/docs/Web/JavaScript/A_re-introduction_to_JavaScript

“JavaScript: The World's Most Misunderstood Programming Language” - http://javascript.crockford.com/javascript.html

“A Survey of the JavaScript Programming Language” - http://javascript.crockford.com/survey.html