JavaScript

WS Slides #2

Content

- Specification
- JavaScript
- Module Pattern
- Pseudo-classical style
- Javascript APIs

JavaScript Specification

JavaScript is an implementation of the ECMAScript Language Specification

- We use ECMA 262 ed. 5.1 (ed 6 upcomming … but when…?)
- Hard reading …
- Original author Brendan Eich

JavaScript Characteristics

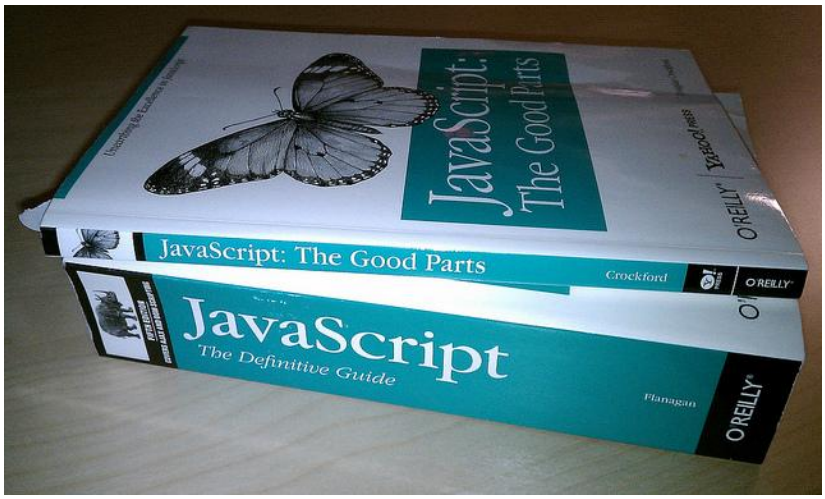
- Interpreted scripting language
- Run in host environment (browser or other)
- C-family syntax
- Non-statically typed
- References
- Lots of implicit type conversions (coercion)
- Prototype based (object based, no classes)
- First class functions
- Closures
- Single threaded
- Garbage collected

JavaScript is a prototype-based scripting language with dynamic typing and first-class functions. This mix of features makes it a multi-paradigm language, supporting object-oriented, imperative, and functional programming styles.

Despite some naming, syntactic, and standard library similarities, JavaScript and Java are otherwise unrelated and have very different semantics.

- Java and JavaScript are as similar as Car and Carpet or …
- …Ham and hamster …

JavaScript Criticism



There is severe criticism of JavaScript, … too many design flaws

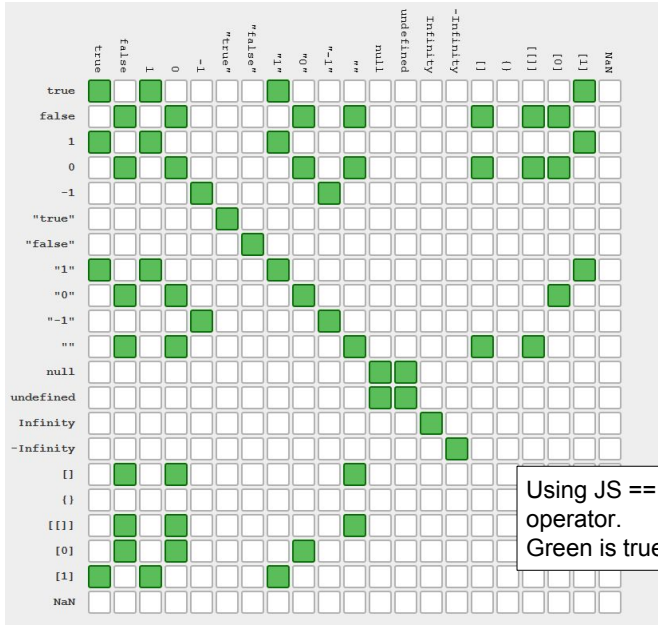
- One global namespace! No modules, no block scope, … and more… The real bad parts of JS
- Some thing will hopefully change with ECMA version 6, but when … ?

Some hard points to grasp

- Types and coercion?
- Scope and hoisting work?
- What is "this"?
- How does prototypal inheritance work?
- … NOT an easy language!

