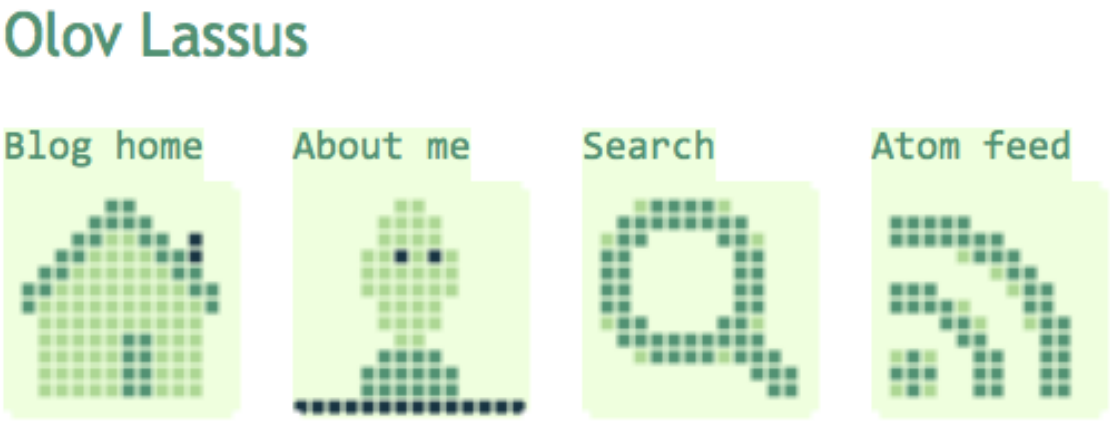
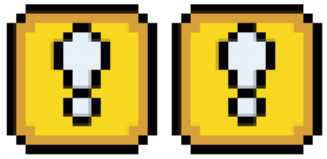



**JS – from good to great (an ode to assert)**

http://lassus.se

@olov

# Correct code is great



**Code that crashes needs improvement**

**Incorrect code that keeps on running is a recipe for disaster**

*Crashing is a good thing!*

# Fail-fast is key to robust programs

**Of course, another key to robust programs is getting the code right**

# But let’s admit

# We make coding mistakes all the time

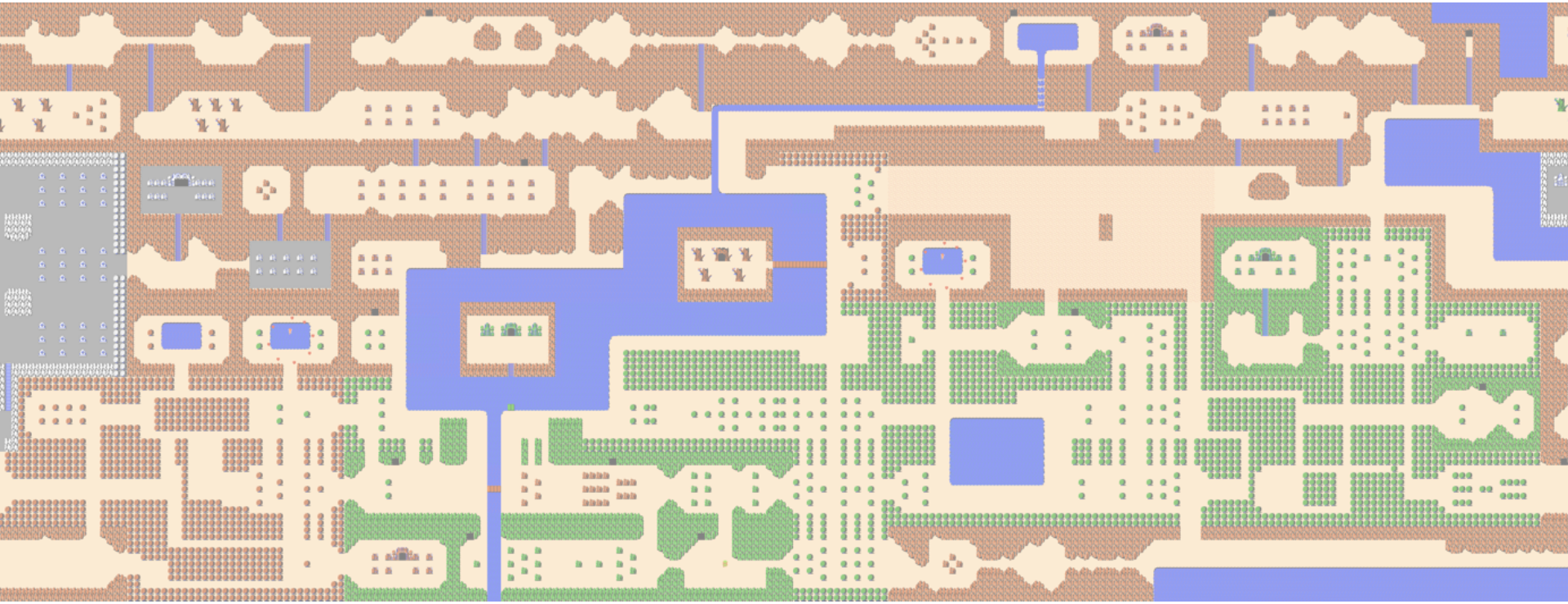
**We even make mistakes when we're troubleshooting code**

# Assumptions

# A disconnect between assumptions and reality

“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.”

