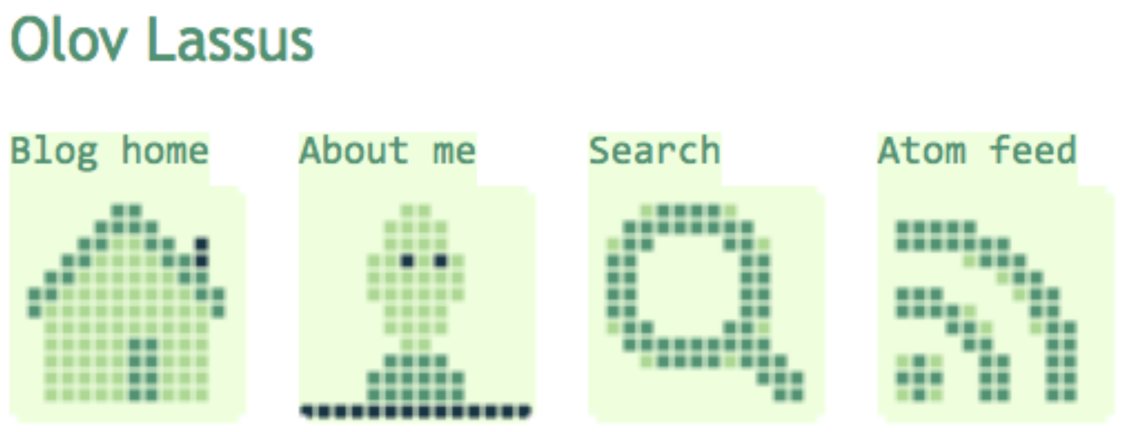


JavaScript: the subsets we use

http://lassus.se

@olov

“Easier to reason about”

Assumptions

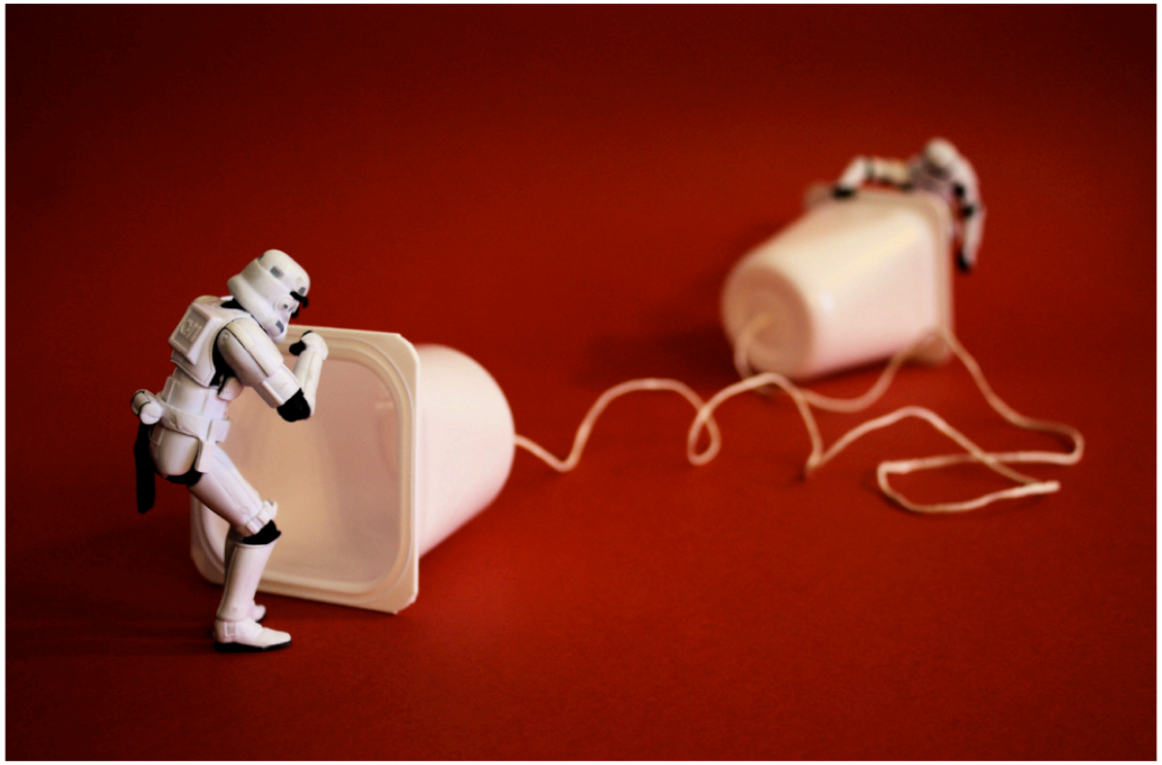
We make mistakes while writing code

"Arinocdcg to rencet rseaerch, the hmuan brian is plrectfey albe to raed colmpex pasasges of txet caiinontng wdors in whcih the lrettes hvae been jmblued, pvioedrd the frsit and lsat leetrts rmeian in teihr crcerot piiotsons."

