Web Applications 2017

Week 1, Slides 2

Content

- ECMAScript 5 (JavaScript)
- Module Pattern
- Pseudo classical style
- Document Object Model
- DOM API
- DOM Events
- JSON
- Promises
- jQuery

ECMAScript 5

ECMAScript Language Specification 5.1 (ES5) is a standard describing JavaScript (and other script languages)

- JavaScript the most popular
- Later we'll have a look at upcoming ES6 (AKA ES2015).

Readings:

- ECMA Script Wikipedia
- JSFiddle (testing code JS code)

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)
- **Objects**
- No classes, prototype based
- First class **functions**
- Closures
- **Single threaded**
- Garbage collected

 =  !?

Java and JavaScript are as similar as Car and Carpet or …

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

Readings

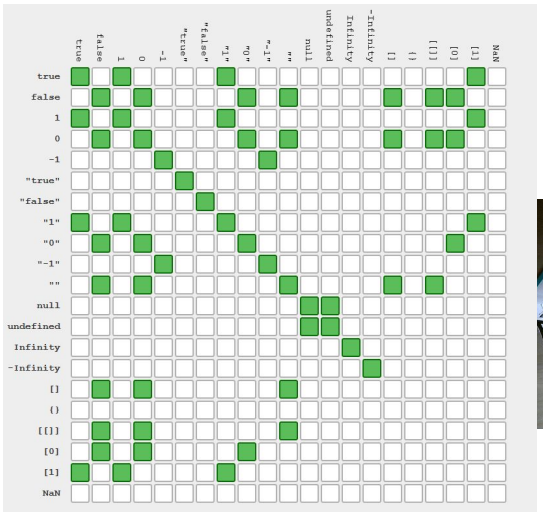
- Language reference at MDN
- JavaScript guide at MDN
- Tutorial at w3schools
- JavaScript style guides

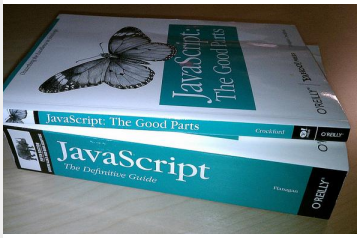
More Readings

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

Design Flaws

Truth Table for == operator (green is true)