**Let me fix that for you /brain**



# “Easier to reason about”

```
Assert(assumption);
```

**Documenting and verifying assumptions, failing faster**

```
Assert(obj.state === FREE);

Assert(arguments.length === 2);

Assert(typeof cost === "number" && isFinite(cost));
```

# Assert examples

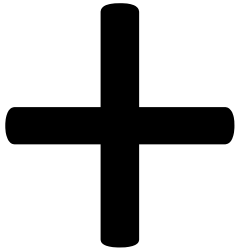
http://blog.lassus.se/2011/03/c-style-assertions-in-javascript-via.html

# Assert

- Easier to reason about
- More robust code that fails fast
- Simplifies refactoring and code changes
- Makes it easier to write unit-tests
- Doesn’t slow you down

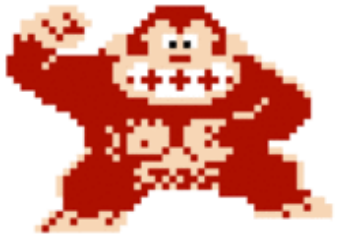
# Some JS assumption jammers

- Undefined and null
- Boxed types
- Function scope, not block scope
- Global pollution
- Falsy values
- Keep on running (as opposed to fail fast)
- Mixing strings and numbers in arithmetic ops



Keep on running

# Let me fix that for you /plus

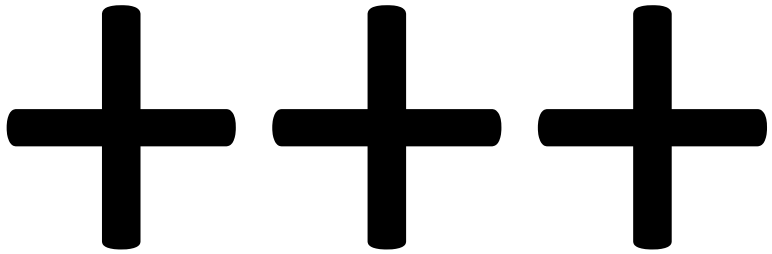


**Weak typing in a dynamically typed language is not very helpful**

# 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 them
- Else convert both to numbers and add them

**Yours to keep**



**Where’s my triple plus?**

< <= >= >

+ - * / %

& | ^ ~v

<< >> >>>

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

**A magic hat full of surprises**

< <= >= >

+ - * / %

& | ^ ~v

<< >> >>>

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

**What if we could restrict the hat to something less surprising?**

< <= >= >

+ - * / %

& | ^ ~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**

```
// what if instead of writing this
var x1 = 1 + y;

// we would write this?
var x2 = __add(1, y);

function __add(l, r) {
  Assert(typeof l === “number” || typeof l === “string”);
  Assert(typeof r === “number” || typeof r === “string”);
  return l + r;
}
```

**We can’t change + semantics, but we can call the strong __add instead**

**Perhaps we can change + after all?**

```
"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

# 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

# Easier to reason about

- When I read source code that has the “use restrict” directive
  - I have an easier time understanding it
  - 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 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, catches the old 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.

# JSShaper

- An extensible framework for syntax tree transformation (tree shaping)
- jsshaper.org – MIT license
- Runs in node and the browser, even on the fly
- The restrict mode checker is a plugin
- A few other plugins
  - Asserter
  - Bitwiser
  - Watcher
  - Yielder (C. Scott Ananian)

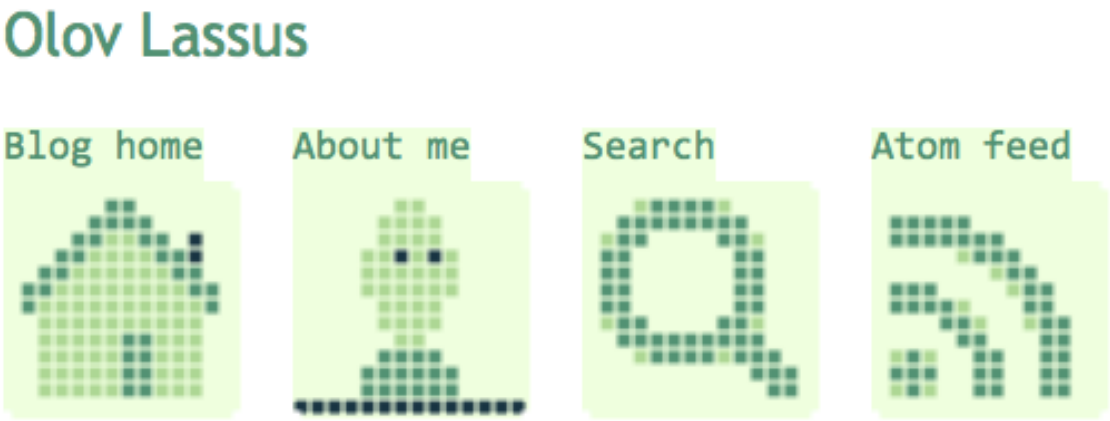
# If you want to “use restrict”

- Toy around on restrictmode.org/try
- Download JSShaper and put the restrict mode checker in your toolchain
  - Use it when running your test suite if not always
- Does it match your subset? Let me know!
- No lock-in, just remove the “use restrict” directive and nobody needs to know you went there in the first place
- It’s still just JavaScript

# “Easier to reason about”

- Choose *your* subset and style guide
- Use tools to help verifying those
- Consider sprinkling Assertions in your code
- Prioritize getting your API:s right
- Challenge your assumptions
- Reading code is a skill, practice it and learn from others

# And have fun




**Thank you**

http://lassus.se

@olov