JavaScript gotchas
JavaScript WAT (starting at 1:23)

Types and Coercion ...



Truth table for == operator

- Coercion (type conversion) occurs with primitive types and many operator; +, ==, !=, >, =<, …

"JavaScript has two sets of equality operators: === and !==, and their evil twins == and != . The good ones work the
way you would expect. If the two operands are of the same type and have the same value, then === produces true
and !== produces false. The evil twins do the right thing when the operands are of the same type,
but if they are of different types, they attempt to coerce the values.
The rules by which they do that are complicated and unmemorable. The lack of transitivity is alarming.
My advice is to never use the evil twins. Instead, always use === and !==
(all of the comparisons in slide produce false with the === operator)."
// Douglas Crockford, JS Guru (bit modified)

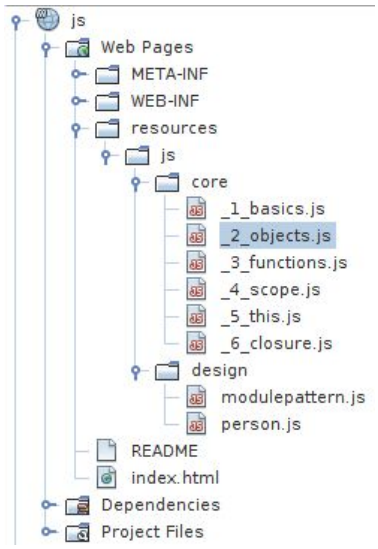
Disclaimer

There are many ways to use and write JavaScript

- No standard common agreed upon way or style
  - Many opinions, what to use and how
- Following samples is one way to do it (a rather sensible way I think)

JavaScript style guides

Crash Course JavaScript

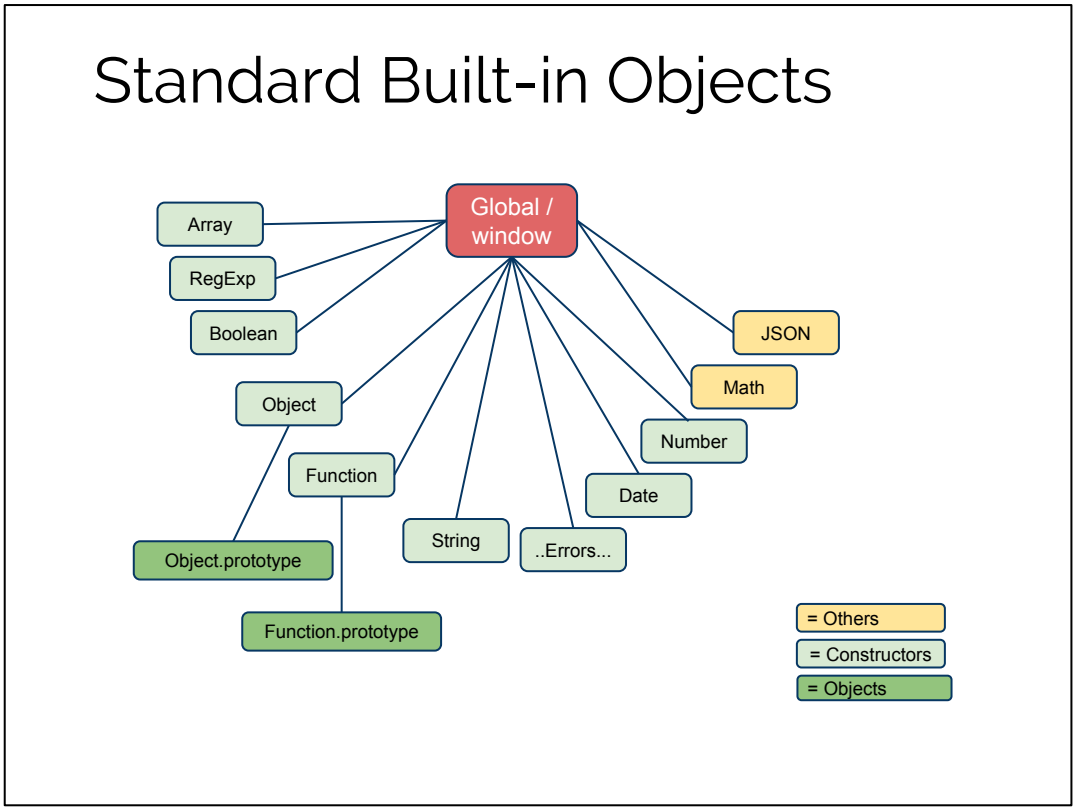


Now we'll run through a lot of code

- See sample project js (pictured in slide)
- Using Chrome Developer Tools as debugger
- In between we'll look at some picture (the following slides)

Some links if you would like to dig deeper.

