Section 8

Programming Style
& Your Brain

Programming Style

&

Your Brain

Douglas Crockford

THINKING,

FAST AND SLOW

DANIEL

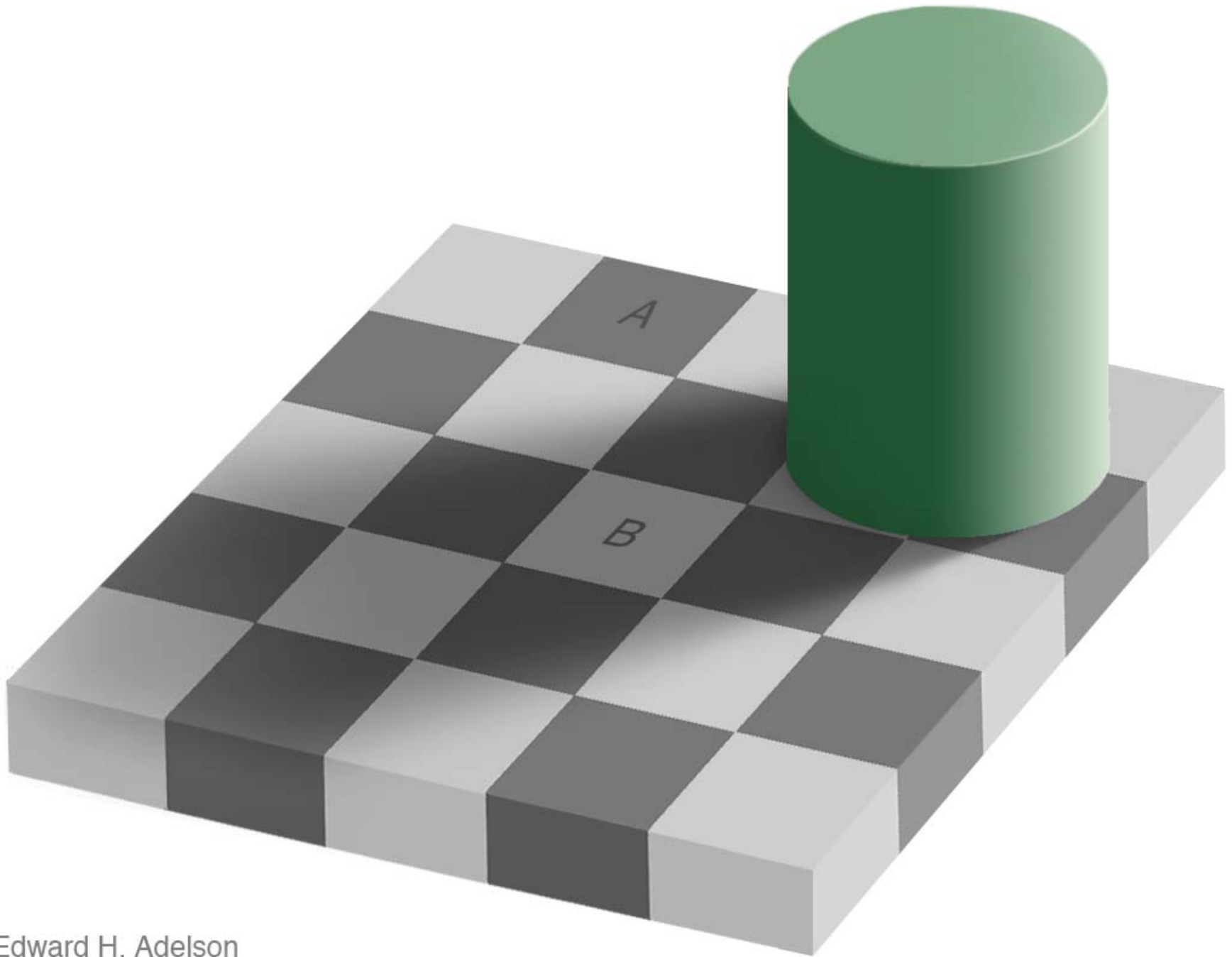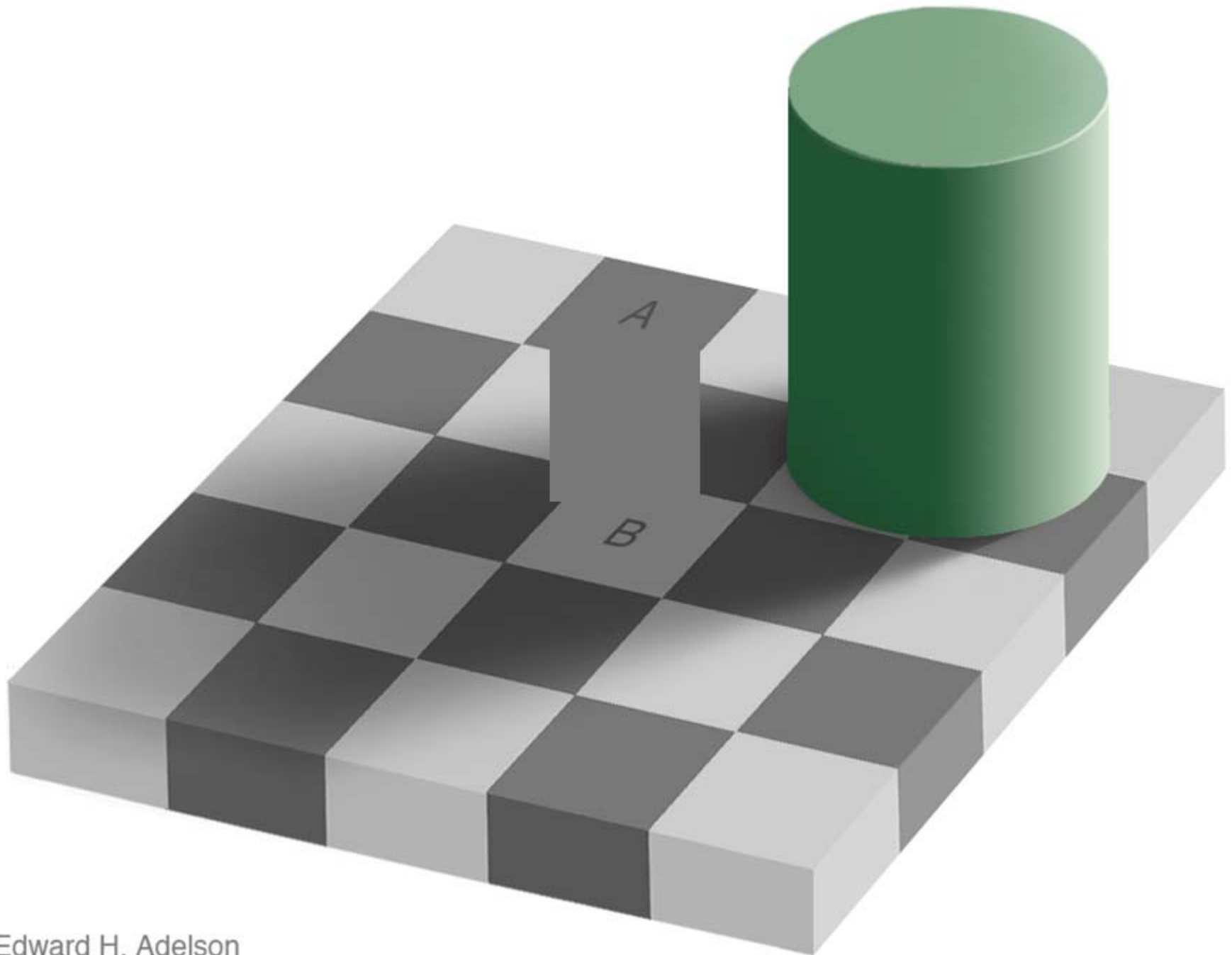# KAHNEMAN

