

# Lecture 1: Getting started...

---

Olivier Liechti  
TWEB



Haute Ecole d'Ingénierie et de Gestion  
du Canton de Vaud

# Today's agenda

14h00 - 15h00	60'	<b>Lecture</b> Introduction First tools JavaScript 101, part 1
<i>15h00 - 15h10</i>	<i>10'</i>	<i>Break</i>
15h10 - 16h25	75'	<b>Lecture</b> Introduction to lab (exercices) Heroku demo JavaScript 101, part 2

# Quick Poll

- **What is your current perception of Web development?**
- **Personal interest**
  - Web apps? It's **for junior developers and kids**, but real software engineers have better things to do.
  - Web apps? I am not a **graphics designer**... what am I doing here?
  - The Web is where some of the **most exciting technologies** are emerging.
  - When I grow up, I want to be a **front-end engineer**.

# Quick Poll

- **What is your current perception of Web development?**
- **Scope**
  - Web development is **purely about building user interfaces**. It's about HTML and CSS pages.
  - Web development is about building **complete applications**, including a user interface.
  - Is there **any kind of application** that does not rely on the Web, in one way or another? Mobile apps, interactive apps, business apps: all of this is built on the Web!

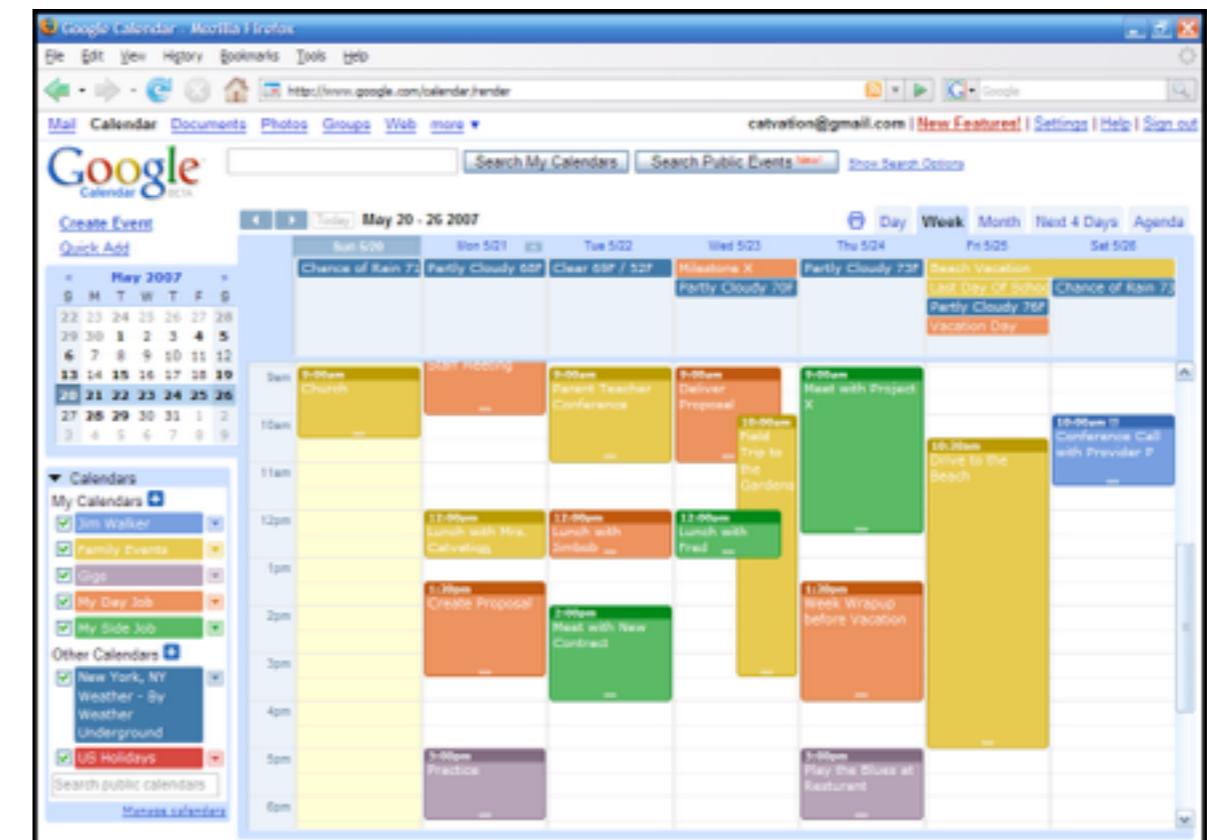
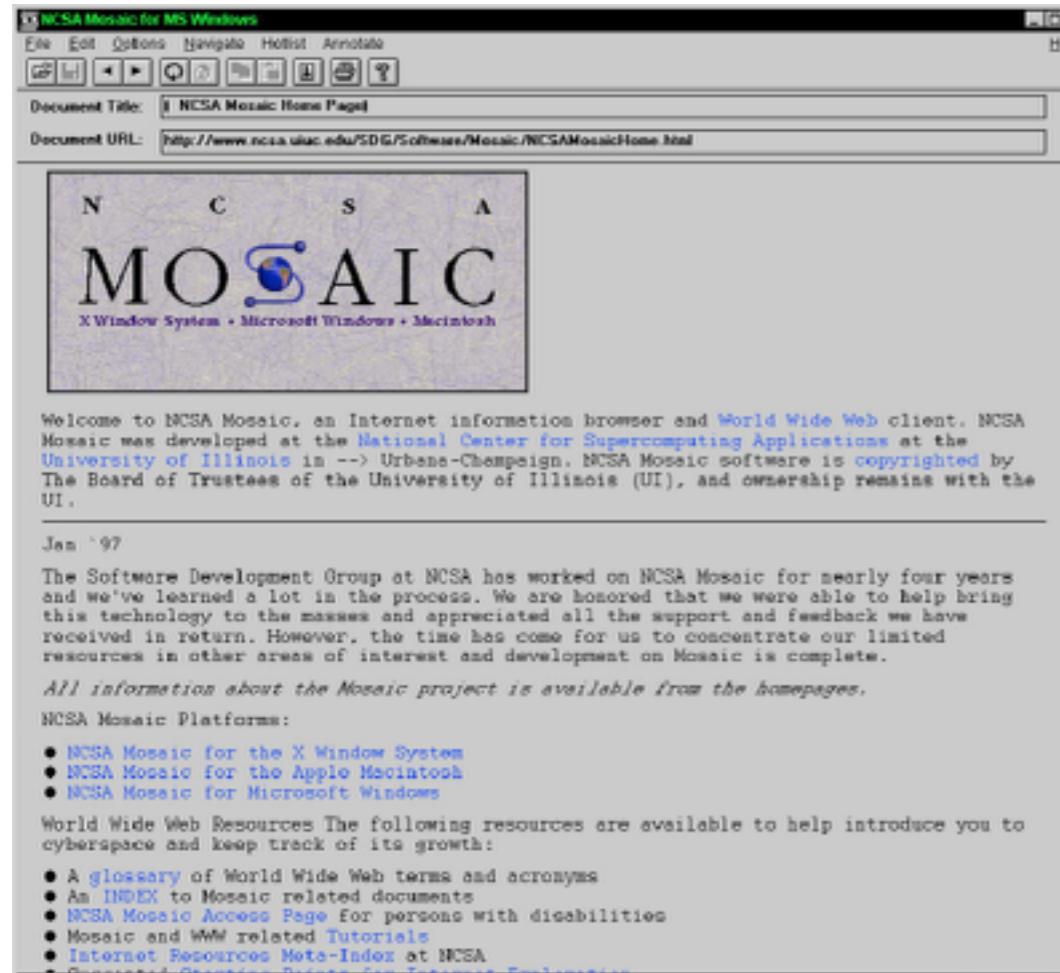


# Introduction

# The Web as an Application Platform

heig-vd

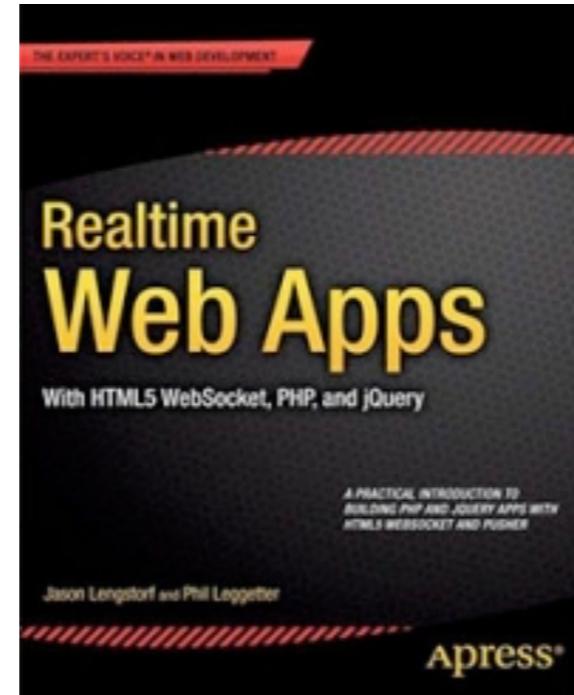
Haute Ecole d'Ingénierie et de Gestion  
du Canton de Vaud



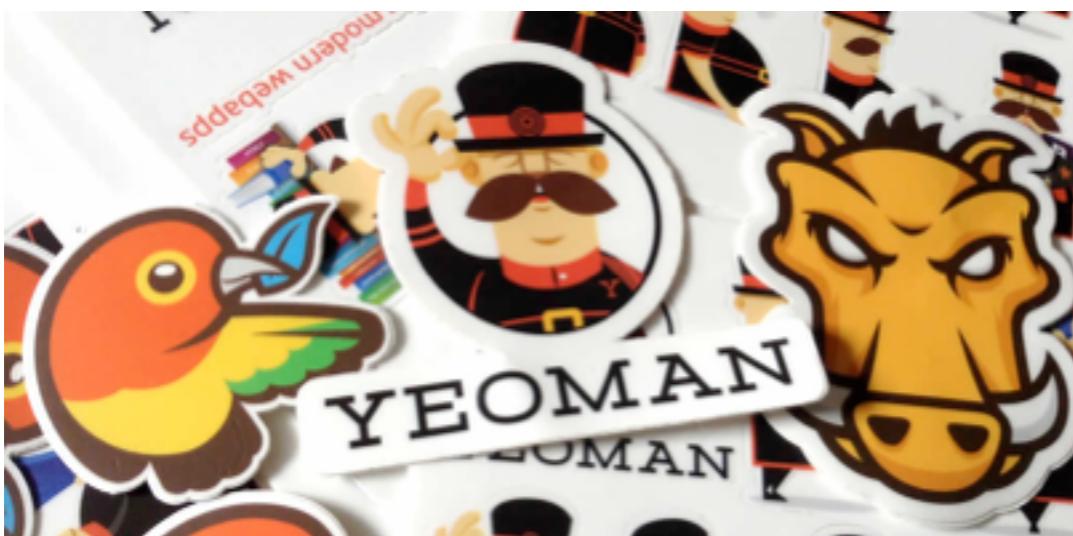
# Some Trends



SPA



Server Push



Automated Development Workflows



Mobile & RWD



Web of Things

# Languages, Platforms, Communities

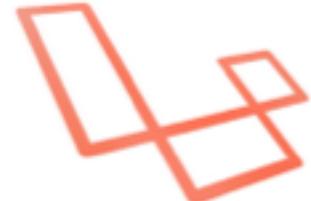


Client

Server



# Languages, Platforms, Communities



laravel



# JavaScript - End to End



Client

express

A large yellow square containing the letters "JS" in a bold, dark gray font.