- Speaking Java (JavaScript tutorial)
- JavaScript, Mozilla Developer Network
- More on coercion (GitHub)
- ECMA-262-3 in detail (hardcore JS by Dmitry Soshnikov)



The Standard defines several built-in objects (supplied by runtime environment)

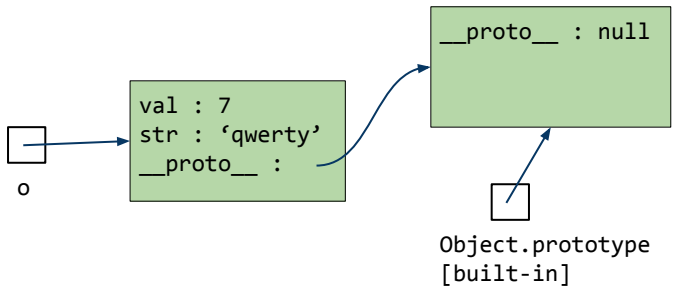
- NOT same as DOM API objects but …
- … the standard Global object in browsers renamed (replaced?) with the window object
- The constructor properties of the global object: Array, Object, Function , etc. (more to come)
- The *.prototype property of the constructor properties (Object.prototype, etc.)
- Other properties: JSON and Math (seems similar to static Java classes)

Docs

- String
- Math

Prototype Chain

```
var o = {
   val: 7,
   str: 'qwerty'
};
```



All objects have an (internal) reference to a “parent” object the “__proto__” (chain ending in the top level
built-in object Object.prototype with no parent)

- __proto__ not part of JS standard interface (not yet, i.e. ECMA 5.1). Don’t use directly in code (use Object.getPrototypeOf(...))
- Parent for object literals will be Object.prototype
- Property lookup will follow __proto__ chain, if property not found in actual object check __proto__ etc. (prototypical inheritance)
- If property not found in chain, undefined returned
- hasOwnProperty()
- … more to come …

Object Instantiation

Object initializers

```
var person = { name : “Otto” };
```

Object.create

```
var person = Object.create( { name : “Otto” })
```

Constructor function

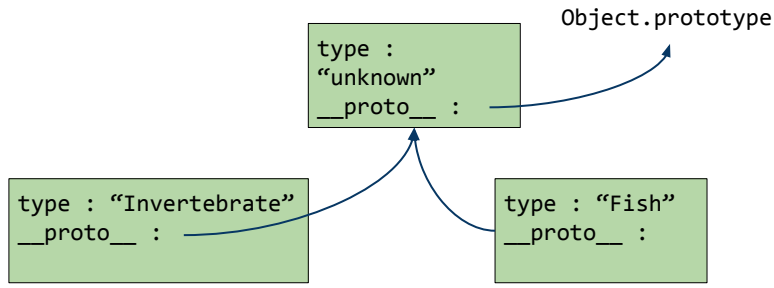
```
var person = new Person(“Otto”);
```

There are at least three ways to create an object

- Object initializers already seen
- Working with objects
- Using new looks like Java but it's not ...