Head.
Gut.

Two Systems.
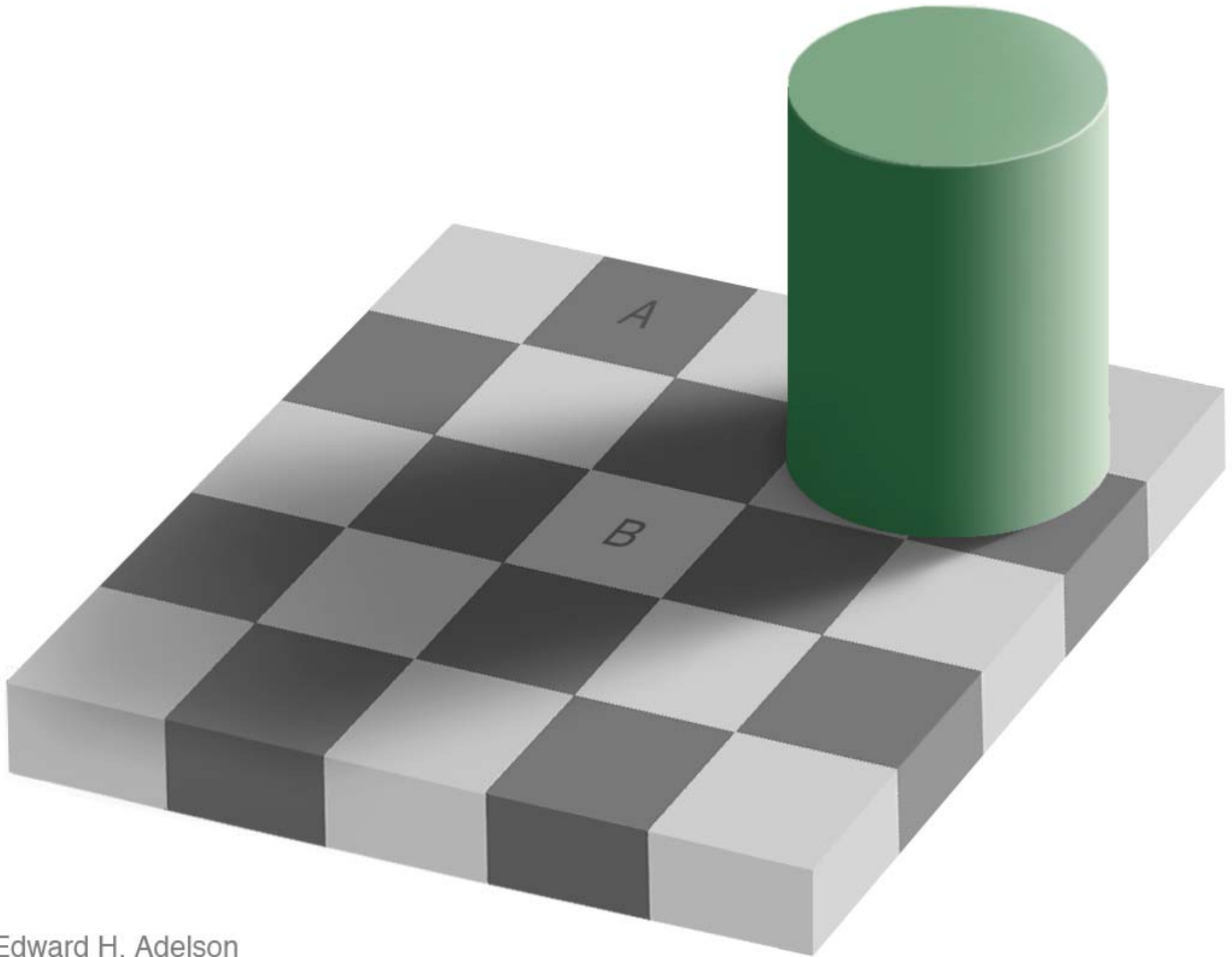
Visual Processing.

An analogy.

Edward H. Adelson

Edward H. Adelson

Edward H. Adelson

Advertising.

Tobacco.

Computer Programs.

**The most complicated things
people make.**

Artificial Intelligence.

Programming Language.

Perfection.

Hunters and Gatherers.

Programming uses Head and Gut.

Tradeoffs.

JavaScript.

Good Parts.

Bad Parts.

JSLint.

**JSLint defines a professional
subset of JavaScript.**

`http://www.JSLint.com/`

WARNING!

JSLint will hurt your feelings.

Left or Right?

```
block           block {

{                   . . . .

    . . . .     }

}
```

Left or Right?

block

{

· · · ·

}

block {

· · · ·

}

- Be consistent.

- Everyone should do it like I do.

Left or Right?

```
return

{

    ok: false

};
```

```
return {

        ok: true

};
```

- SILENT ERROR!

- Works well in JavaScript.

Prefer forms that are error resistant.

switch statement.

The fallthrough hazard.

"That hardly ever happens"

is another way of saying
"It happens".

A good style can help produce better programs.

Style is should not be about personal preference and self-expression.

THEROMANSWROTELATIN
ALLINUPPERCASEWITH
NOWORDBREAKS
ORPUNCTUATION

Medieval copyists introduced lowercase, word breaks, and punctuation.

These innovations helped reduce the error rate.