The Node.js logo, which consists of the word "node" in a lowercase sans-serif font next to a green hexagon containing a white dot, followed by ".js" in a smaller green hexagon.

The Socket.IO logo, which consists of the words "SOCKET" and "IO" stacked vertically in a blue sans-serif font.

The Meteor logo, which features a stylized white blob icon above the word "Meteor" in a black sans-serif font.

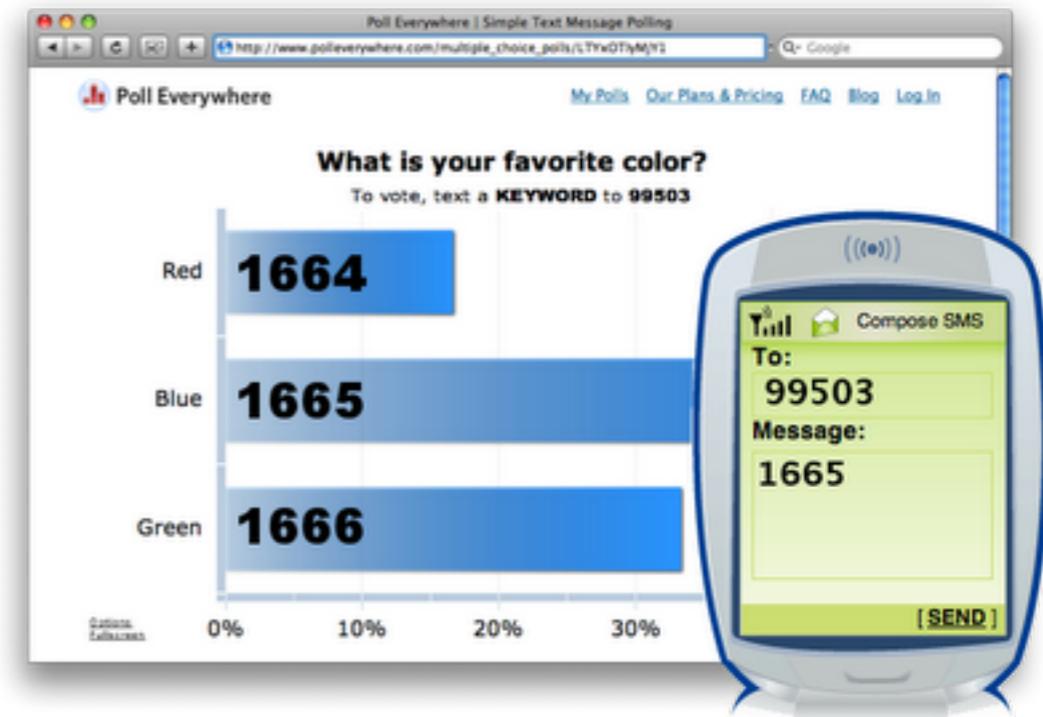
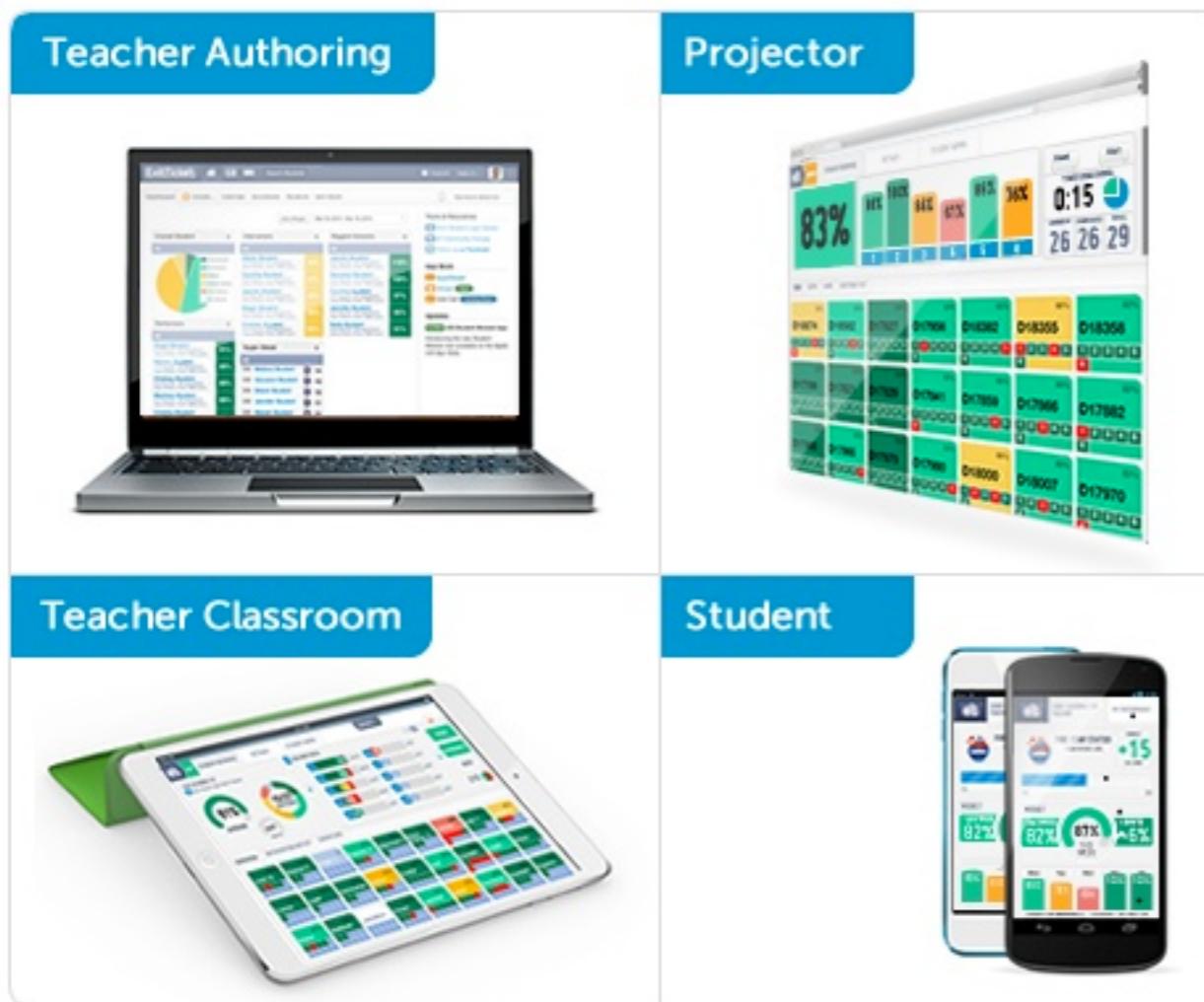
Server

# ECMAScript 6 is on its way (but not for us)



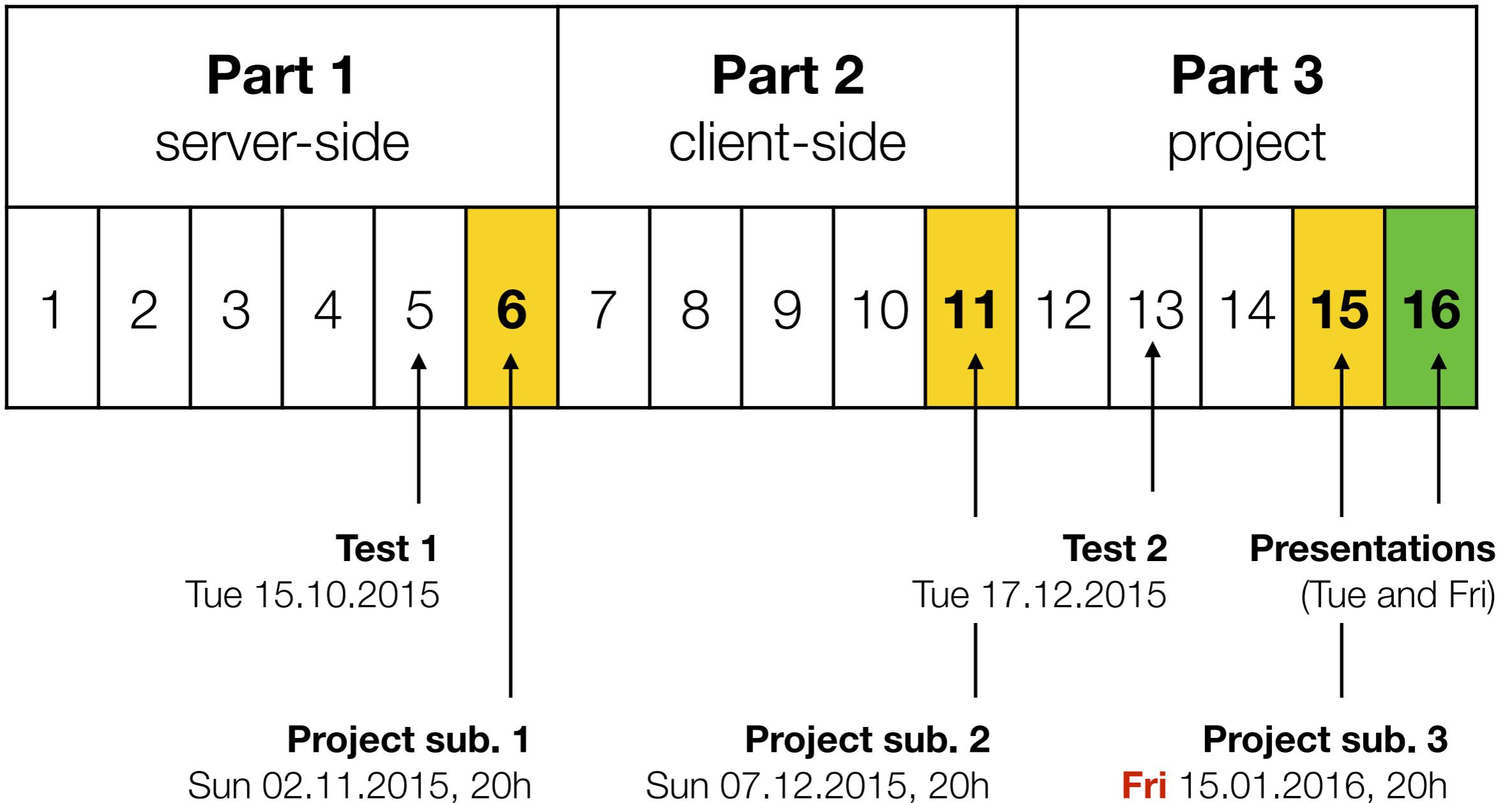
<http://es6-features.org/>

# Project: interactive presentations & polls



Groups: up to 4 students

# Planning



# Evaluation

Theory	Practice	
<b>Test 1</b> 50%	<b>Project part 1</b> 33%	
<b>Test 2</b> 50%	<b>Project part 2</b> 34%	
	<b>Project part 3</b> 33%	
<b>Theory grade 50%</b>	<b>Practice grade 50%</b>	<b>Final Grade</b>



# Quick Poll

# Quick Poll

- **What is your current perception of JavaScript?**
- **Scope**
  - It's a "**toy**" language for creating animations on web pages, but I would not use it for anything "serious".
  - It's a **very powerful language**. It is essential on the client side, but it is also really interesting on the server side.

