JavaScript coding tips

Axel Rauschmayer
rauschma.de

Sud Web 2014

Coding styles on GitHub

Source: sideeffect.kr/popularconvention/#javascript

Comma: first or last?

7% comma first

```
var foo = 1
  , bar = 2
  , baz = 3;
```

```
var obj = {
      foo: 1
    , bar: 2
    , baz: 3
};
```

93% comma last

```
var foo = 1,
    bar = 2,
    baz = 3;
```

```
var obj = {
    foo: 1,
    bar: 2,
    baz: 3
};
```

Indentation: spaces vs. tabs

81% spaces

```
function foo() {
␣␣return 'bar';
}
```

19% tabs

```
function foo() {
→   return 'bar';
}
```

Space after function name?

33% space after function name

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

67% no space after function name

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

Spaces inside parens?

6% spaces inside parens

```
function fn( x, y ) {
  ...
}

if ( true ) {
  ...
}
```

94% no spaces inside parens

```
function fn(x, y) {
  ...
}

if (true) {
  ...
}
```

Object literals

22% no space after colon

```
{
  foo:1,
  bar:2,
  baz:3
}
```

64% space after colon

```
{
  foo: 1,
  bar: 2,
  baz: 3
}
```

14% space before & after

```
{
  foo : 1,
  bar : 2,
  baz : 3
}
```

Conditional statements

79% space after keyword

```
if (true) {
   ...
}
while (true) {
   ...
}
switch (v) {
   ...
}
```

21% no space after keyword

```
if(true) {
   ...
}
while(true) {
   ...
}
switch(v) {
   ...
}
```

String literals

57% single quotes

```
var foo = 'bar';
```

```
var obj = {
  'foo': 'bar'
};
```

43% double quotes

```
var foo = "bar";
```

```
var obj = {
  "foo": "bar"
};
```

General tips

Be consistent

New project:

- Come up with a style
- Document it
- Automatically check it (tools: JSLint, JSHint, ESLint)
- Follow it consistently

Existing project:

- Follow their style (even if you don’t like it)

Code should be easy to understand

Code is written once, read many times. Treat it accordingly.

> *Programs must be written for people to read, and only incidentally for machines to execute.*
>
> —Abelson, Sussman

Code should be easy to understand

- Shorter isn't always better
  - **Familiar** + longer > **unfamiliar** + shorter
  - Humans read tokens, not characters: `redBalloon` > `rdBlln`
- Write code like a textbook
  - A guide to your mental universe
  - Complemented by documentation
- Don't be clever
- Avoid optimizing for speed or code size (esp. early on)

Commonly accepted best practices

Best practices

- Use strict mode
- Always use strict equality (===). No exceptions!
- Avoid global variables
- Always write semicolons
- Single quotes for strings (common, but not a must)

Use: 1TBS (One True Brace Style)

```
function foo(x, y, z) {
    if (x) {
        a();
    } else {
        b();
        c();
    }
}
```

Don’t use: Allman brace style

```
function foo(x, y, z)
{
    if (x)
    {
        a();
    }
    else
    {
        b();
        c();
    }
}
```

Literals are better than constructors

```
var obj = {}; // yes
var obj = new Object(); // no

var arr = []; // yes
var arr = new Array(); // no

var regex = /abc/; // yes
var regex = new RegExp('abc'); // avoid
```

Acceptable cleverness

```
// Optional parameter x
function f(x) {
    x = x || 0;
    ...
}
```

Naming entities

- Not capitalized:
  - Functions, variables: `myFunction`
  - Methods: `obj.myMethod`
  - Modules: `my_module`
- Capitalized:
  - Constructors: `MyConstructor`
  - Constants: `MY_CONSTANT`

Less mainstream rules

Camel case

Camel case:

- JavaScript: `decodeURIComponent`, `JSON`
  - Shudder: `XMLHttpRequest`
- I prefer: `processHtmlFile`

Four spaces for indentation

```
// My preference:
// 4 spaces
function abs(x) {
    if (x >= 0) {
        return x;
    } else {
        return -x;
    }
}
```

```
// Often used:
// 2 spaces
function abs(x) {
  if (x >= 0) {
    return x;
  } else {
    return -x;
  }
}
```

One variable declaration per line

```
// no
var foo = 3,
    bar = 2,
    baz;
```

```
// yes
var foo = 3;
var bar = 2;
var baz;
```

Advantages:

- Protects against some typos (forgetting a comma)
- Easier: insert, delete, rearrange lines
- Every line has context information
- Automatically indented correctly

Coercing

Use `Boolean`, `Number`, `String` (as functions) to coerce.

```
var bool = Boolean(7); // yes
var bool = !!7; // no

var num = Number('123'); // yes
var num = +'123'; // no

var str = String(true); // yes
var str = ''+true; // no
```

Keep declarations local

Close to where they are used.