Object.create

```
var animal = { type : “unknown” };
var lobster = Object.create(animal);
lobster.type = “Invertebrate”;
var fish = Object.create(animal);
fish.type = “Fish”;
```

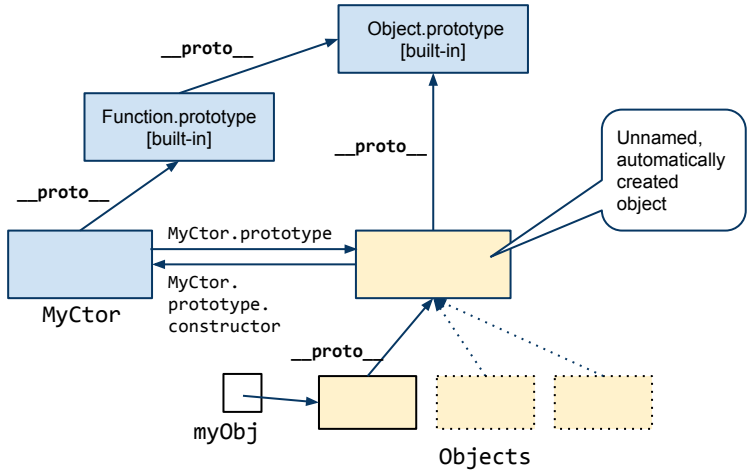


Object.create

- Will create object and set __proto__ to parameter object.

Constructor Functions

```
function MyCtor() { … }
var myObj = new MyCtor();
```

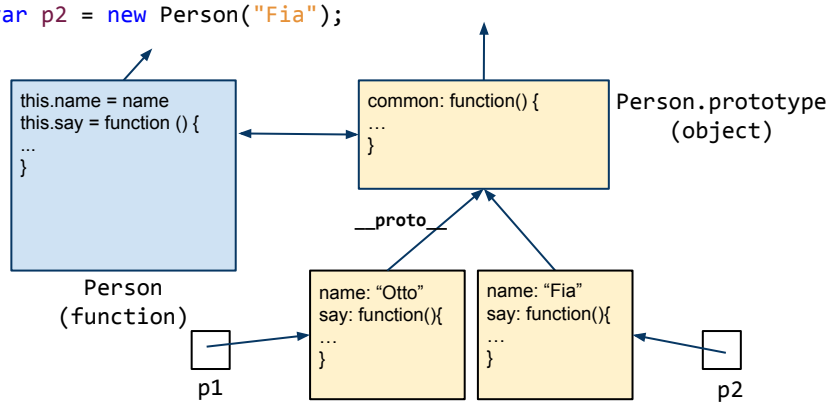


All function (function are objects) have a prototype property (NOT same as __proto__, the object prototype)

- The property is in standard API, ok to use
- Prototype property points to automatically created nameless object …
- … used as __proto__ for all objects created by the constructor function in combination with new operator
- Properties assigned to MyCtor.prototype shared by all objects (similar to Java inheritance)
- Nothing special with constructor function so as an idiom we use leading uppercase to distinguish.
- If forgetting new, variable will be undefined

Constructor Function Sample

```
function Person(name) {
    this.name = name || "unknown"; // this is actual object
    this.say = function () { … };
}
Person.prototype.common = function (){ … };
var p1 = new Person("Otto");
var p2 = new Person("Fia");
```



See code sample _2_objects.js

Module Pattern

```
var myModule = (function() {
    // Private parts
    var myPrivateVar = 0;
    myPrivateMethod = function(foo) { ... };

    // Public API
    return {
        myPublicVar: "foo",
        myPublicFunction: function(bar) {
            myPrivateVar++;
            myPrivateMethod(bar);
        }
    };
})();  // <---- !
```