# Quick Poll

- **What is your current perception of JavaScript?**
- **Personal taste**
  - I hate it.
  - I am not a big fan.
  - It's kind of interesting.
  - I love it.
  - I don't care.

# Quick Poll

- **What is your current perception of JavaScript?**
- **Relationship to Java**
  - It's Java, with a few syntactic differences.
  - It has nothing to do with Java, except for some common syntax.

# Quick Poll

- **What is your current perception of JavaScript?**
- **Current knowledge**
  - **Novice:** I may have hacked a few scripts on web pages, but mostly by copy-pasting examples and without fully understanding the language (what is a prototype?).
  - **Intermediate:** I have used JavaScript quite a bit. I can describe the object-oriented model, I understand what a constructor is and how it works. I have quite a bit of experience with JQuery and other libraries. I am always working with a debugger.
  - **Expert:** closures and modules have no secret for me, I have read "JavaScript: the good parts". I have designed my development workflow with yeoman, grunt, bower and a few other tools. I know who Paul Irish is.



heig-vd

Haute Ecole d'Ingénierie et de Gestion  
du Canton de Vaud

# Douglas Crockford: JavaScript: The Good Parts

[https://www.youtube.com/watch?v= DKkVvOt6dk](https://www.youtube.com/watch?v=DKkVvOt6dk)

# JavaScript is built on some very good ideas and a few very bad ones.

heig-vd

Haute Ecole d'Ingénierie et de Gestion  
du Canton de Vaud

**JavaScript is an important language** because it is the language of the web browser. Its association with the browser makes it one of the most popular programming languages in the world. **At the same time, it is one of the most despised programming languages in the world.** [...]

Most people in that situation **don't even bother to learn JavaScript first**, and then they are surprised when JavaScript turns out to have significant differences from the some other language they would rather be using, and that those differences matter.

The amazing thing about JavaScript is that it is possible to get work done with it without knowing much about the language, or even knowing much about programming. It is a language with enormous expressive power. It is even better when you know what you're doing. **Programming is difficult business. It should never be undertaken in ignorance.**

*Unearthing the Excellence in JavaScript*



Douglas Crockford

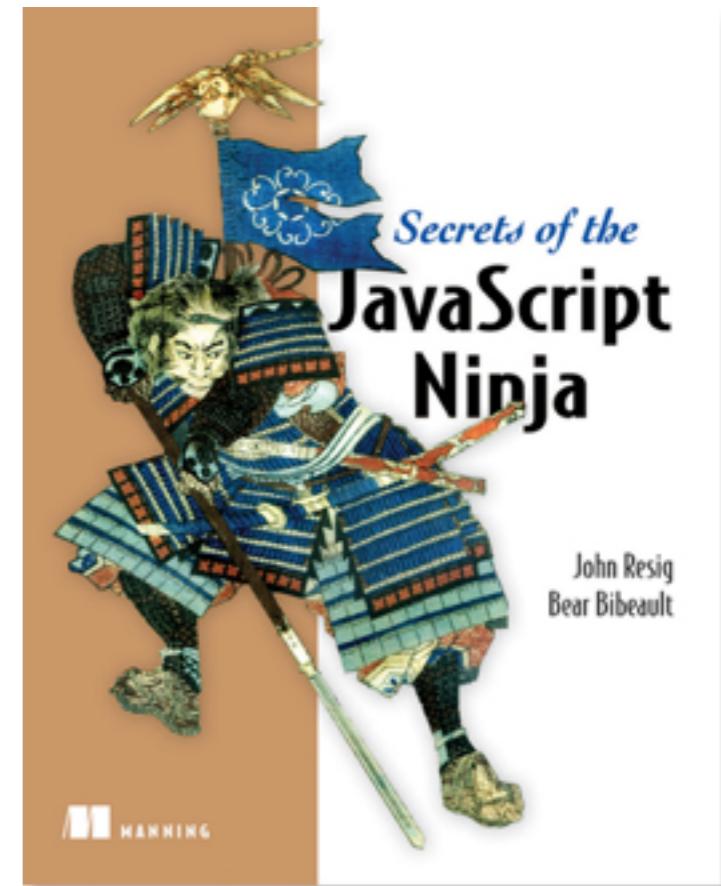
# JavaScript is important. That wasn't always so, but it's true now.

**Web applications are expected to give users a rich user interface experience**, and without JavaScript, you might as well just be showing pictures of kittens. More than ever, web developers need to have a sound grasp of the language that brings life to web applications.

And like orange juice and breakfast, **JavaScript isn't just for browsers anymore**. The language has knocked down the walls of the browser and is being **used on the server** in engines such as Rhino and V8, and in frameworks like Node.js.

Although this book is primarily focused on JavaScript for web applications, the fundamentals of the language presented in part 2 of this book are applicable across the board.

With more and more developers using JavaScript, it's now more important than ever that they **grasp its fundamentals**, so that they can become true ninjas of the language.





# Tools, part 1

# Experimenting with JavaScript

*What do I need to write, execute and debug JavaScript code?*

- Should I work on the **server side** or on the **client side**?
- Should I use a **simple text editor** or a **complete IDE**?
- Should I rather use an **online programming environment**?
- What kinds of **developer tools**, such as debuggers, are available?

# Browsers and Developers Tools

---

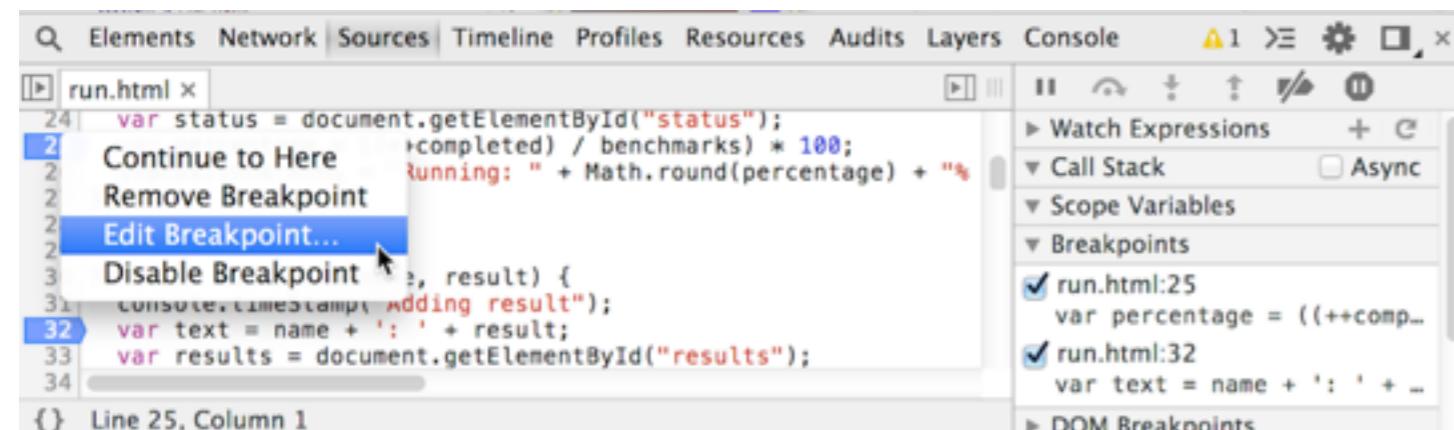


# Browsers and Developers Tools

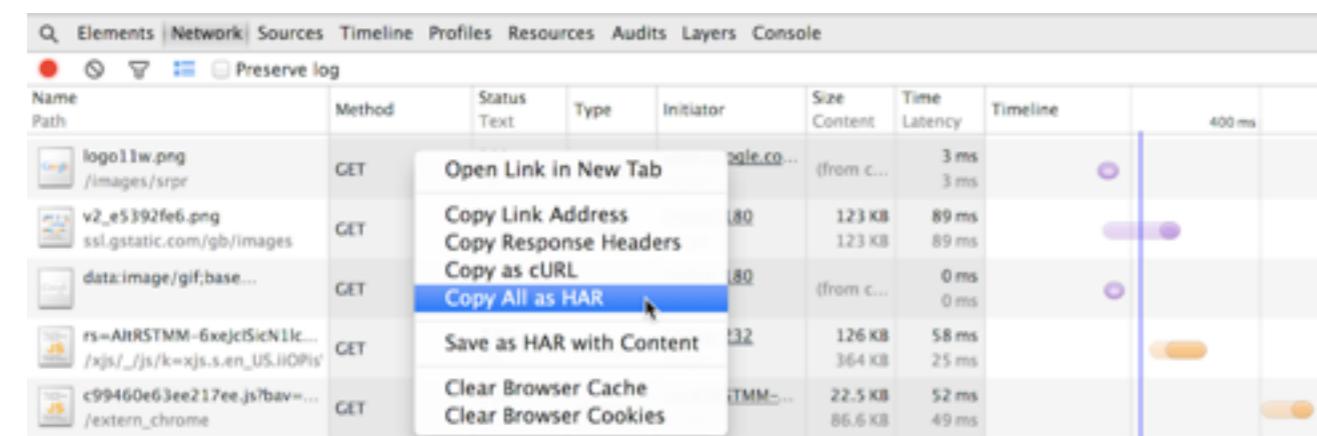
## Chrome DevTools Overview

The Chrome Developer Tools (DevTools for short), are a set web authoring and debugging tools built into Google Chrome. The DevTools provide web developers deep access into the internals of the browser and their web application. Use the DevTools to efficiently track down layout issues, set JavaScript breakpoints, and get insights for code optimization.

Note: If you are a web developer and want to get the latest version of DevTools, you should use [Google Chrome Canary](#).



A screenshot of the Chrome DevTools Sources tab. The left pane shows a portion of a JavaScript file named 'run.html' with several lines of code. Lines 24, 25, and 32 have breakpoints set, indicated by blue numbers and red highlights. The right pane shows the Breakpoints sidebar, which lists the three breakpoints with their file names and line numbers. The sidebar also includes sections for Watch Expressions, Call Stack, Scope Variables, and DOM Breakpoints.



A screenshot of the Chrome DevTools Network tab. It displays a table of network requests with columns for Name, Method, Status, Type, Initiator, Size, Time, and Timeline. A context menu is open over a row for a 'Copy All as HAR' request, with options like Open Link in New Tab, Copy Link Address, Copy Response Headers, Copy as cURL, and Save as HAR with Content.