The brain comes with built-in error correction

We make mistakes while writing code

We also make mistakes while troubleshooting code



A disconnect between assumptions and reality

Learn from them to shrink the disconnect gap

How do we reduce those mistakes?

Atul Gawande: The Checklist Manifesto

“Your mind doesn’t think of a bayonet in San Francisco”

`Assert()`

Verifying assumptions

```
function renameIdentifier(node, name) {
   Assert(node.type === tkn.IDENTIFIER);
   /* ... */
}

Assert(obj.state === FREE);

Assert(i >= 0 && j < len && i <= j);
```

More robust
Easier to reason about

http://blog.lassus.se/2011/03/c-style-assertions-in-javascript-via.html

Some JS assumption jammers

- Undefined and null
- Boxed types
- Mixing strings and numbers in arithmetic ops
- Function scope, not block scope
- Global pollution
- Falsy values
- Silent errors (as opposed to failing fast)

```
var length = 3;
console.log(lenght); // prints 3
```

Introducing a harmony proposal: ES6 spelling corrector

```
var length = 3;
console.log(lenght); // prints 3
```

Nah, that’s ridiculous

```
var length = 3;
console.log(lenght); // prints 3
```

The language shouldn’t encourage sloppiness by guessing for us

+

Right?

a + b

- Convert a and b to primitives
  - (i.e. undefined, null, boolean, number, string)
- If at least one of them is a string, then convert both to strings and concatenate
- Else convert both to numbers and add

a + b

- Convert a and b to primitives
  - (i.e. undefined, null, boolean, number, string)
- If at least one of them is a string, then convert both to strings and concatenate
- Else convert both to numbers and add

```
Number(undefined)
Number(null)
Number(false)
Number(“”)?
```

QUICK!

a + b

- Convert a and b to primitives
  - (i.e. undefined, null, boolean, number, string)
- If at least one of them is a string, then convert both to strings and concatenate
- Else convert both to numbers and add

a + b

- Convert a and b to primitives
  - (i.e. undefined, null, boolean, number, string)
  - Specifically, call a.valueOf() and a.toString() in that order
- If at least one of them is a string, then convert both to strings and concatenate
- Else convert both to numbers and add

a + b

- Convert a and b to primitives
  - (i.e. undefined, null, boolean, number, string)
  - Specifically, call a.valueOf() and a.toString() in that order
    - Unless a is a Date. In that case call a.toString() and a.valueOf() in that order.
- If at least one of them is a string, then convert both to strings and concatenate
- Else convert both to numbers and add

Alternative a + b

- Add numbers
- Concatenate strings

```
lenght = 10; // introduces global
```

```
var y; // initialized to undefined
var x = 1 + y; // NaN
```

Been there, done that

```
“use strict”; // ES5 strict mode
lenght = 10; // throws

“use restrict”; // restrict mode for JS
var y; // initialized to undefined
var x = 1 + y; // throws
```

ES5 strict mode (which is awesome), restrict mode for JavaScript

http://restrictmode.org
http://jsshaper.org

;

You have a choice and it's easy to verify statically

===

You have a choice and it’s easy to verify statically

< <= >= >

+ - * / %

& | ^ ~v

<< >> >>>

-v ++v v++ --v v--

A magic hat full of surprises

```
<  <=  >=  >
+  -  *  /  %
&  |  ^  ~v
<<  >>  >>>
-v  ++v  v++  --v  v--
```

Restrict mode is an attempt to take away some of that magic

```
< <= >= >
+ - * / %
& | ^ ~v
<< >> >>>
-v ++v v++ --v v--
```

Goal: assumption-friendlier, more robust and easier to reason about

```
< <= >= >
+ - * / %
& | ^ ~v
<< >> >>>
-v ++v v++ --v v--
```