The Module

```
myModule.myPublicVar;
myModule.myPublicFunction("...");
```

Module pattern

There are some variations

- here is one
- JS design patterns

Pseudo-Classical Style

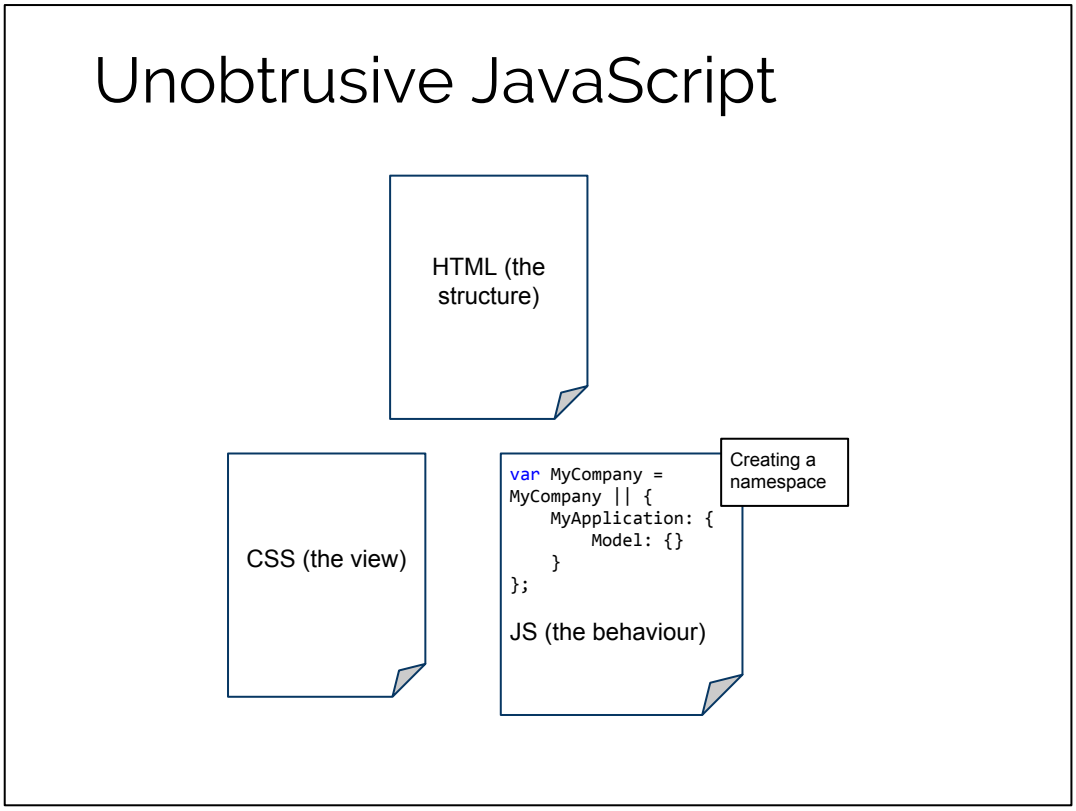
```
var Person = function(name, age, sex){
    this.name = name || “unknown”;
    this.age = age || -1;
    this.sex = sex || “unknown”;
};

Person.prototype = [ module pattern ]


var p = new Person("Otto", 13, "Male");
```

Pseudo-classical style

- To emulate class based OO languages like Java
- PC style = Constructor + Constructor.prototype + Module pattern
- If inheritance set: sub.prototype.__proto_ = base.prototype
- … override, polymorphism and more.
- Details of the object model (using Object.create)

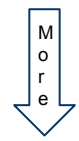


Unobtrusive JavaScript

- Separations of concerns
- No CSS, JS in HTML
- Namespaces

JavaScript APIs

2015-02-24	Pointer Events	Recommendation *Nightly Draft*
2015-02-10	Vibration API	Recommendation *Nightly Draft*
2015-02-03	Server-Sent Events	Recommendation *Nightly Draft*
2015-01-08	Indexed Database API	Recommendation *Nightly Draft*
2014-03-13	Metadata API for Media Resources 1.0	Recommendation
2014-02-11	Progress Events	Recommendation *Nightly Draft*
2014-01-16	JSON-LD 1.0 Processing Algorithms and API	Recommendation
2013-12-12	Performance Timeline	Recommendation
2013-12-12	User Timing	Recommendation
2013-10-31	Widget Interface	Recommendation *Nightly Draft*
2013-10-29	Page Visibility (Second Edition)	Recommendation
2013-10-24	Geolocation API Specification	Recommendation *Nightly Draft*
2013-10-10	Touch Events	Recommendation *Nightly Draft*
2013-02-21	Selectors API Level 1	Recommendation
2012-12-17	Navigation Timing	Recommendation
2012-12-17	High Resolution Time	Recommendation
2008-12-22	Element Traversal Specification	Recommendation
2015-08-06	Runtime and Security Model for Web Applications	Group Note
2015-07-23	The app: URL Scheme	Group Note *Nightly Draft*
2015-07-23	TCP and UDP Socket API	Group Note *Nightly Draft*
2015-07-23	Task Scheduler API Specification	Group Note *Nightly Draft*
2015-07-14	Permissions for Device API Access	Group Note



There are many

- DOM API in HTML5 specification
- … others