Name	Method	Status	Type	Initiator	Size	Time	Timeline
log01w.png /images/srp...	GET	Open Link in New Tab	single.co...	(from c...	3 ms	3 ms	
v2_e5392fe6.png ssl.gstatic.com/gb/images	GET	Copy Link Address	80	123 KB	89 ms		
data:image/gif;base...	GET	Copy Response Headers	80	123 KB	89 ms		
	GET	Copy as cURL	80	0 ms	0 ms		
	GET	Copy All as HAR	80	(from c...	0 ms		
	GET	Save as HAR with Content	132	126 KB	58 ms		
	GET	Clear Browser Cache	132	364 KB	25 ms		
c99460e63ee217ee.js?bav=...	GET	Clear Browser Cookies	132	22.5 KB	52 ms		
				86.6 KB	49 ms		

<https://developer.chrome.com/devtools>



**How to access the DevTools**

**The DevTools window**

**Inspecting the DOM and styles**

**Working with the console**

**Debugging JavaScript**

**Improving network performance**

**Audits**

**Improving rendering performance**

**JavaScript CSS performance**

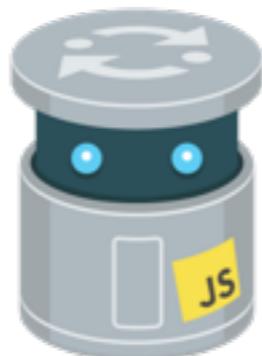
**Inspecting storage**

**Further reading**

**Further resources**

## *Setup 1: Online Programming Environment*

- There are several online environments, which can be used to write, execute, debug and **share** JavaScript programs.
- Use your browser and **developer tools** within your browser.



# Experimenting with JavaScript

The screenshot shows the JSFiddle interface with the following details:

- Frameworks & Extensions:** jQuery 2.1.0 (selected), Bootstrap 3.2.0 (checked).
- onDomready:**

```
// this function will be called when the DOM is ready
$(function() {
  $("h2").html("...by this text");
  console.log("hello console");
});
```
- HTML:**

```
<div class="container">
  <h1>Sandbox</h1>
  <div class="panel panel-default">
    <div class="panel-body">
      <h2>This will be replaced...</h2>
      A simple demo of Bootstrap and JQuery.
    </div>
  </div>
</div>
```
- CSS:** None.
- Result:** Displays "Sandbox" and "...by this text".
- Bottom Left:** Keyboard shortcuts icon.

# Experimenting with JavaScript

The screenshot shows the JS Bin interface. On the left, there's a sidebar with a bin icon, 'New bin' and 'Open bin...' buttons, and a checkbox for 'Textarea editor mode'. The main area has tabs for 'HTML', 'CSS', 'JavaScript', 'Console', and 'Output'. The 'JavaScript' tab is active, displaying the following code:

```
// this function will be called
when the DOM is ready
$(function() {
  $("h2").html("...by this
text");
  console.log("hello console");
});
```

The 'Output' tab shows the result: **Sandbox** with the text "...by this text". Below it, a note says "A simple demo of Bootstrap and JQuery." A 'Bin info' button at the bottom right indicates the bin was created "just now".

# Experimenting with JavaScript

The screenshot shows the CodePen interface with the following details:

- Header:** CodePen – A Pen by Olivier
- URL:** codepen.io/wasadigi/pen/imadn
- Editor:** Editor tab selected.
- HTML:** Contains a simple Bootstrap and JQuery demo.
- CSS:** Empty.
- JS:** Contains a JQuery script that replaces an h2 element with "...by this text" and logs "hello console".
- Preview:** Shows the "Sandbox" page with the updated content.
- Bottom:** Navigation buttons (Collections, Embed, Details & Comments, Delete), a save message ("Last saved 1 minute ago"), a keyboard icon, and a full-screen icon.

```
1<div class="container">
2  <h1>Sandbox</h1>
3  <div class="panel panel-default">
4    <div class="panel-body">
5      <h2>This will be replaced...</h2>
6      A simple demo of Bootstrap and
7      JQuery.
8    </div>
9  </div>
9</div>
```

```
1// this function will be called when the
2DOM is ready
2$(function() {
3  $("h2").html("...by this text");
4  console.log("hello console");
5});
```

**Sandbox**

...by this text

A simple demo of Bootstrap and JQuery.

# Experimenting with JavaScript

---



Codepen Vs JSFiddle Vs CSSDeck Vs Liveweave Vs JSBin Vs Dabblet

January 13, 2014

<http://mediaunmasked.com/reviews/software/codepen-jsfiddle-cssdeck-liveweave-jsbin-dabblet/>

## *Setup 2: IDE + Browser + Dev Tools*

- Select an editor to write the JavaScript (and HTML/CSS) code.
- **Configure the editor with “JSLint” capabilities**
- Select one browser.
- Use developer tools within the browser.

# Some of the Available Editors...

---



TextMate



Komodo IDE



Sublime Text



Cloud9 IDE



NetBeans



# JavaScript 101 (Part 1)

# JavaScript 101, Part 1

- Types
- Scopes
- Objects
- Prototypal inheritance
- Functions
- Constructors
- Arrays

# Rule #1

## JavaScript defines 6 types

```
var aNumber = 3.12;
var aBoolean = true;
var aString = "HEIG-VD";
var anObject = {
  aProperty: null
};

// t is true for all of these:
var t;
t = typeof aNumber === "number";
t = typeof aBoolean === "boolean";
t = typeof aString === "string";
t = typeof anObject === "object";
t = typeof anObject.aProperty ===
  "object";
t = typeof anObject.foobar ===
  "undefined";
```

- The 6 types are:
  - number
  - boolean
  - string
  - object
  - undefined
  - null
- null is a type, but `typeof null === object`.
- JavaScript is a dynamic language: when you declare a variable, you don't specify a type (and the type can change over time).

## Rule #2

There are 2 scopes for variables:

the (evil) global scope and the function scope

```
var aVariableInGlobalScope;

function myFunction() {
    var aVariableInFunctionScope;
    anotherVariableInGlobalScope;
}

function myFunction2() {
    for (i=0; i<10; i++) {
        //i is in global scope!
    }
    for (var j=0; j<10; j++) {
        //j is in function scope!
    }
}
```

- A variable declared within a function is **not accessible** outside this function.
- Unless using **strict mode**, it is not mandatory to declare variables (beware of typos...)
- Two scripts loaded from the same HTML page share the same global scope (beware of **conflicts...**).
- There is **no block scope**.

# Rule #3

## Objects are dynamic bags of properties

```
// let's create an object
var person = {
  firstName: 'olivier',
  lastName: 'liechti'
};

// we can dynamically add properties
person.gender = 'male';
person['zip'] = 1446;

// and delete them
delete person.zip;

for (var key in person) {
  console.log(key + " : " +
person[key]);
};
```

- There are different ways to **access properties** of an object.
- JavaScript is **dynamic**: it is possible to **add** and **remove** properties to an object at any time.
- Every object has a different list of properties (**no class**).

## Rule #4

# The language has no support for classes

## There are 3 ways to create objects

```
// create an object with a literal
var person = {
  firstName: 'olivier',
  lastName: 'liechti'
};

// create an object with a prototype
var child = Object.create(person);

// create an object with a constructor
var child = new Person('olivier',
  'liechti');
```

- **class** is a reserved word in JavaScript, but it is not used in the current version of the language (reserved for the future).
- A **constructor** is function like any other (uppercase is a coding convention).
- It is the use of the **new** keyword that triggers the object creation process.

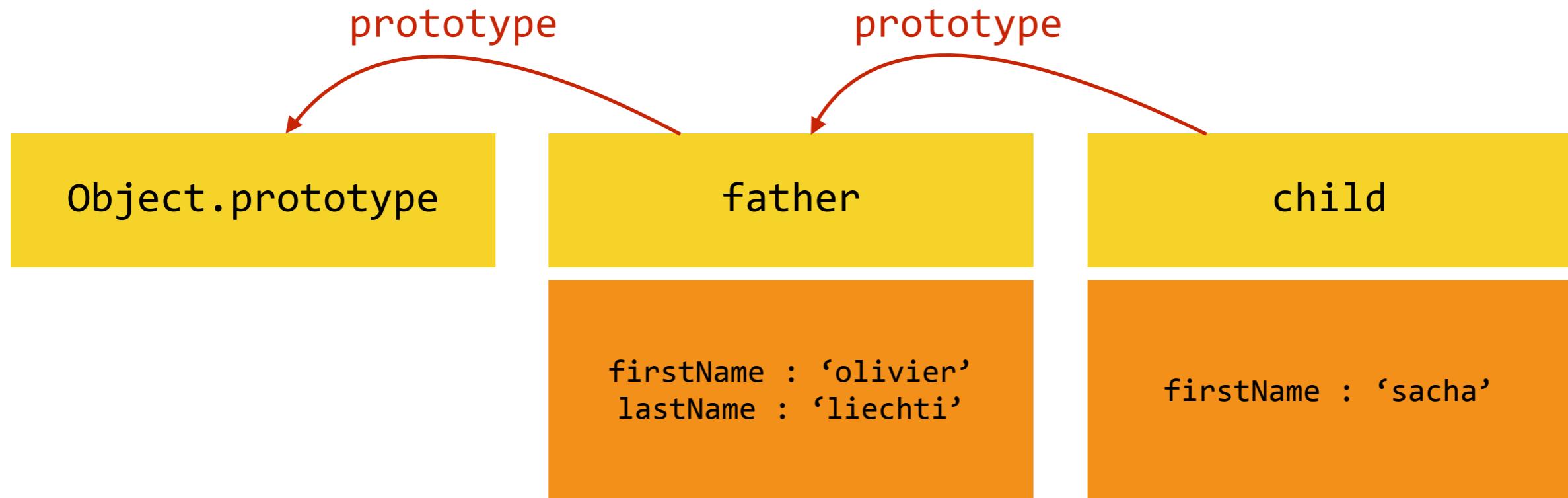
# Rule #5

## Every object inherits from a prototype object

```
var person = {  
    firstName: "olivier",  
    lastName: "liechti"  
};  
// person's prototype is Object.prototype  
  
var father = {};  
var child = Object.create(father);  
// child's prototype is father  
  
function Person(fn, ln) {  
    this.firstName = fn;  
    this.lastName = ln;  
}  
var john = new Person("John", "Doe");  
// john's prototype is Person.prototype
```

## Rule #5

# Every object inherits from a prototype object



```
console.log(child.lastName);
// prints 'liechti' on the
console
```

- Every object inherits from a prototype object.  
**It inherits and can override its properties**, including its methods.
- Objects created with object literals inherit from **Object.prototype**.
- When you access the property of an object, JavaScript **looks up the prototype chain** until it finds an ancestor that has a value for this property.

# Rule #6

## With patterns, it is possible to implement class-like data structures

```
function Person(fn, ln) {  
    var privateVar;  
    this.firstName = fn;  
    this.lastName = ln;  
    this.badGreet = function() {  
        console.log("Hi " + this.firstName);  
    };  
};  
  
Person.prototype.greet = function() {  
    console.log("Hey " + this.firstName);  
};  
  
var p1 = new Person("olivier", "liechti");  
  
p1.badGreet();  
p1.greet();
```

- **badGreet** is a property that will be replicated for every object created with the Person constructor:
  - poor memory management
  - not possible to alter behavior of all instances at once
- **greet** is a property that will be shared by all instances (because it will be looked up along the object inheritance chain).
- **privateVar** is not accessible outside of the constructor.
- **firstName** is publicly accessible (no encapsulation).

# Rule #7

## Arrays are objects

```
var fruits = ["apple", "pear"];

fruits.push("banana");
console.log(Object.getPrototypeOf(fruits));

for (var i=0; i<fruits.length; i++) {
  console.log("fruits[" + i + "] = " + fruits[i]);
}

var transformedFruits = fruits.map( function(fruit) {
  return fruit.toUpperCase();
});
transformedFruits.forEach( function(fruit) {
  console.log(fruit);
});

var count = fruits.reduce( function(val, fruit) {
  console.log("reducer invoked with " + val);
  return val+1;
}, 0);
console.log("There are " + count + " fruits in the array");
```



# JavaScript/CSS Libraries

Bootstrap

getbootstrap.com/2.3.2/index.html

Home Get started Scaffolding Base CSS Components JavaScript Customize Bootstrap

# Bootstrap

Sleek, intuitive, and powerful front-end framework for faster and easier web development.

