JS Best Practices

Created by Johannes Hoppe

My Goal

Higher code qualitiy. Pragmatic solutions.
No fancy stuff.

Know Your MS Tools

Visual Studio 2010/2012
JScript Editor Extensions
Resharper 7.1
JSHint
Chutzpah
Firebug / F12

Know the pitfalls

Implied globals

Forgetting var

```
var foo = function() {
    bar = 1;
};
```

Boolean type conversion

To Truthy or to Falsy. That is the only question!

```
var el = document.getElementById('does_not_exist');

if(el == false) {
    alert("shouldn't we see this message?!");
}
```

Trailing comma

works on my machine!

```
var foo = {
    bar: "bar",
    baz: "baz",
};
```

Return undefined

señor developers wear mustaches

```
var foo = function() {
    return
    {
        x : "looks like C# now!"
    };
}
```

Associative arrays

they don't exist

```
var x = [];
x['foo'] = "bar";
```

try .. catch .. finally

who cares about the reason?

```
var foo = function() {
    try {
       doCrazyStuff;
    } catch (e) {
        return false;
    }
    return true;
};
```

for .. in

use a framework

```
var data = {
    foo: "oh",
    bar: "my"
}

for (var d in data) {
    console.log(data[d]);
}
```

for .. in

& never touch Object.prototype

```
 Object.prototype.yetAnotherToString = function() {
    return "god";
}
```

Hoisting

declare upfront all variables

```
var foo = "global";

var bar = function() {
   alert(foo);
   var foo = "local";
   alert(foo);
};
```

Eval

... and the job is done

```
function poorMansJsonParser(text) {
    return eval("(" + text + ")");
}

var text = ' { "hello" : "world" } ';
var json = poorMansJsonParser(text);
```

Eval is evil!

Never ever!

```
var text = ' function() { alert("hacked!"); })( ';
```

Globals

the mother of all antipatterns

```
function foo() {
    return "bar";
}

console.log(this['foo']());
```

Every time you clutter the global namespace,
somewhere in the world a helpless kitten dies!

Pretty Code

Coding conventions

1. intentation (4 spaces!)
2. curly braces everywhere
3. semicolons everywhere
4. constructor functions: UpperCamelCase
5. all other: lowerCamelCase

Globals

reduce, minimize, delete or kill them

```
(function() { "wtf?" })();
```

The switch-case syndrome

a functional language wants functions!

```
switch (something) {

    case 1:
        doFirst();
        break;

    case 2:
        doSecond();
        break;

    case 3:
        doThird();
        break;
}
```

Lookup tables

avoid the switch-case syndrome

```
var methods = {
    1: doFirst,
    2: doSecond,
    3: doThird
};
if (methods[something]) {
    methods[something]();
}
```

Inheritance

favour composition over inheritance (FCoI)

> “ Because inheritance exposes a subclass to details of its parent's implementation, it's often said that 'inheritance breaks encapsulation'. ”

(Gang of Four 1995:19)

Revealing Module Pattern

```
var myRevealingModule = function () {

    var _name = "Johannes";

    function greetings() {
        console.log("Hello " + _name);
    }

    function setName(name) {
        _name = name;
    }

    return {
        setName: setName,
        greetings: greetings
    };
}();
```

» Documentation

Modul loaders

use AMD (require.js)

```
define('test', ['jquery'], function() {

    return {
        saySomething : function() { alert("hello!"); }
    }
});

require(['test'], function(t) {
    t.saySomething();
});
```

Events

Publish/Subscribe Pattern

```
var $events = $({});

$events.bind('somethingHappens', function() {
    alert("Something happened!");
});

$events.trigger('somethingHappens');
```

TDD with Jasmine

Why Jasmine?

BDD-style similar to JSpec or RSpec,
created by authors of jsUnit and Screw.Unit

independent from any browser, DOM,
framework or host language

integrates into continuous build systems

Jasmine Bootstrap

```
<!DOCTYPE html>
<html>
<head>
    <title>Jasmine Spec Runner</title>

    <link rel="stylesheet" href="lib/jasmine-1.3.1/jasmine.css" />
    <script src="lib/jasmine-1.3.1/jasmine.js"></script>
    <script src="lib/jasmine-1.3.1/jasmine-html.js"></script>

    <!-- include source files here... -->
    <script src="src/Player.js"></script>
    <script src="src/Song.js"></script>

    <!-- include spec files here... -->
    <script src="spec/SpecHelper.js"></script>
    <script src="spec/PlayerSpec.js"></script>

    <script>

        (function () {

            var htmlReporter = new jasmine.HtmlReporter();
            var jasmineEnv = jasmine.getEnv();

            jasmineEnv.addReporter(htmlReporter);
            jasmineEnv.specFilter = function (spec) {
                return htmlReporter.specFilter(spec);
            };

            var currentWindowOnload = window.onload;

            window.onload = function () {
```

Output

```
Jasmine 1.3.1 revision 1354556913                    finished in 0.018s

• • • • •

Passing 5 specs                                      No try/catch

Player
  should be able to play a Song

  when song has been paused
    should indicate that the song is currently paused
    should be possible to resume
  tells the current song if the user has made it a favorite

  #resume
    should throw an exception if song is already playing
```

Hello World

```
var helloWorld = function() {
    return "Hello World!";
};

describe('helloWorld', function() {
    it('says hello', function() {

        expect(helloWorld()).toEqual("Hello World!");
    });
});

jasmine.getEnv().execute();
```

hint: press F12 and paste this code!

