

www.qconferences.com  
www.qconbeijing.com



伦敦 | 北京 | 东京 | 纽约 | 圣保罗 | 上海 | 旧金山

London · Beijing · Tokyo · New York · Sao Paulo · Shanghai · San Francisco

# QCon全球软件开发大会

International Software Development Conference

# Programming Style & Your Brain

Douglas Crockford

THINKING,  
FAST AND SLOW



DANIEL  
KAHNEMAN

WINNER OF THE NOBEL PRIZE IN ECONOMICS

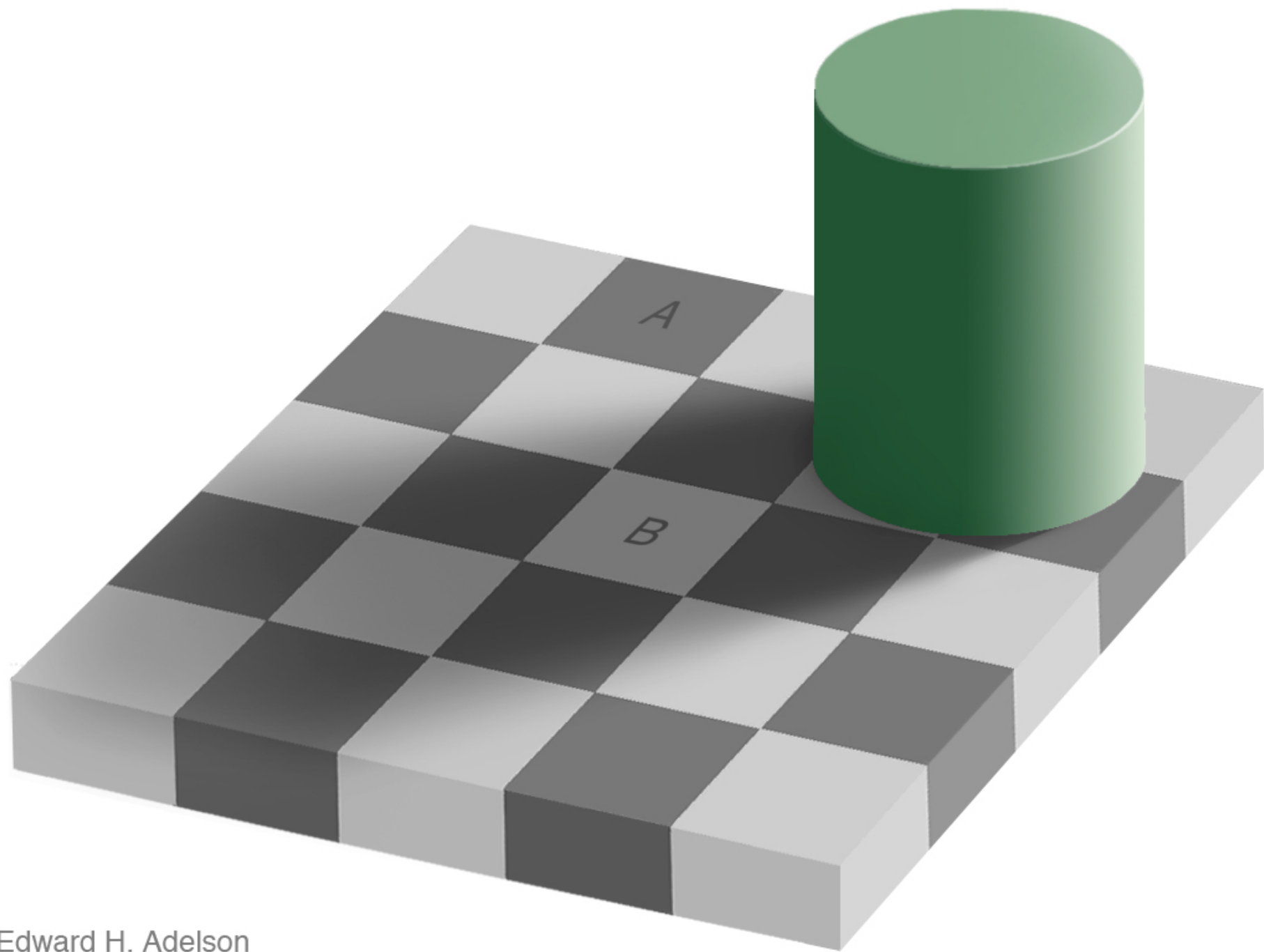
Head.

Gut.