[Download Bootstrap](#)

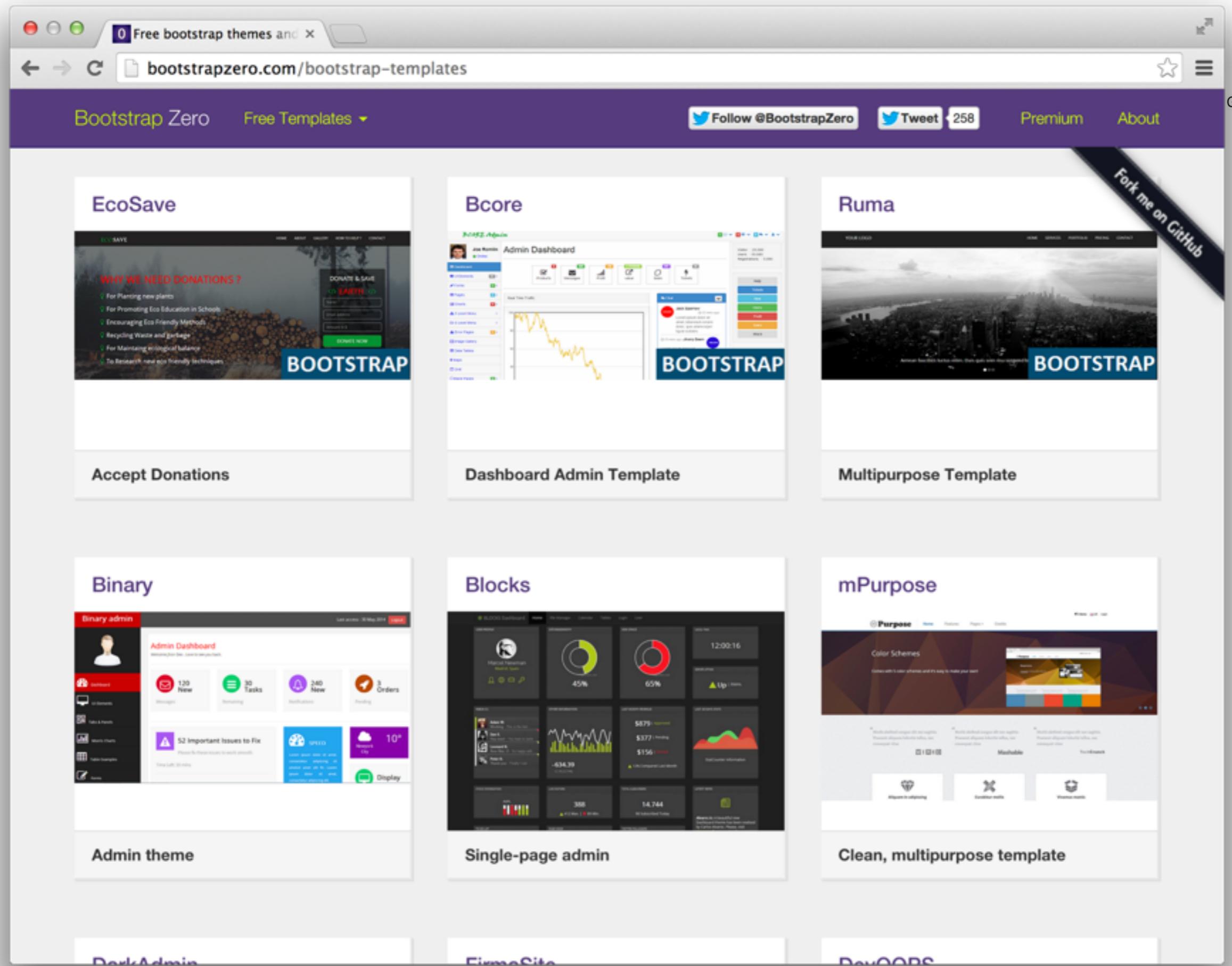
[GitHub project](#) [Examples](#) [Extend](#) [Version 2.3.2](#)

 HACK  
REACTOR  
The 1st Remote  
Immersive JavaScript  
Coding Program

[Apply](#)

Previously Only in San Francisco, attend the "Harvard" of coding schools wherever you are.

ads via Carbon



WrapBootstrap - Bootstrap

https://wrapbootstrap.com

win a free theme in our weekly raffle! Tweet 10.6K a random tweeter is selected each week close

{wrap}bootstrap Themes Logos Selling Bounties Bootswatch ➔ Sign in Sign up

Select a Category Enter search terms Search

# Bootstrap Themes & Templates

Bootstrap is an HTML5 & CSS3 framework designed to help you kickstart the development of webapps and sites. WrapBootstrap is a marketplace for premium Bootstrap themes and templates. Impress your clients and visitors while using a single, rock-solid foundation.

[View all the themes »](#) Or make them! [Sell your designs](#)

[Share](#)  [Tweet](#)

19.9k Followers 22.9k Followers 757 Subscribers 36.5k Subscribers

See new themes weekly:

Email address [Subscribe](#)

[See all newest on WrapBootstrap »](#)



jQuery is a **fast**, **small**, and feature-rich JavaScript library. It makes things like HTML **document traversal** and **manipulation**, **event handling**, **animation**, and **Ajax** much simpler with an easy-to-use API that works **across a multitude of browsers**.



# Using jQuery Core

- [\\$ vs \\$\(\)](#)
- [\\$\( document \).ready\(\)](#)
- [Avoiding Conflicts with Other Libraries](#)
- [Attributes](#)
- [Selecting Elements](#)
- [Working with Selections](#)
- [Manipulating Elements](#)
- [The jQuery Object](#)
- [Traversing](#)
- [CSS, Styling, & Dimensions](#)
- [Data Methods](#)
- [Utility Methods](#)
- [Iterating over jQuery and non-jQuery Objects](#)
- [Using jQuery's .index\(\) Function](#)

<http://learn.jquery.com/using-jquery-core/>



## Selecting Elements by ID

```
1 | $( "#myId" ); // Note IDs must be unique per page.
```

## Selecting Elements by Class Name

```
1 | $(".myClass");
```

## Selecting Elements by Attribute

```
1 | $("input[name='first_name']); // Beware, this can be very slow in older browsers
```

## Selecting Elements by Compound CSS Selector

```
1 | $("#contents ul.people li");
```



heig-vd

Haute Ecole d'Ingénierie et de Gestion  
du Canton de Vaud

Handlebars.js is an extension to the Mustache **templating language** created by Chris Wanstrath.

Handlebars.js and Mustache are both **logicless templating languages that keep the view and the code separated** like we all know they should be.



heig-vd

Haute Ecole d'Ingénierie et de Gestion  
du Canton de Vaud

```
Hello  
{firstName}  
!
```

template

```
var source = $("#student-template").html();  
var template = Handlebars.compile(source);  
var student = {firstName: 'Olivier'};  
var html = template(student);
```

```
{  
  firstName: "Olivier",  
  lastName: "Liechti"  
}
```

object

```
Hello  
Olivier!
```

HTML

# underscore.js

heig-vd

Haute Ecole d'Ingénierie et de Gestion  
du Canton de Vaud

Underscore is a JavaScript library that provides a whole mess of useful **functional programming helpers** without extending any built-in objects.

It's the answer to the question: "If I sit down in front of a blank HTML page, and want to start being productive immediately, what do I need?" ... and **the tie to go along with jQuery's tux and Backbone's suspenders**.

The screenshot shows a web browser window with the title bar "underscore.js" and the address bar "underscorejs.org/docs/underscore.html". The main content area displays the "UNDERSCORE.JS" documentation. On the left side, there is a sidebar with the following information:

- Underscore.js 1.7.0**
- <http://underscorejs.org>
- (c) 2009–2014 Jeremy Ashkenas, DocumentCloud and Investigative Reporters & Editors
- Underscore may be freely distributed under the MIT license.

The main content area has a section titled "BASELINE SETUP" which includes the following instructions:

- Establish the root object, `window` in the browser, or `exports` on the server.
- Save the previous value of the `_` variable.
- Save bytes in the minified (but not gzipped) version:
- Create quick reference variables for speed access to core prototypes.
- All **ECMAScript 5** native function implementations that we hope to use are declared here.

On the right side of the page, the source code for Underscore.js is shown. It starts with a self-executing anonymous function:

```
(function() {
```

Inside the function, the code initializes the `root` variable to `this`:

```
    var root = this;
```

It then saves the previous value of the `_` variable:

```
    var previousUnderscore = root._;
```

Next, it saves references to native array and object prototypes:

```
    var ArrayProto = Array.prototype, ObjProto = Object.prototype, FuncProto = Function.prototype;
```

It creates quick reference variables for array prototypes:

```
    var
        push      = ArrayProto.push,
        slice     = ArrayProto.slice,
        concat   = ArrayProto.concat,
        toString  = ObjProto.toString,
        hasOwnProperty = ObjProto.hasOwnProperty;
```

Finally, it saves references to native `isArray` and `keys` functions:

```
    var
        nativeIsArray = Array.isArray,
        nativeKeys    = Object.keys;
```

**heig-vd**

Haute Ecole d'Ingénierie et de Gestion  
du Canton de Vaud



# Lab 01

[https://www.youtube.com/watch?](https://www.youtube.com/watch?v=F6HcVrZTaVs&list=PLfKkysTy70Qa3LHon4n0RWAuAyEiMzDkO&index=1)