Matchers

```
expect(x).toEqual(y);
expect(x).toBe(y);
expect(x).toMatch(pattern);
expect(x).toBeDefined();
expect(x).toBeUndefined();
expect(x).toBeNull();
expect(x).toBeTruthy();
expect(x).toBeFalsy();
expect(x).toContain(y);
expect(x).toBeLessThan(y);
expect(x).toBeGreaterThan(y);
expect(function(){fn();}).toThrow(e);
```

Own matchers

```
beforeEach(function () {
    this.addMatchers({
        isACat: function () {
            return this.actual.isFluffy() && this.actual.isLazy();
        }
    });
});

describe('Garfield', function () {
    it('is a cat', function () {
        expect(new Garfield()).isACat();
    });
});
```

» Documentation

Test-Driven Development

1. Write your tests
2. Watch them fail
3. Make them pass
4. Refactor
5. Repeat

see Growing Object-Oriented Software, Guided by Tests, page 6
see Working Effectively with Legacy Code, page 62 or many other

1. Write your test

```
describe("saveFormat", function () {

    var original = '{0} - {1} - {2}';

    it("should replace placeholders", function () {
        var expected = 'A - B - C';
        var formated = saveFormat(original, 'A', 'B', 'C');
        expect(formated).toEqual(expected);
    });

    it("should encode injected content", function () {
        var expected = 'A - &lt;b&gt;TEST&lt;/b&gt; - C';
        var formated = saveFormat(original, 'A', '<b>TEST</b>', 'C');
        expect(formated).toEqual(expected);
    });
});
```

2. Watch them fail

```
var saveFormat = function() {
    return "boo!";
};
jasmine.getEnv().execute();
```

Demo

3. Make them pass

```
var saveFormat = function(txt) {

    $(arguments).each(function (i, item) {
        if (i > 0) {
            item = ($('<div/>').text(item).html());
            txt = txt.replace("{" + (i - 1) + "}", item);
        }
    });
    return txt;
};
jasmine.getEnv().execute();
```

Demo

4. Refactor

```
function htmlEncode(input) {
    return ($('<div/>').text(input).html());
}

var saveFormat = function(txt) {

    $.each(arguments, function (i, item) {
        if (i > 0) {
            item = htmlEncode(item);
            txt = txt.replace("{" + (i - 1) + "}", item);
        }
    });
    return txt;
};
jasmine.getEnv().execute();
```

Demo

5. Repeat

```
function htmlEncode(input) {
    return ($('<div/>').text(input).html());
}

var saveFormat = function() {

    var args = Array.prototype.slice.call(arguments);
    var txt = args.shift();

    $.each(args, function (i, item) {
        item = htmlEncode(item);
        txt = txt.replace("{" + i + "}", item);
    });
    return txt;
};
jasmine.getEnv().execute();
```

Demo

Testing HTML

Jasmine is DOM agnostic
comes without tools to set up HTML fixtures

Definition:
A test fixture is a fixed state of a set of
objects used as a baseline for running tests.

First Solution

in memory fixture with jQuery

```
describe('trivial jQuery plugin', function () {

    var fixture;
    beforeEach(function () {
        fixture = $('<div>some HTML code here</div>');
    });

    it('should do something', function () {
        fixture.myPlugin();
        expect(fixture).toHaveClass("newClass");
    });
});
jasmine.getEnv().execute();
```

... only works for trivial plugins!

Clumsy Solution

directly append to/remove from DOM

```
describe('my jQuery plugin', function () {

    beforeEach(function () {
        $('#fixture').remove();
        $('body').append('<div id="fixture">HTML</div>');
    });

    it('should do something', function () {
        $('#fixture').myPlugin();
        expect($('#fixture')).toHaveClass("newClass");
    });
});
jasmine.getEnv().execute();
```

jasmine-jquery

custom matchers, HTML/style/JSON fixtures, event spies

```
describe('my jQuery plugin', function () {

    beforeEach(function() {
        jasmine.getFixtures().fixturesPath='js/5_jasmine-demo_jquery';
        jasmine.getFixtures().load('jquery.myPlugin.spec.html');
    });

    it('should do something', function() {

        var $div = $('#helloWorld').myPlugin();
        expect($div).toHaveClass("newClass");
    });
});
jasmine.getEnv().execute();
```

Demo

TDD → BDD

Spies

test behaviour

```
describe('Garfield', function () {
    describe('when told to be nice', function() {

        var garfield;
        beforeEach(function() {
            garfield = new Garfield();
            spyOn(garfield, 'goToFridgeAndEatPizza').andCallThrough();
        });

        it('should answer with ok', function() {
            var answer = garfield.beNice();
            expect(answer).toEqual("ok");
        });

        it('should steal pizza', function () {
            garfield.beNice();
            expect(garfield.goToFridgeAndEatPizza).toHaveBeenCalled();
        });
    });
});
jasmine.getEnv().execute();
```

A spy can stub any function and tracks calls to it and all arguments.

Demo

Danke!