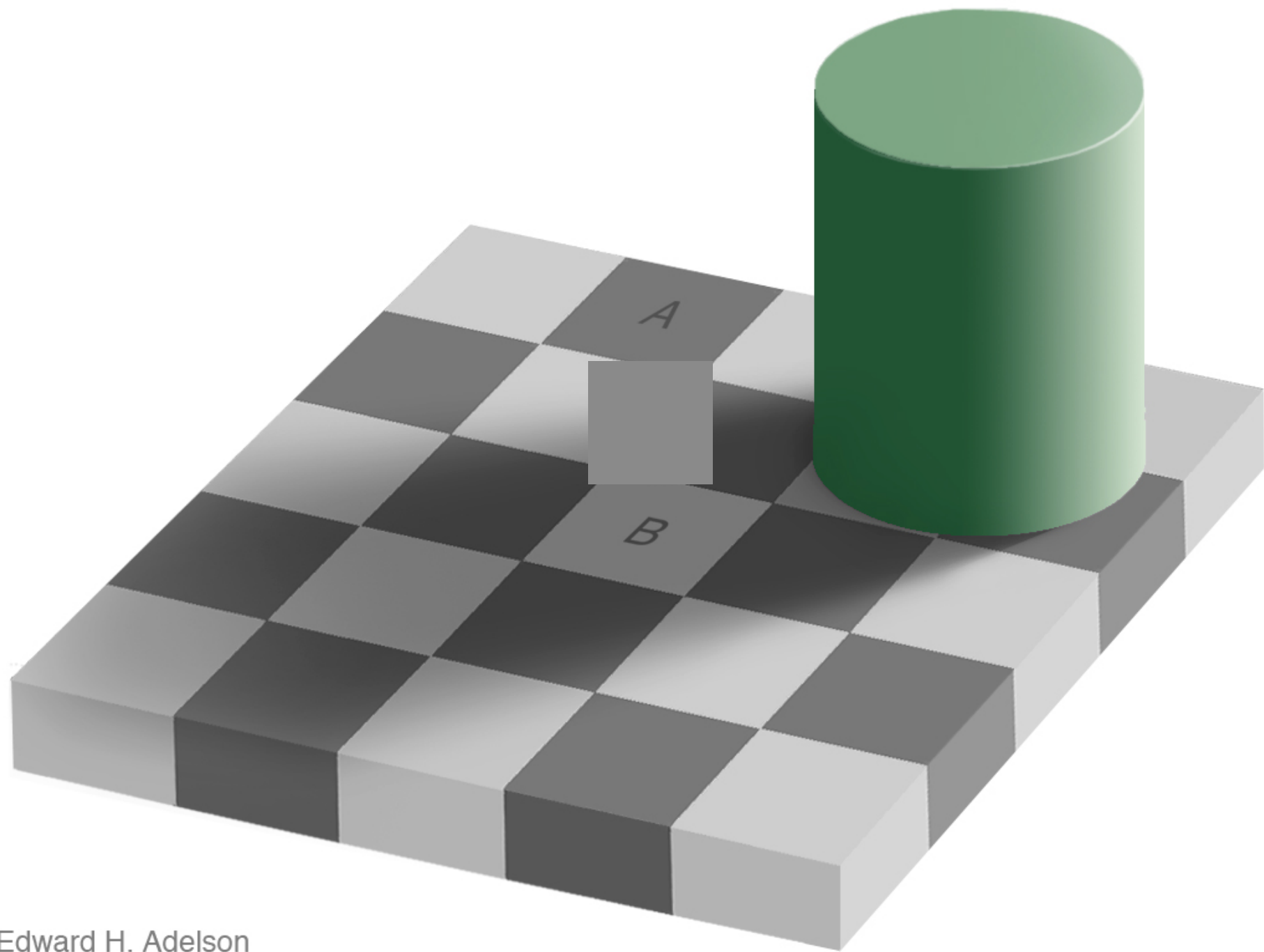
Two Systems.

# Visual Processing.

An analogy.









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
```

```
};
```

- SILENT ERROR!

```
return {
```

```
    ok: true
```

```
};
```

- Works well in JavaScript.

```
return
```

```
{
```

```
    ok: false
```

```
};
```

```
return; // semicolon insertion
{
    ok: false
};
```



```
return;
```

```
{ // block
```

```
    ok: false
```

```
};
```

```
return;
```

```
{
```

```
    ok: false // label
```

```
};
```

```
return;
```

```
{ // useless
```

```
    ok: false // expression
```

```
}; // statement
```

```
return;
```

```
{
```

```
    ok: false; // semicolon
```

```
};           // insertion
```

```
return;
```

```
{
```

```
    ok: false;
```

```
}; // empty statement
```

```
return;
```

```
{  // unreachable statement
```

```
    ok: false;
```

```
}
```

```
return
```

```
{
```

```
    ok: false
```

```
};
```

```
return;
```

```
{
```

```
    ok: false;
```

```
}
```

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.

THE ROMANS WROTE LATIN  
ALL IN UPPER CASE WITH  
NO WORD BREAKS  
OR PUNCTUATION

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 between a function name and (.
- 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.

```
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 1et.

# 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.

```
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.

++

++

x += 1



++

x += 1

x++

++

x += 1

~~x++~~

++x

```
++x;
```

```
++x;
```

```
x += 2;
```

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

# Bad stylists

- Under educated.
- Old school.
- Thrill seeker.
- Exhibitionist.

“That was intentional.”

“I know what I’m doing.”

# Performance.

- Performance specific code is usually crufty.
- Clean code is easier to reason about.
- Premature optimization is the root of all evil. DONALD KNUTH
- Most of the code has a negligible impact on performance. Only optimize the code that is taking the time.
- Algorithm replacement is vastly more effective than code fiddling

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.



# The Analytical Language of John Wilkins

## Jorge Luis Borges

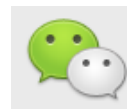
These ambiguities, redundancies and deficiencies remind us of those which doctor Franz Kuhn attributes to a certain Chinese encyclopedia entitled The Celestial Emporium of Benevolent Knowledge. In its remote pages it is written that the animals are divided into

- a. belonging to the Emperor
- b. embalmed
- c. trained
- d. piglets
- e. sirens
- f. fabulous
- g. stray dogs
- h. included in this classification
- i. trembling like crazy
- j. innumerable
- k. drawn with a very fine camelhair brush
- l. et cetera
- m. just broke the vase
- n. from a distance look like flies





@InfoQ



infoqchina

软件  
正在改变世界!