v=F6HcVrZTaVs&list=PLfKkysTy70Qa3LHon4n0RWAuAyEiMzDkO&index=1

color is mapped  
to gender

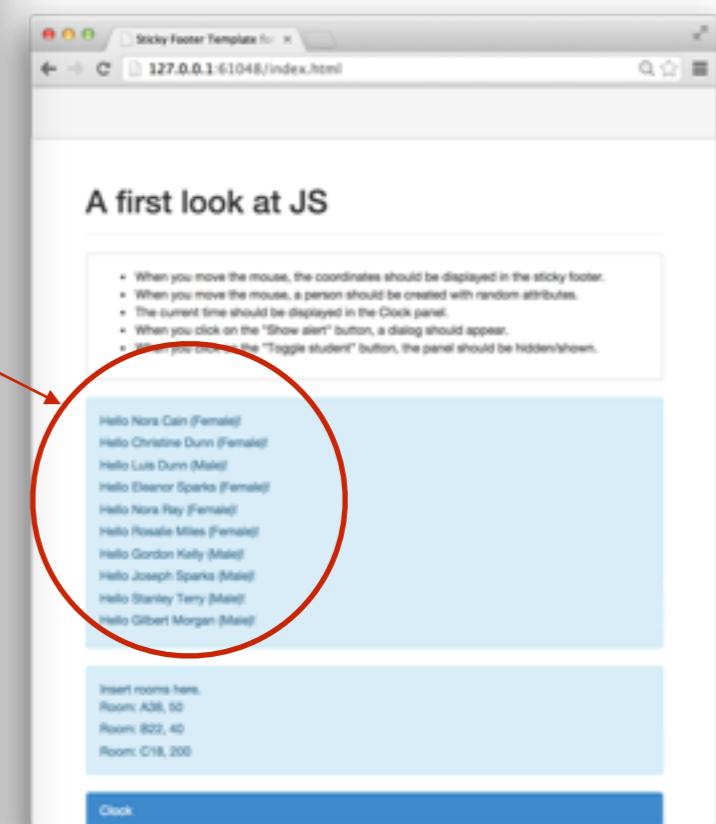
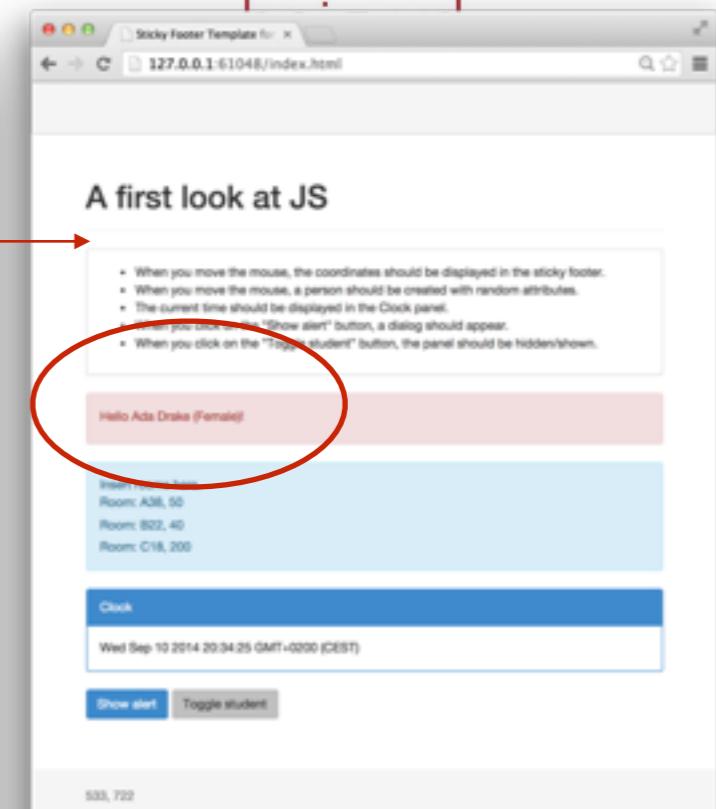
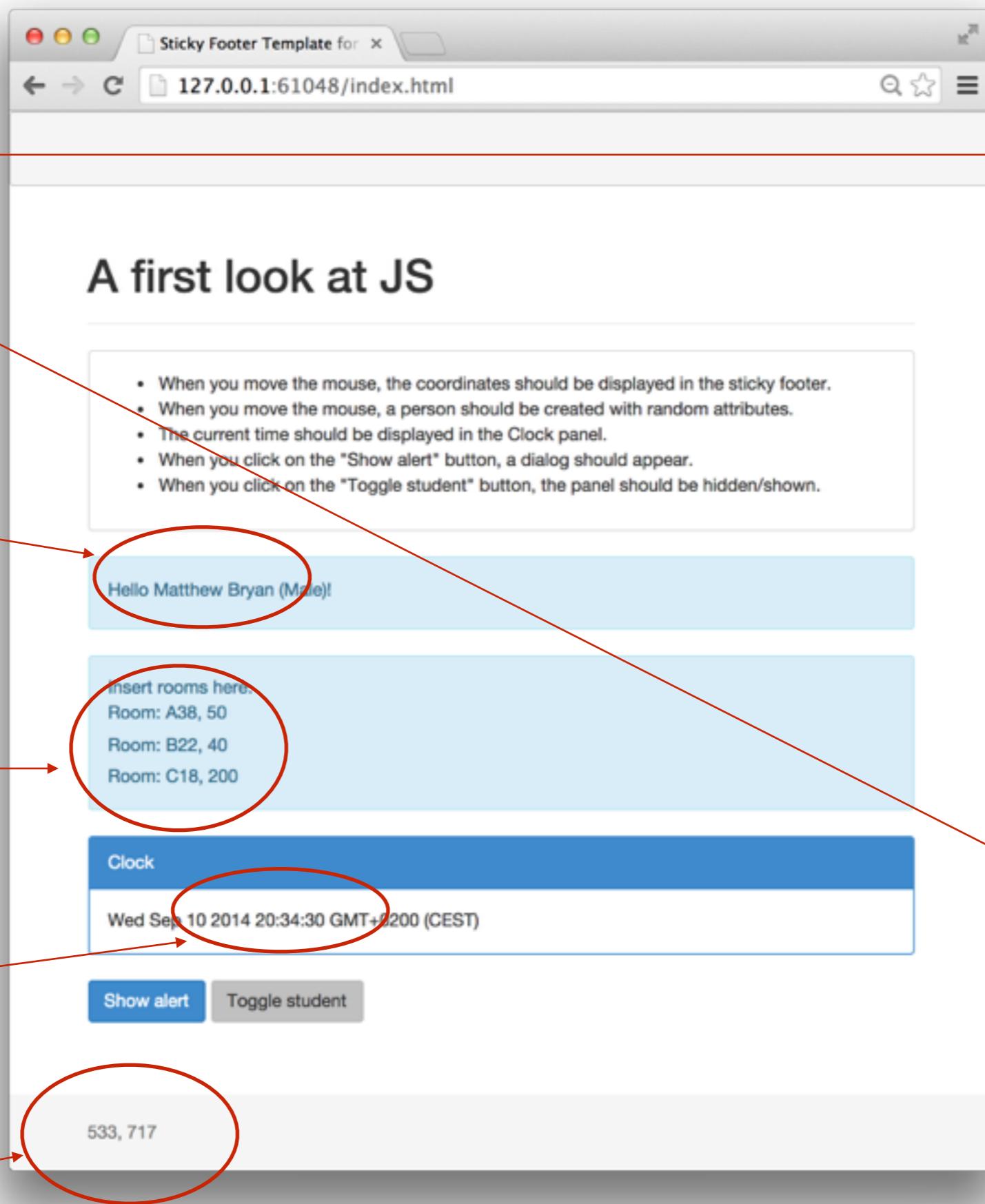
pressing SHIFT  
adds lines

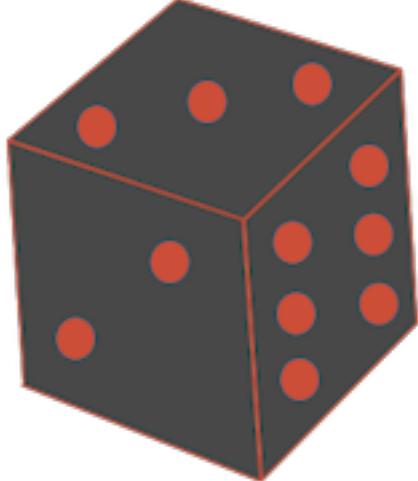
data is  
generated  
randomly when  
mouse moves

data is fetched  
via AJAX

clock is updated  
regularly

mouse  
coordinates are  
updated





# Chance

heig-vd

Haute Ecole d'Ingénierie et de Gestion  
du Canton de Vaud

Chance is a **minimalist generator of random strings, numbers, etc.** to help reduce some monotony particularly while writing automated tests or anywhere else you need anything random.

GitHub Pages

https://pages.github.com

Pages Help

# GitHub Pages

## Websites for you and your projects.

Hosted directly from your [GitHub repository](#). Just edit, push, and your changes are live.

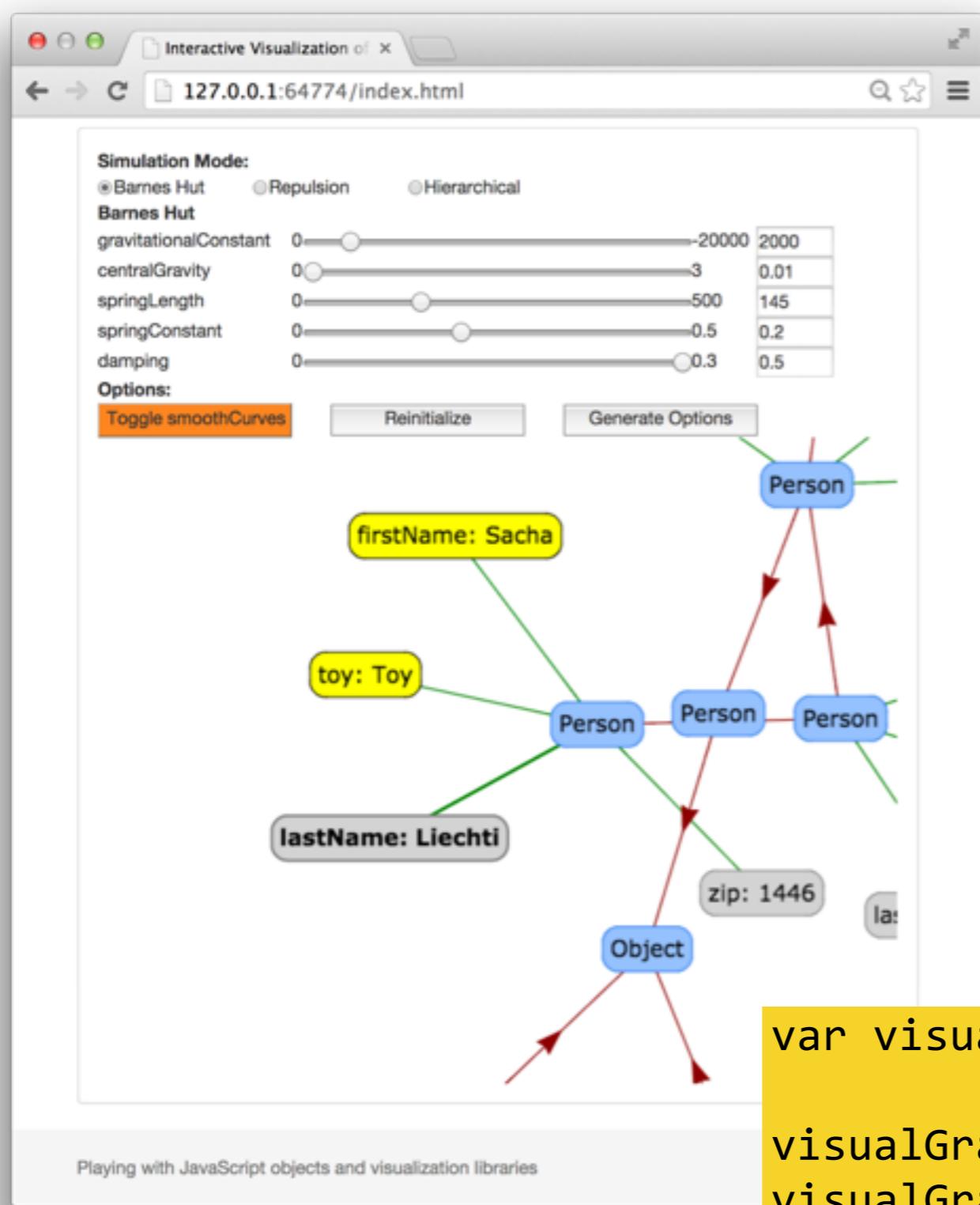
The screenshot shows a GitHub Pages site for the 'Foundation' project. The page title is 'Foundation | The Most Advanced Responsive Front-end Framework'. The main heading is 'Foundation'. Below it is a bullet-point list: 'The most advanced responsive front-end framework in the world.' A blue button labeled 'Download Foundation 5' is present. At the bottom, there's a star icon followed by '14.9k stargazers' and a Twitter icon followed by '@foundationzurb'. To the right of the desktop view is a mobile phone displaying the same Foundation website, and further right is a portion of a laptop screen showing the same content.