Keep declarations local

```
// Often recommended:
var elem;
for (var i=0; i<arr.length; i++) {
    elem = arr[i];
    ...
}

// I prefer:
for (var i=0; i<arr.length; i++) {
    var elem = arr[i];
    ...
}
```

Keep declarations local

```
// Often recommended:
var tmp;
if (x < 0) {
    tmp = -x;
    ...
}

// I prefer:
if (x < 0) {
    var tmp = -x;
    ...
}
```

Keep declarations local

Often recommended:

```
// Only used inside loop
function helperFunc() {
  ...
}
arr.forEach(function (x) {
  ...
});
```

I prefer:

```
arr.forEach(function (x) {
  function helperFunc() {
    ...
  }
  ...
});
```

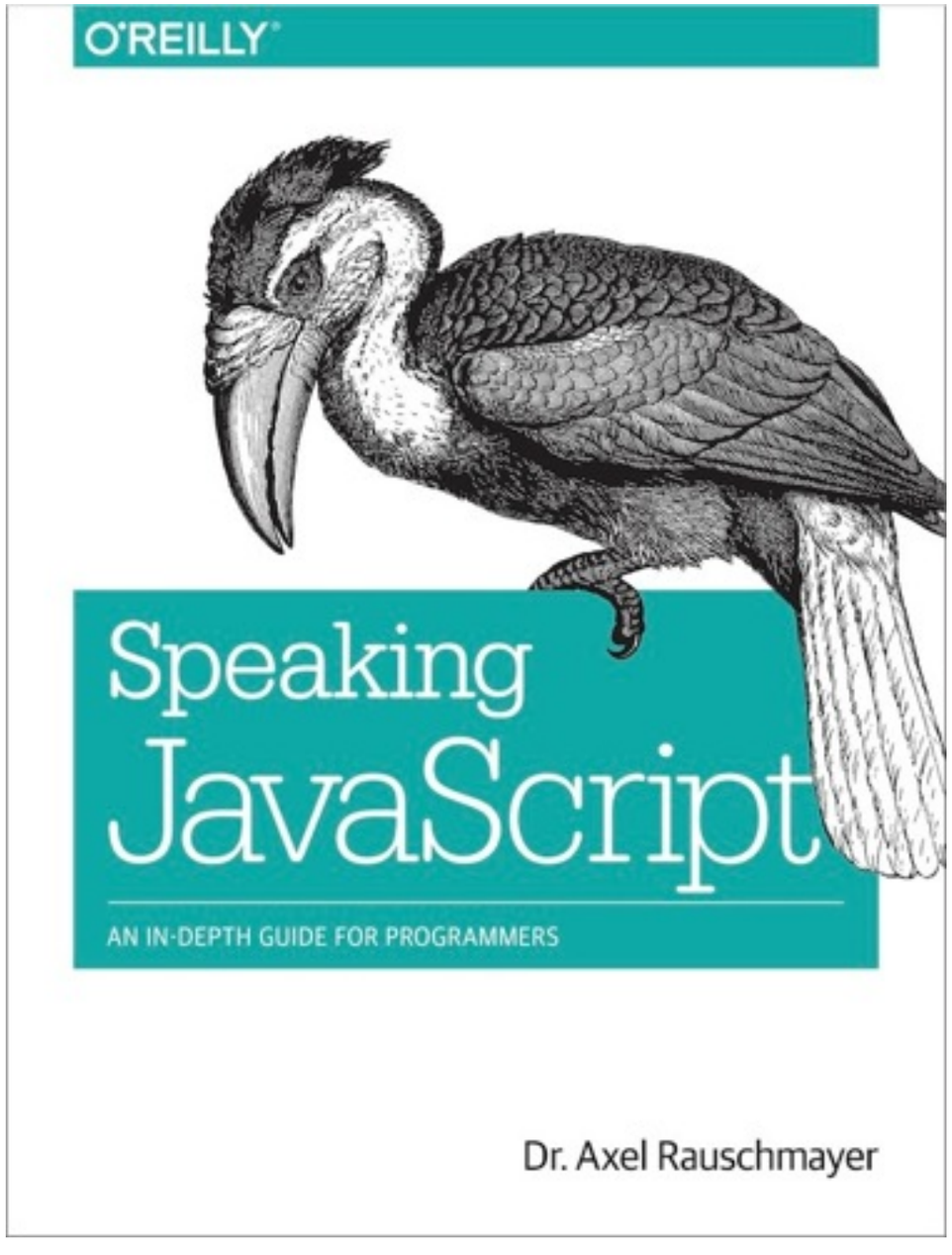
Keep declarations local

Advantages:

- Code fragments are easier to
  - understand
  - move and reuse
  - delete
- Often engines optimize automatically
  ⇒ no performance penalty

Thanks!

More info (chapter 26):
speakingjs.com/es5/



Bonus slides

Object-orientation

- Prefer constructors over other instance creation patterns
  - Mainstream
  - Optimized
  - Forward-looking (ECMAScript 6 classes are constructors)
- Avoid closures for private data
  - Less elegant code
  - But: only way to keep data completely private
- Use parens for constructor calls without arguments:

```
var foo = new Foo;  // no
var foo = new Foo();  // yes
```

Avoid `this` as implicit parameter

```
// Avoid
$('ul.tabs li').on('click',
    function () {
        var tab = $(this);
        highlightTab(tab);
        ...
    });

// Prefer
$('ul.tabs li').on('click',
    function (event) {
        var tab = $(event.target);
        highlightTab(tab);
        ...
    });
```

Avoid `this` as implicit parameter

```
// Avoid
beforeEach(function () {
    this.addMatchers({
        toBeInRange: function (start, end) {
            ...
        }
    });
});

// Prefer
beforeEach(function (api) {
    api.addMatchers({
        toBeInRange(start, end) {
            ...
        }
    });
});
```