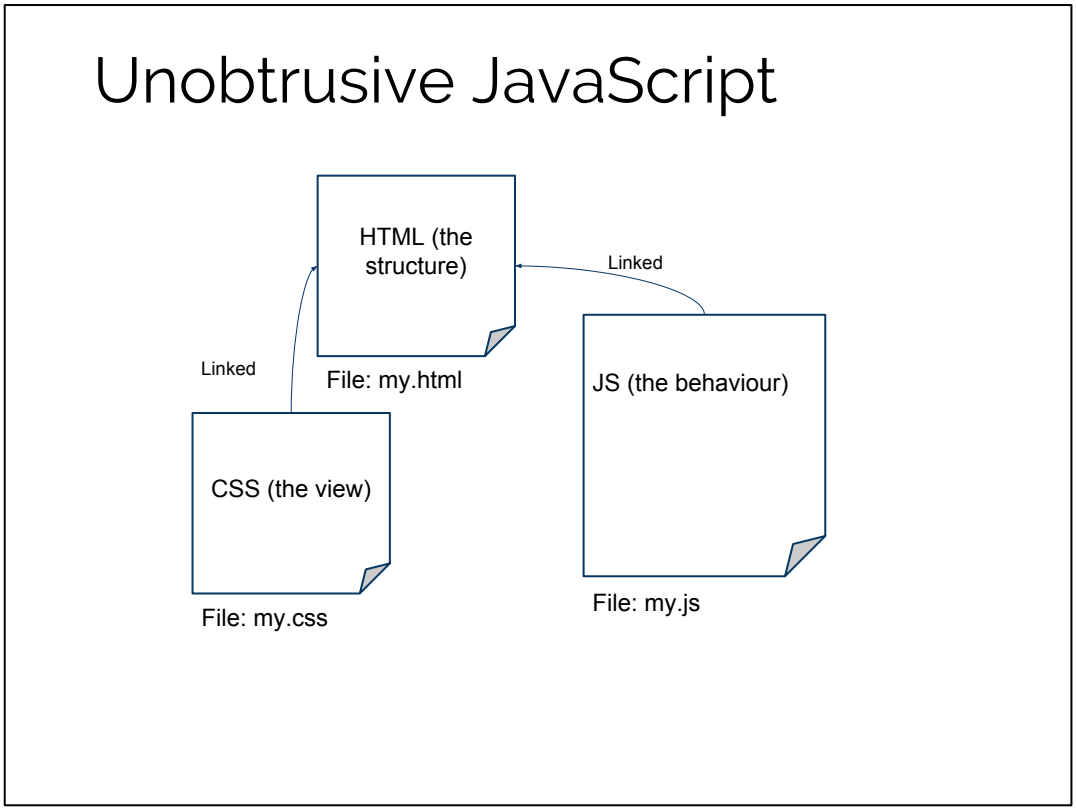




JS had got severe criticism for design flaws.
Criticism: Truth table for == operator

Readings:

- The real bad parts of JS
- JavaScript gotchas
- JavaScript WAT (starting at 1:23)
- Some alternative languages (compiled to JavaScript, we don't use only for knowledge)
  - CoffeScript
  - TypeScript
  - Dart



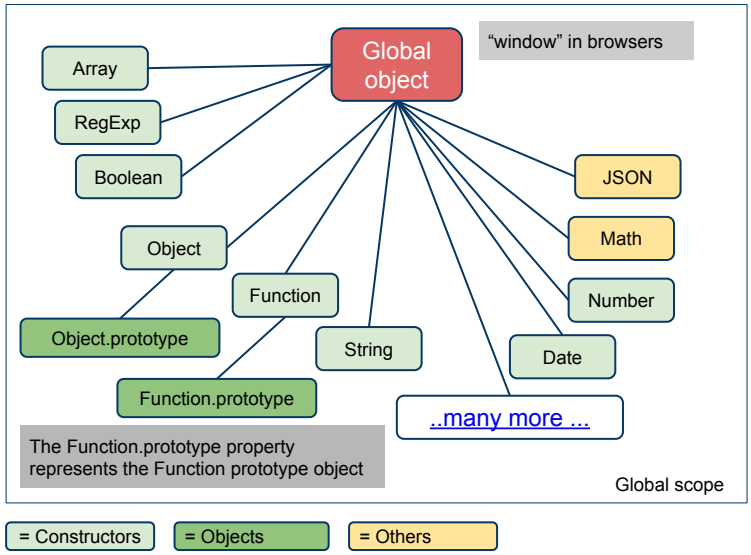
Separations of concerns, no CSS, JS in HTML

- I.e. no CSS or JS in HTML

Readings:

- Unobtrusive JavaScript

Standard Built-in Objects



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

Object Instantiation

Object initializers (object literal)

var = Variable declaration

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

Object.create

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

Constructor function

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

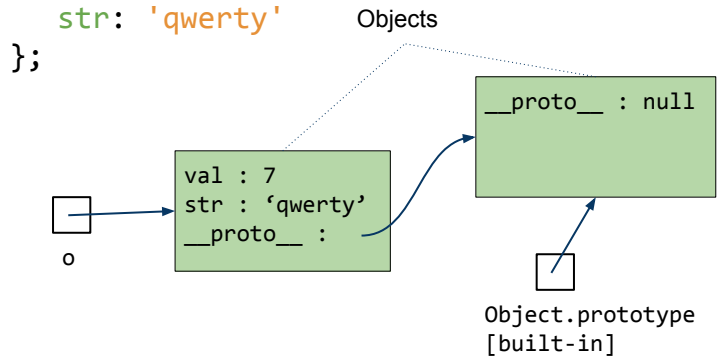
This is NOT like Java!