Goal: subset semantics, reach the famous 99%

< <= >= >

+ - * / %

& | ^ ~v

<< >> >>>

-v ++v v++ --v v--

strings or numbers,
never a mix

```
< <= >= >
+ - * / %
& | ^ ~v
<< >> >>>
-v ++v v++ --v v--
```

strings or numbers,
in any combination

< <= >= >

+ - * / %

& | ^ ~v

<< >> >>>

-v ++v v++ --v v--

numbers

```
== !=
=== !==
+v !v
&& || ?:
o[k]
```

A friendly deterministic face
Just like before

Restrict mode idea

- I promise to limit myself to this subset
- A checker inserts type-assertions in my program
- I write tests just like before
- Whenever I break the subset promise, the program fails fast
- A restrict mode clean program executes identically with or without the checker, so I deploy the original (non-checked) program

```
"use strict"; "use restrict";

function average(x, y) {
  return (x + y) / 2;
}

var x;
print(average(1, 2));
print(average(1, "2"));
print(average(1, x));
```

Demo
restrictmode.org/try

Easier to reason about

- When I read source code that has the “use restrict” directive
  - I have an easier time reasoning about the program
  - Just like Assert’s
- When I need to fix bugs in it
  - The assumption versus reality disconnect just became a lot smaller

Is it “the right” subset?

- It must be helpful and can’t get in the way
- Applied to existing projects, we’d like the subset mismatch to be tiny

Is it “the right” subset?

- **v8bench**: ok, found a bad practice (using strings in arithmetic’s)
- **Kraken**: ok, found the NaN bug
- **jQuery**: ok, found an undefined bug
- **JSLint**: ok, found a debatable practice (returning this from String method)

For this little experiment, “yes”. Finding issues was unexpected.

```
"use strict"; // try removing me

Number.prototype.isPrime = function() {
  Assert(this <= 10); // TODO larger primes
  return this === 2 || this === 3 ||
         this === 5 || this === 7;
}

print((5).isPrime());
```

ES5 strict mode primitive-this considered harmful

JSShaper

- An extensible framework for syntax shaping (syntax tree transformation)
- jsshaper.org
- The restrict-mode checker is a plugin
- A few other plugins
  - Asserter
  - Bitwiser
  - Watcher
  - Yielder (C. Scott Ananian)

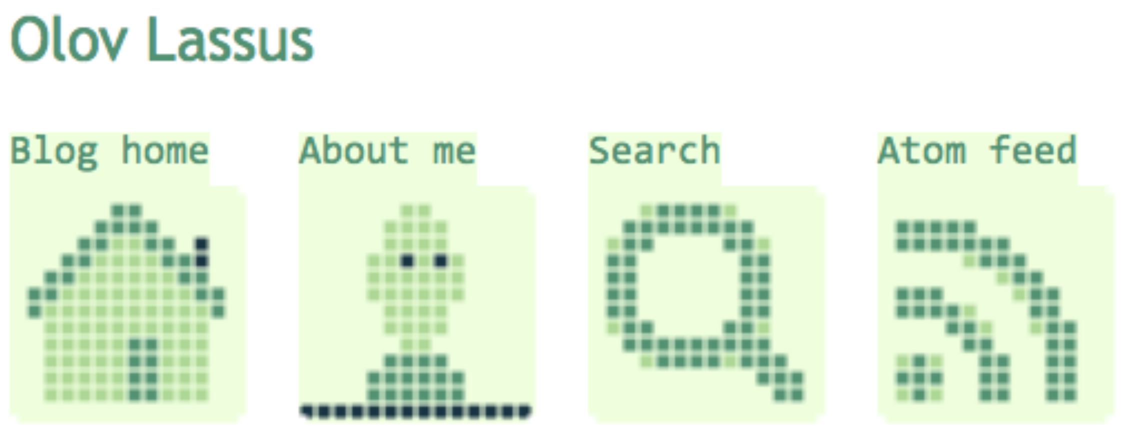
If you want to “use restrict”

- Toy around on restrictmode.org/try
- Download JSShaper and do post-edit processing of your code with the checker
  - Before running your test suite if not always
- I’d love feedback on how the subset matches the code you write
- No lock-in, just remove the “use restrict” directive and nobody needs to know you went there in the first place
- It’s still JavaScript

“Easier to reason about”

- Choose your subset and style guide
- Tools to help verifying those
- Assertive-rich programming
- Prioritize getting your API:s right
  - “If you program, you are an API designer”
- Challenge your assumptions
- Reading code is a skill, practice it and learn from others

And have fun



Thank you

http://lassus.se

@olov