Good use of style can help reduce the occurrence of errors.

The Elements of Style
William Strunk

http://www.crockford.com/wrrrld/style.html

Programs must communicate clearly to people.

Use elements of good composition where applicable.

For example, use a space after a comma, not before.

Use spaces to disambiguate parens.

- No space before (when used to invoke a function.
- No space between a function name and a parameter list.
- One space between all other names and (.
- Wrong:

```
foo (bar);
return(a+b);
if(a=== 0) {…}
function foo (b) {…}
function(x) {…}
```

Immediately Invocable Function Expressions

```
function () {

    ...

}();    // Syntax error!
```

Immediately Invocable Function Expressions

```
(function () {

    ...

})();
```

Immediately Invocable Function Expressions

```
(function () {
    ...
})();
```

Dog balls

Immediately Invocable Function Expressions

```
(function () {

    ...

}());    // Neatness counts.
```

The Heartbreak of Automatic Semicolon Insertion

```
x = y      // <-- Missing semicolon

(function () {

    ...

}());
```

- Never rely on automatic semicolon insertion!

with statement.

```
with (o) {

    foo = koda;

}
```

- ☐ o.foo = koda;

- ☐ o.foo = o.koda;

- ☐ foo = koda;

- ☐ foo = o.koda;

with statement.

```
with (o) {

    foo = koda;

}
```

- ☐ o.foo = koda;
- ☐ o.foo = o.koda;
- ☐ foo = koda;
- ☐ foo = o.koda;

I am not saying that it isn't useful.
I am saying that there is never a case
where it isn't confusing.

Confusion must be avoided.

Transitivity? What's That?

```
0 == ''              // true

0 == '0'             // true

'' == '0'            // false

false == 'false'     // false

false == '0'         // true

" \t\r\n " == 0      // true
```

Always use `===`, never `==`.

If there is a feature of a language that is sometimes problematic, and if it can be replaced with another feature that is more reliable, then always use the more reliable feature.

Multiline string literals

```
var long_line_1 = "This is a \
long line"; // ok


var long_line_2 = "This is a \
long line"; // syntax error
```

Avoid forms that are difficult to distinguish from common errors.

"That was intentional."

"I know what I'm doing."

```
if (a = b) {…}


a = b;
if (a) {…}


if (a === b) {…}
```

Make your programs look like what they do.

Scope.

Block scope v function scope.

var statement.

- It gets split into two parts:

    The declaration part gets hoisted to the top of the function, initializing with `undefined`.

    The initialization part turns into an ordinary assignment. So

    ```
    function foo() {

        ...

        var myVar = 0, myOtherVar;
    ```

- Expands into

    ```
    function foo() {

        var myVar = undefined,

            myOtherVar = undefined;

        ...

        myVar = 0;
    ```

Declare all variables at the top of the function.

Declare all functions before you call them.

for (var i …) {…}

Variable `i` is not scoped to the loop.

Write in the language you are writing in.

Let there be let.

Global variables.

- Global variables are evil.

- Avoid global variables.

- When using global variables, be explicit.

  `UPPER_CASE`

- Global variables should be as rare as hens teeth and stick out like a sore thumb.

new prefix

Forgetting new causes a
constructor to clobber global
variables without warning.

Fixed in ES5/strict.

Constructor functions should be named with InitialCaps.

Nothing else should be named with InitialCaps.

```
var a = b = 0;

var a = 0, b = 0;

b = 0;
var a = b;
```

Write in a way that clearly communicates your intent.

+ +

```
++

x += 1
```

```
++

x += 1

x++
```

```
++

x += 1

x̶+̶+̶

++x
```

```
++x;
++x;


x += 2;
```

For no cost, by adopting a more rigorous style, many classes of errors can be automatically avoided.

```
if (a) b(); c();
```

```
if (a) b(); c();

if (a) {b(); c();}

if (a) {b();} c();
```

As our processes become more agile, our coding must be more resilient.

Programming is the most complicated thing that humans do.

Computer programs must be perfect.

Humans are not good at perfect.

Designing a programming style demands discipline.

It is not selecting features because they are liked, or pretty, or familiar.

The Abyss

The JSLint style was driven by the need to automatically detect defects.

Forms that can hide defects are considered defective.

Language Subsetting.

Only a madman would use all of C++.

There will be bugs.

Do what you can to move the odds
to your favor.

Good style is good for your gut.

Thank you and good night.