Readings

- Working with objects

Prototype Chain

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

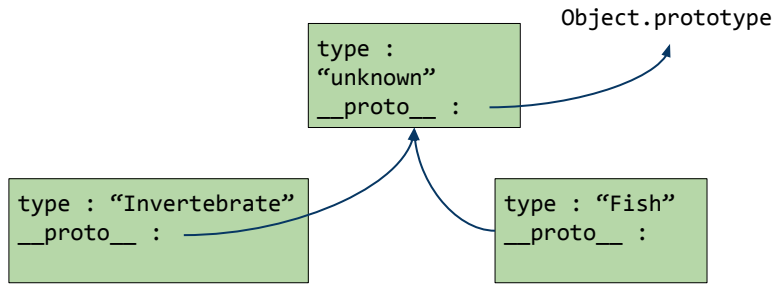


All objects have an (internal) reference to a “parent” object the “__proto__” (chain ending in the top level
built-in object Object.prototype with no parent)

- __proto__ not part of JS standard interface (not yet, i.e. ECMA 5.1).
- 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.create

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

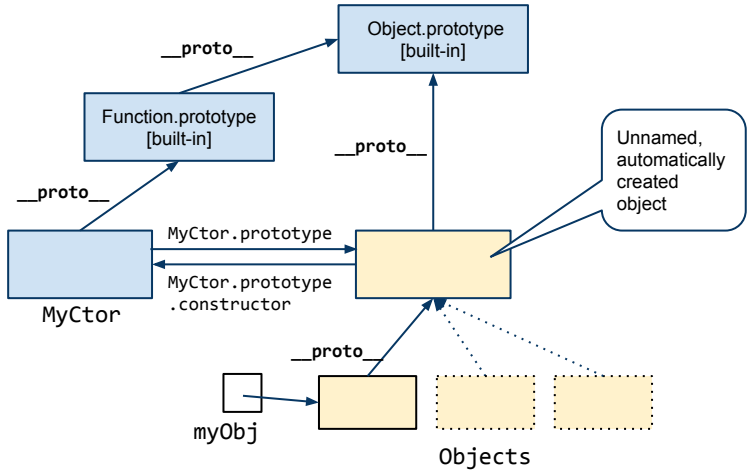


Object.create

- Will create object and set __proto__ to parameter object.

Constructor Functions

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

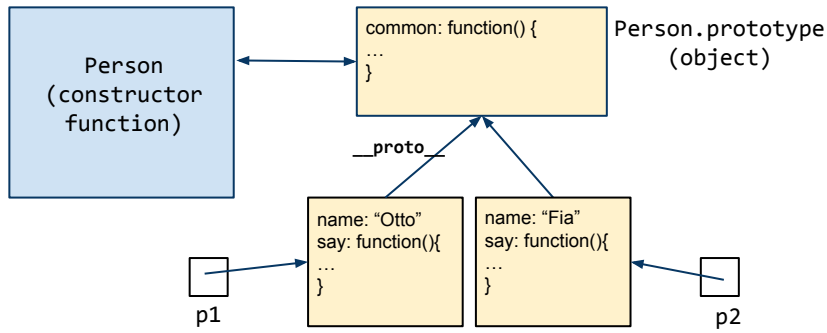


All function (function are objects) have a prototype property (NOT same as __proto__, the object prototype)

- 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, as an idiom we use leading uppercase to distinguish.
- If forgetting new, variable will be undefined!

Constructor Function Sample

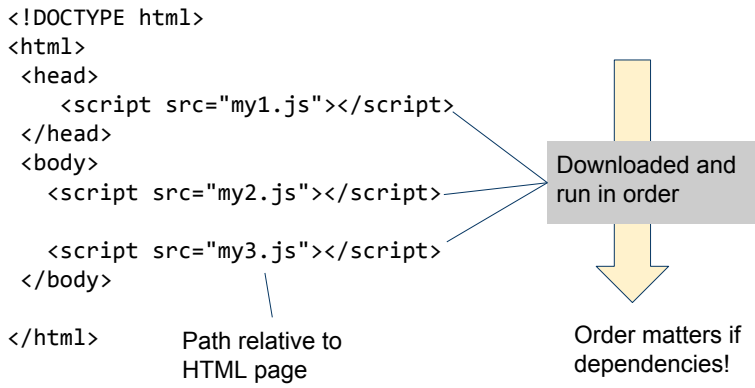
```
function Person(name) {
    this.name = name || "unknown"; // this is actual object
    this.say = function () { … };
}
Person.prototype.common = function (){ … }; // 'Inherited'
var p1 = new Person("Otto");
var p2 = new Person("Fia");
```