**heig-vd**

Haute Ecole d'Ingénierie et de Gestion  
du Canton de Vaud



# Lab 02



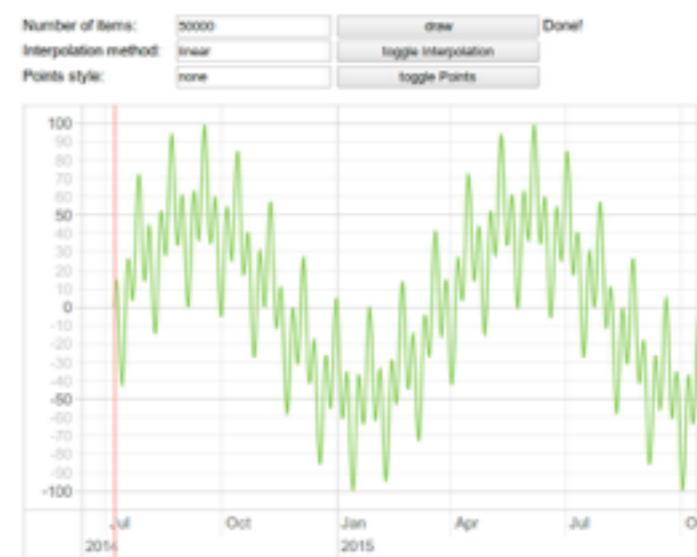
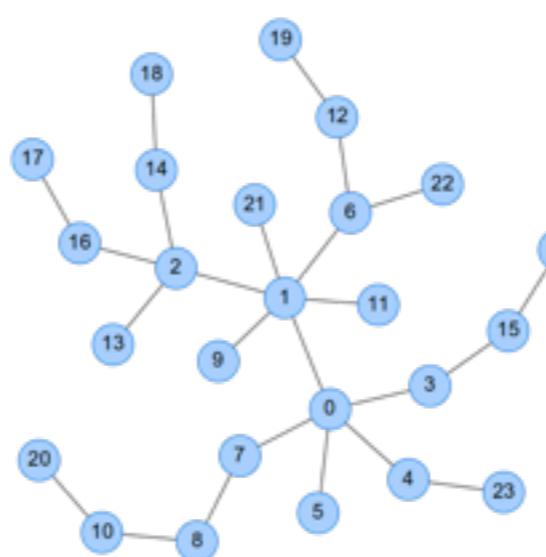
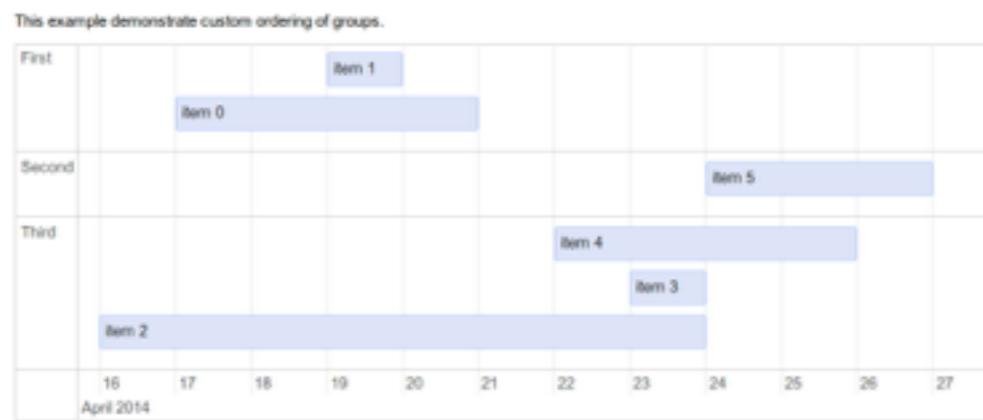
[https://www.youtube.com/watch?  
v=3VoYI-  
HHfz0&list=PLfKkysTy70Qa3LHon4n  
0RWaAuAyEiMzDkO&index=2](https://www.youtube.com/watch?v=3VoYI-HHfz0&list=PLfKkysTy70Qa3LHon4n0RWaAuAyEiMzDkO&index=2)

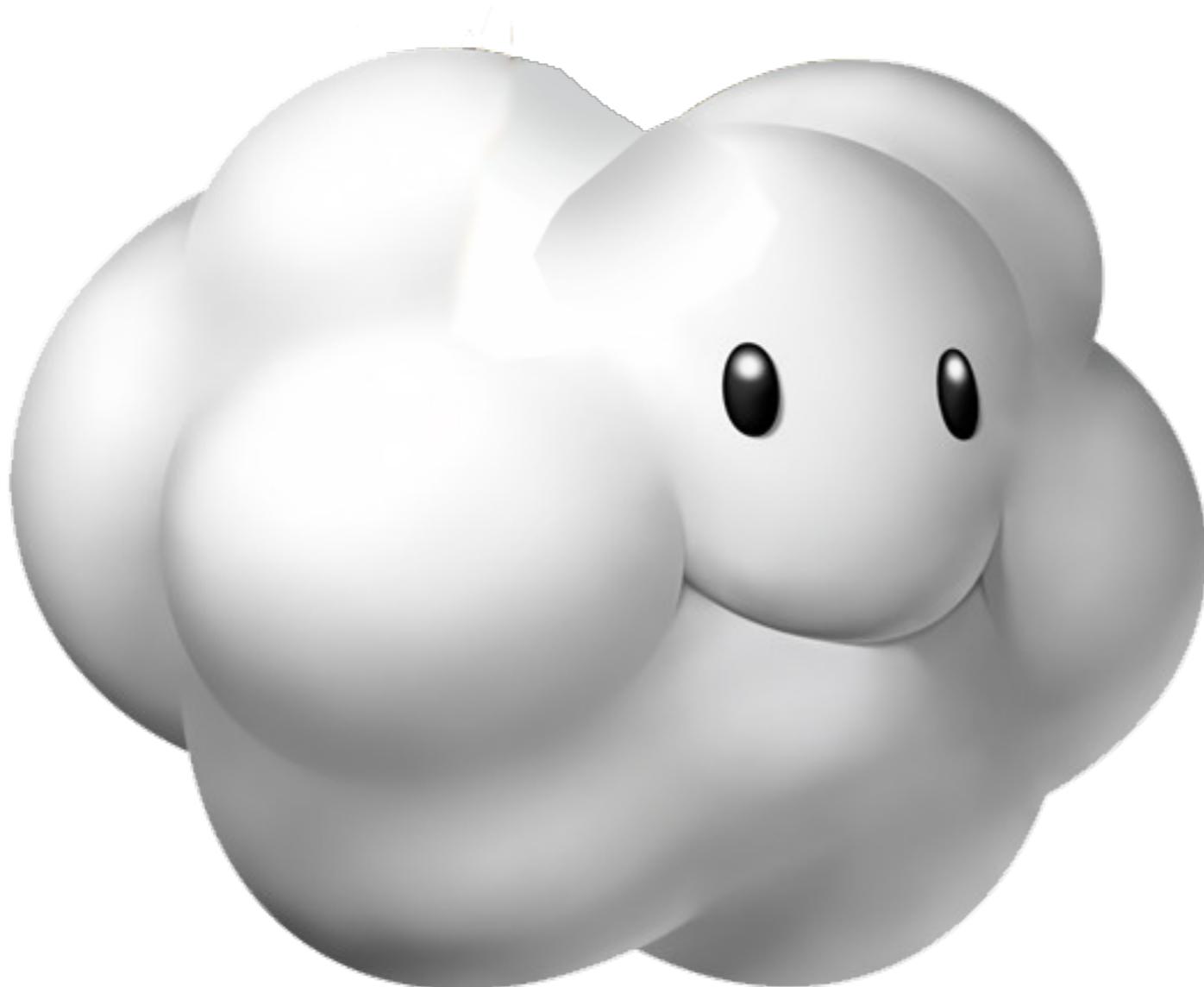
```
var visualGraph3 = new jsvis.VisualizationGraph();

visualGraph3.addJsObjectToGraph(sacha);
visualGraph3.addJsObjectToGraph(sacha.toy);
visualGraph3.addJsObjectToGraph(stephan);
visualGraph3.addJsObjectToGraph(akebono);
```



**Vis.js** is a dynamic, **browser based visualization** library. The library is designed to be easy to use, to handle large amounts of dynamic data, and to enable manipulation of and interaction with the data.





heig-vd

Haute Ecole d'Ingénierie et de Gestion  
du Canton de Vaud

# Welcome to The Cloud

tweb-demo - Resources | Help

Heroku, Inc. [US] https://dashboard.heroku.com/apps/tweb-demo/resources

Dashboard Apps tweb-demo ★

olivier.liechti@wasabi-tech... ▾

Resources Deploy Metrics Activity Access Settings

FAVORITES

★ Favorite any app to pin it here in the sidebar

Personal Apps

Free dynos

Edit

web vendor/bin/heroku-php-apache2 \$0.00

Add-ons FIND MORE ADD-ONS

Quickly add add-ons from Elements

You haven't added any add-ons yet

Add-ons provide tools and services for developing, extending and operating your app

Estimated Monthly Cost \$0.00

LATEST FROM THE BLOG

Introducing Heroku Private Spaces: Private PaaS, delivered as-a-Service

heroku.com/home Blog Careers Privacy Policy Feedback

# Trick to deploy static content on heroku



heroku

```
$ heroku login
$ git init
$ heroku create tweb-demo
$ echo "<h2>hello. i am on the cloud.</h2>" > home.html
$ echo "<?php include_once('home.html'); ?>" > index.php
$ echo "{}" > composer.json

$ git add *
$ git commit -m "My first static site on heroku"
$ git push heroku master

$ heroku logs
```

JS

# JavaScript 101 (Part 2)

# JavaScript 101, Part 2

- Functions are objects
- Function.prototype vs myFunction.prototype
- Closures
- Module patterns
- this

# Rule #8

## Functions are objects

```
function aFunction() {  
}  
  
var f = function() {  
}  
  
var g = function g() {  
    g(); // recursive call  
}  
  
var h = function(functionParam) {  
    functionParam();  
}  
  
h(f);  
h(g);
```

## Rule #9

**Whenever a function is defined, an object is created.**

Its prototype is Function.prototype.

Its prototype property is the object that will be the prototype of instances created with the function used as a constructor with the **new** keyword.

```
function Robot() {  
}  
var r2d2 = new Robot();  
  
var t1 = Object.getPrototypeOf(Robot) === Function.prototype;  
var t2 = Robot.__proto__ === Function.prototype; //  
deprecated  
  
var t3 = Object.getPrototypeOf(r2d2) === Robot.prototype;  
var t4 = Object.getPrototypeOf(Robot.prototype) ===  
Object.prototype;
```

## Rule #10

# "new" makes constructors behave in a special way, but constructors are normal functions

```
function Animal(name) {  
    this.name = name;  
}  
var cat = new Animal("Félix");
```

1. A **new object** is created.
2. Its **prototype** is set to `Animal.prototype`
3. The constructor function is called; **this** is bound to the newly created object.
4. In general, the constructor does not return any object. In this case, the result of the `new` expression is the newly created object.