See code sample _2_objects.js

JS in Browser

```
<!DOCTYPE html>
<html>
 <head>
    <script src="my1.js"></script>
 </head>
 <body>
   <script src="my2.js"></script>

   <script src="my3.js"></script>
 </body>

</html>
```



JS Details Crash Course

We'll browse through JS code covering the following topics

- Basics
- Functions
- Scope
- this
- Closures

Module Pattern

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

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

The Module

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

ES5 lacks any module system (ES6 has). Module pattern is a way to create modules in pure ES5

There are some variations

- here is one
- JS design patterns

Pseudo-Classical Style

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

Person.prototype = [ module pattern here ]


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

Readings:

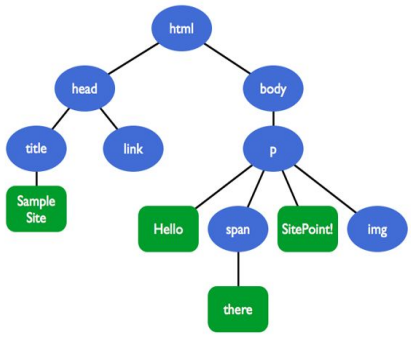
- … override, polymorphism and more.
- Details of the object model

Document Object Model (DOM)

HTML Document

```
<!DOCTYPE html>
<html>
    <head>
        <title>Sample Site</title>
        <link href="mystyle.css">
    </head>
    <body>
        <p>
            Hello
            <span>there</span>
            SitePoint!
        </p>
        <img src="smiley.gif"/>
    </body>
</html>
```

HTML DOM



Accessible with JS!

Readings:

- DOM Wikipedia
- Intro to DOM
- Node tree (specified in W3C DOM 4 recommendation, not in HTML5)
- Test with Live DOM viewer

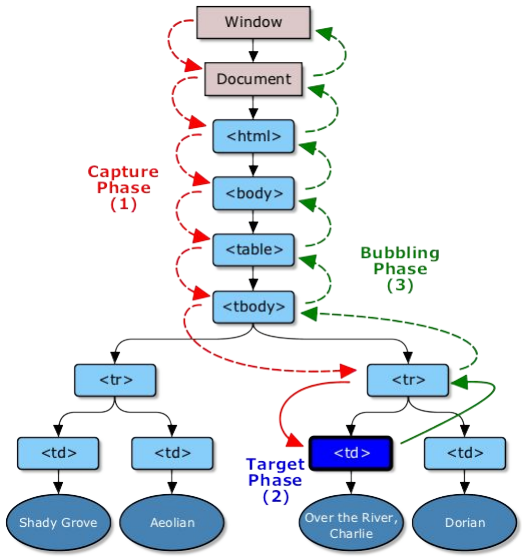
DOM API

```
// Create a new div element and give it some content
var newDiv = document.createElement("div");
var newContent = document.createTextNode("Hi there and
greetings!");


newDiv.appendChild(newContent); //Add to the newly created div.

// Add the newly created element and its content into the DOM
var currentDiv = document.getElementById("div1");
document.body.insertBefore(newDiv, currentDiv);
```

Readings

- MDN DOM Reference

DOM Events



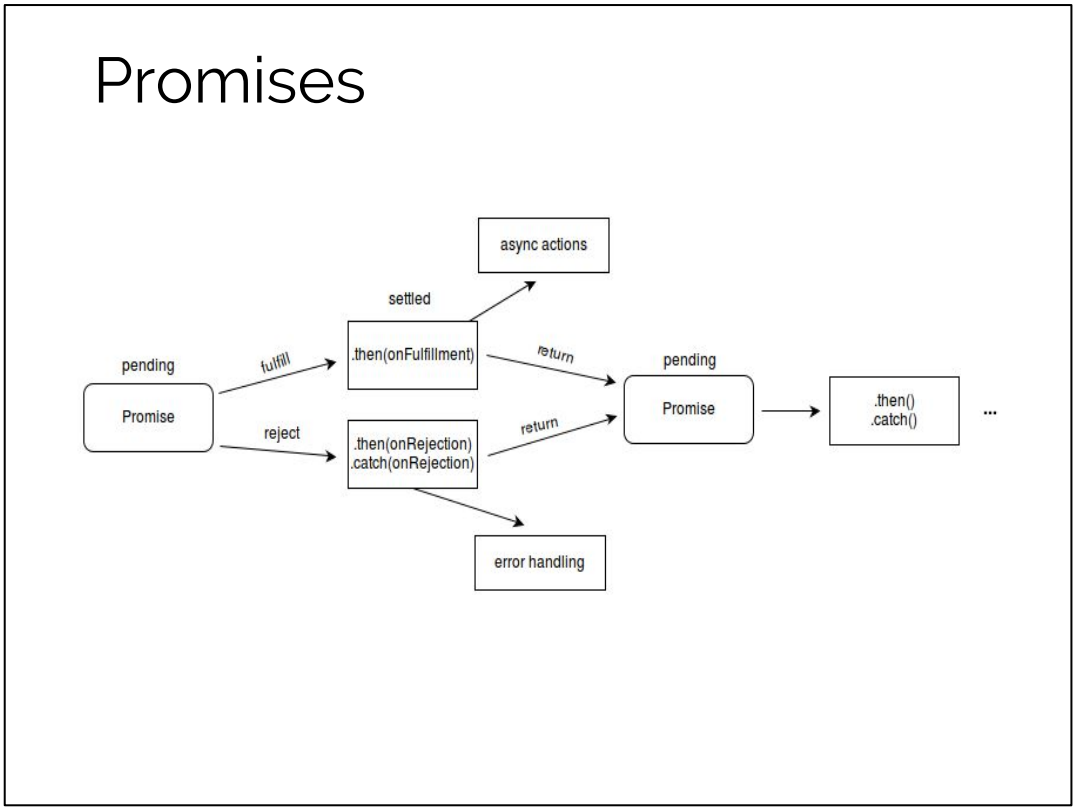
Chrome developer tools

JavaScript Object Notation (JSON)

```
{
     "firstName": "John",
     "lastName": "Smith",
     "age": 25,
     "address": {
         "streetAddress": "21 2nd Street",
         "city": "New York",
         "state": "NY",
         "postalCode": "10021"
     },
     "phoneNumber": [
         { "type": "home", "number": "212 555-1234" },
         { "type": "fax", "number": "646 555-4567" }
     ],
     "newSubscription": false,
     "companyName": null
 }
```

Readings

- JSON
- JSON org



The Promise object is used for asynchronous computations. A Promise represents a value which may be available now, or in the future, or never.

Readings

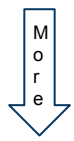
- Promises MDN
- Intro to Promises Google
-

Chaining Promises

```
var FlightDashboard = function( $scope, user, flightService, weatherService ){
        travelService
            .getDeparture( user )             // Request #1
            .then( function( departure ){
                $scope.departure = departure; // Response Handler #1
                return travelService.getFlight( departure.flightID );// Request #2
            })
            .then( function( flight ){
                $scope.flight = flight;    // Response Handler #2
                // Request #3
                return weatherService.getForecast( $scope.departure.date );
            })
            .then( function( weather ){
                $scope.weather = weather;  // Response Handler #3
            });
        };
```