# Rule #11

## Functions can be nested

```
function f1(p1) {  
    console.log("f1 can see " + p1);  
    function f2(p2) {  
        console.log("f2 can see " + p2 + " " + p1);  
        function f3(p3) {  
            console.log("f3 can see " + p3 + " " + p2 + " " + p1);  
        }  
        f3(3);  
    }  
    f2(2);  
}  
f1(1);
```

- An **object** is created for every function.
- Each function has access to variables defined in the **parent** functions (an in the **global scope**).

# Rule #12

## A closure is formed when a nested function accesses a *free variable*

```
function f1(p1) {  
    console.log("f1 can see " + p1);  
    function f2(p2) {  
        console.log("f2 can see " + p2 + " " + p1);  
        function f3(p3) {  
            console.log("f3 can see " + p3 + " " + p2 + " " + p1);  
        }  
        f3(3);  
    }  
    f2(2);  
}  
f1(1);
```

- In a function, a **free variable** is a variable that is neither a local variable, nor a parameter of the function.
- A **closure** is the combination of a code block (the function code) and saved parent environments.

```
▼ function f3(p3) { console.log("f3 can see " + p3 + " " + p2 + " " + p1); }  
  arguments: null  
  caller: null  
  length: 1  
  name: "f3"  
  ▶ prototype: f3  
  ▶ __proto__: function Empty() {}  
  ▶ <function scope>  
    ▼ Closure  
      p2: 2  
    ▼ Closure  
      p1: 1  
    ▶ Global: Window
```

# Rule #13

## When 2 functions are created in the same scope, closures are formed on the same environment.

```
function parent() {  
  
    var sharedVariable;  
  
    function child1() {  
        sharedVariable = 1;  
    }  
  
    function child2() {  
        sharedVariable = 2;  
    }  
  
    function getValue() {  
        return sharedVariable;  
    }  
  
    return {  
        toggleToOne: child1,  
        toggleToTwo: child2,  
        getValue: getValue  
    }  
}  
  
var myToggle1 = parent();  
var myToggle2 = parent();  
myToggle1.toggleToTwo();  
myToggle2.toggleToOne();  
console.log(myToggle1.getValue());  
console.log(myToggle2.getValue());
```

- This pattern can be used to implement **private variables** in Javascript.
- But remember that in this case, the code of the functions is **cloned** in all instances (unlike when you use `Class.prototype.method`).
- Be aware of a **well known issue** when closures are formed within a loop.

```
function test() {  
    var functions = [];  
  
    for (var i=0; i<10; i++) {  
        functions.push(function() {  
            console.log(i);  
        });  
    };  
  
    return functions;  
}  
  
var fs = test();  
fs.forEach( function(f) { f(); } );
```



- **What is happening?**
- **Why?**
- **How do you fix this?**

# Rule #14

## Patterns are applied to create modules

```
var myModule = (function() {  
  
    var aPrivateVar;  
    var privateFunction_1 = function() {}  
    var privateFunction_2 = function() {}  
  
    return {  
        publicFunction: privateFunction_1  
    }  
  
})();  
  
myModule.publicFunction();
```

<http://codepen.io/wasadigi/full/GxsfC/>

- The function is **immediately invoked**.

- When `privateFunction_1` accesses `aPrivateVar`, a **closure** is formed.
- is **available even after** the immediately invoked function has returned.
- `privateFunction_1` and `privateFunction_2` share the same parent scope.

```
Hello world
> dir(myModule)
▼ Object ⓘ
  ▼ publicFunction: function () {
    arguments: null
    caller: null
    length: 0
    name: ""
    ► prototype: Object
    ► __proto__: function Empty() {}
    ▼ <function scope>
      ▼ Closure
        aPrivateVar: "world"
        ► Global: Window
        ► __proto__: Object
    ◁ undefined
  >
```

# Rule #15

## The value of this depends on the way a function has been called

```
function Robot(name) {  
  this.name = name;  
}
```

```
Robot.prototype.greet = function greet() {  
  debugger;  
  console.log("hello " + this.name);  
  if (this.name === undefined) {  
    return; // oops...  
  }  
  greet(); // no dot notation!  
}
```

```
var r2d2 = new Robot("r2d2");  
r2d2.greet();
```

- When a function is called on an object (i.e. **as a method**), then this refers to the object.
- When a function is called **as a function** (no dot notation), then this refers to the global object).
- There are methods defined on **Function.prototype** to control the value of this: **apply** and **call**.

# References

JavaScript | MDN

Mozilla Foundation [US] https://developer.mozilla.org/en/docs/Web/JavaScript

Sign in with mozilla

MDN MOZILLA DEVELOPER NETWORK

ZONES WEB PLATFORM TOOLS DEMOS CONNECT

MDN > Web technology for developers > JavaScript

LANGUAGES EDIT ⚙️

# JavaScript

▲ HIDE SIDEBAR

SEE ALSO

JavaScript Tutorials:

- ▶ JavaScript Guide
- ▼ Introductory
  - Getting started
  - JavaScript technologies overview
  - Introduction to Object Oriented JavaScript
- ▼ Intermediate
  - A re-introduction to JavaScript
  - JavaScript data structures
  - Equality comparisons and when to use them
- ▶ Advanced

JavaScript® (often shortened to JS) is a lightweight, interpreted, object-oriented language with [first-class functions](#), most known as the scripting language for Web pages, but [used in many non-browser environments](#) as well such as [node.js](#) or [Apache CouchDB](#). It is a [prototype-based](#), multi-paradigm scripting language that is dynamic, and supports object-oriented, imperative, and functional programming styles. Read more [about JavaScript](#).

The JavaScript standard is [ECMAScript](#). As of 2012, all modern browsers fully support ECMAScript 5.1. Older browsers support at least ECMAScript 3. A 6th major revision of the standard is in the works.

This section of the site is dedicated to the JavaScript language itself, the parts that are not specific to Web pages, or other host environments. For information about APIs specific to Web pages, please see [Web APIs](#) and [DOM](#).

JavaScript is not to be confused with the [Java programming language](#). Java is a trademark or registered trademark of Oracle in the U.S. and other countries.

## Tutorials

Learn how to program JavaScript.

## Reference

Browse the complete [JS reference](#) documentation.

### Standard objects

# **MUST READ** for the Tests

- **A re-introduction to JavaScript**
  - [https://developer.mozilla.org/en-US/docs/Web/JavaScript/A\\_re-introduction\\_to\\_JavaScript](https://developer.mozilla.org/en-US/docs/Web/JavaScript/A_re-introduction_to_JavaScript)
- **Inheritance and the prototype chain**
  - [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Inheritance\\_and\\_the\\_prototype\\_chain](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Inheritance_and_the_prototype_chain)
- **Introduction to Object-Oriented JavaScript**
  - [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Introduction\\_to\\_Object-Oriented\\_JavaScript](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Introduction_to_Object-Oriented_JavaScript)

JS JavaScript Objects in Detail X

javascriptissexy.com/javascript-objects-in-detail/

# JavaScript Objects in Detail

January 27 Last Year

Show Modern Dev Ad

## JavaScript.is (Sexy)

Learn everything about modern web application development with JavaScript and HTML5.

Sponsored By:

**Grammar & Writing for Creators**

Improve Quickly and Significantly  
Write Powerful & Eloquent Prose

Bloggers Writers Programmers Designers Other Professionals

RICHARD OH STANLEY

Follow on Twitter

### Recent Posts

▶ Beautiful JavaScript: Easily Create

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 are immutable, they cannot be changed, while objects are mutable.

#### What is an Object

An object is an unordered list of primitive data (and sometimes reference data types) types that are stored as name-value pairs. Each item in the list is called a property (functions are called methods) and each property name has to be unique and can be a string or a number.

Here is a simple object:

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

To reiterate: Think of an object as a list that contains items and each item (a property) in the list is stored by a name-value pair. The property names in the example above are firstName and favoriteAuthor. And the values for each are "Richard" and "Conrad."

ECMA-262 » JavaScript. Th x

dmitrysoshnikov.com/ecmascript/javascript-the-core/

Home About 

Menu

# ECMA-262

by Dmitry Soshnikov

## JavaScript. The core.

 Tweet 285  Like 168  +1 117  Share 79  Share

Read this article in: Japanese, German (version 2), Arabic, Russian, French, Chinese.

1. An object
2. A prototype chain
3. Constructor
4. Execution context stack
5. Execution context
6. Variable object
7. Activation object
8. Scope chain
9. Closures
10. This value
11. Conclusion

This note is an overview and summary of the "ECMA-262-3 in detail" series. Every section contains references to the appropriate matching chapters so you can read them to get a deeper understanding.

Intended audience: experienced programmers, professionals.

We start out by considering the concept of an *object*, which is fundamental to ECMAScript.

### An object

ECMAScript, being a highly-abstracted object-oriented language, deals with *objects*. There are also *primitives*, but they, when needed, are also converted to objects.

An object is a *collection of properties* and has a *single prototype object*. The prototype may be either an object or the *null* value.

Search...

### Articles

- ES6 Notes: Default values of parameters
- Pattern Matching
- Essentials of Interpretation. Checkpoint: part 1
- Essentials of Interpretation. Intro.
- ECMA-262-5 in detail. Chapter 3.2. Lexical environments: ECMAScript implementation.

### Comments

-  micha-s on Note 2. ECMAScript. Equality operators. Another way to check for NaN is [js]NaN != NaN[/js]
-  micha-s on Note 1. ECMAScript. Bound functions. I am completely misunderstanding the "Constructor with various number of arguments" section.
-  Dmitry Soshnikov on ES6

javascript the good parts - X

https://www.youtube.com/results?search\_query=javascript+the+good+parts

YouTube CH

javascript the good parts

Upload Sign In

Filters ▾

About 27,000 results

**JavaScript: The Good Parts**  
by GoogleTechTalks • 5 years ago • 409,631 views  
Google Tech Talks Web Exponents presented by Doug Crockford February 27, 2009  
blog post: ...  
OFFICIAL CC 1:03:48

**Douglas Crockford: JavaScript: The Good Parts**  
by YUI Library • 3 years ago • 13,180 views  
In this talk from 2007, Douglas Crockford takes us on a journey through the lens of his own personal experience with JavaScript ...  
39:38

**Flexible Navigation | JavaScript Good Parts | CSS Animation Tricks | The Treehouse Show Episode 75**  
by Treehouse • 7 months ago • 9,750 views  
In this episode of The Treehouse Show, Nick Pettit (@nickrp) and Jason Seifer (@jseifer) talk about the latest in web design, web ...  
14:41 HD

**Houston Developers Book Club: JavaScript the Good Parts Ch 4**  
by Jonathan Birkholz • 8 months ago  
This meeting will be about Chapter 4: Functions from JavaScript the Good Parts. For this time: - Less confusion with Google ...  
1:10:21 HD

**Test Your Knowledge of Function Scope with Douglas Crockford**  
by MJG International (Frontend Masters) • 1 year ago • 9,342 views  
In this lesson Douglas Crockford tests your knowledge of JavaScript function scope.  
10:31 HD