More JavaScript APIs

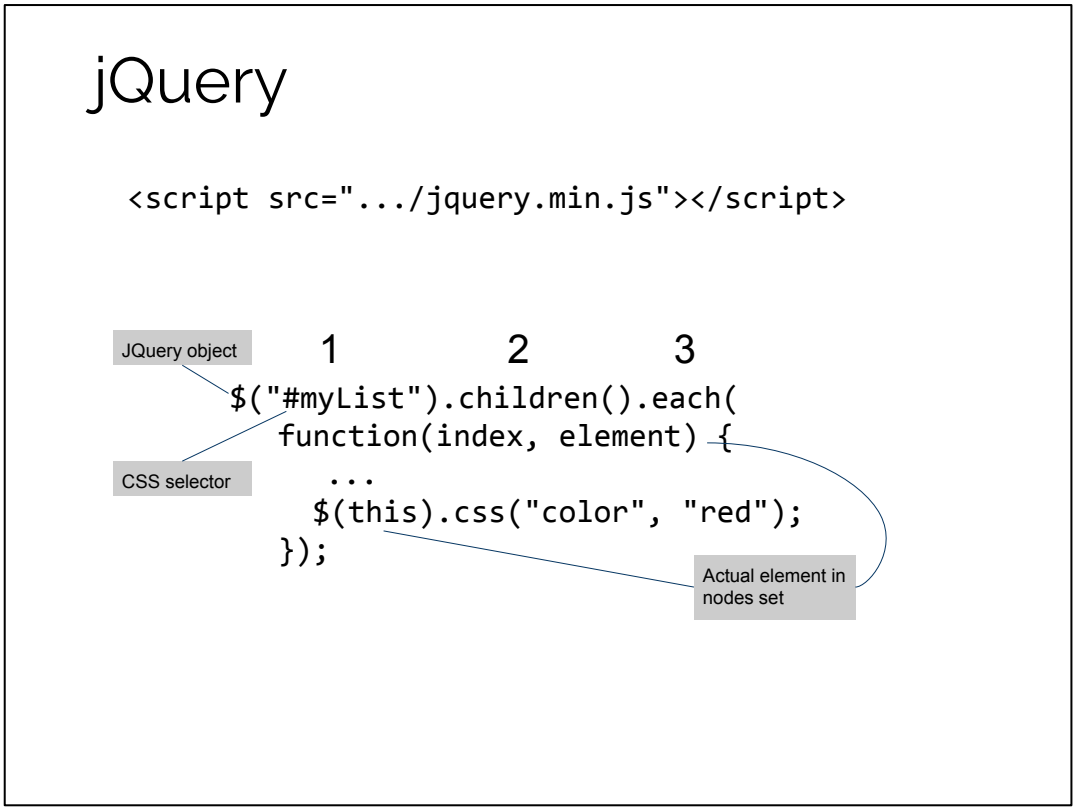
2015-02-24	Pointer Events	Recommendation *Nightly Draft*
2015-02-10	Vibration API	Recommendation *Nightly Draft*
2015-02-03	Server-Sent Events	Recommendation *Nightly Draft*
2015-01-08	Indexed Database API	Recommendation *Nightly Draft*
2014-03-13	Metadata API for Media Resources 1.0	Recommendation
2014-02-11	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



DOM API in HTML5 specification

Readings:

- JS APIs current status



Higher level JS library. Simplified DOM API etc.

jQuery philosophy

1. Find some DOM nodes using CSS selectors. Nodes will be “wrapped” in the jQuery object (objects enhanced with jQuery methods)
2. Do something with them using jQuery methods (instead of native JS API i.e. DOM). Chain multiple method calls to process the set of nodes
3. Don't need to explicitly traverse the set. Use implicit iteration (each()-method) to get final result. Possible to get "previous" set back

Reading

- jQuery

jQuery Elements

Defer execution (until DOM loaded)

```
$(function () { … });
```

Find

```
var ul = $("#myList");
var children = $("#myList").children();
var parent = $("#myList").parent();
```

Add

```
$("<input id='btnClose' type='button'
                        value='Save'/>").appendTo('#popUp');
```

Modify

```
$('div.second').replaceWith('<h2>New heading</h2>');
```

Delete

```
$('.productTableBody tr').remove();
```

Find and manipulate elements

- To defer execution until DOM is ready wrap code in $ (code run when DOM ready, use to set up listeners etc.)
- Use CSS selectors to find element(s) then jQuery methods to change, add, remove, …

jQuery Attributes and Style

Attribute

```
$('#greatphoto').attr('alt');  // Get
$('#greatphoto').attr('alt', 'Beijing Brush
Seller');     // Set
```

Style

```
$('#mydiv').css('color') // Get
$('#mydiv').css('color', 'green')  // Set
```

Get some user input

```
var value = $(“#myInput”).val();
```

Find and manipulate attributes, styles and input.

jQuery Events

Existing node

```
$(function(){
   $("#btn1").on("click", listener);
});
```

Non-existing node

```
$(function(){
  $(document).on("dblclick","#myBtn", listener);
 });
```

Event names similar to native Event API

- Skip leading on
- Setup listeners when DOM finished using $(function() { … })
- NOTE: No parentheses for listener (it's the function reference we want, not executing it)