# Functional JavaScript

Douglas Crockford

Yahoo! Inc.

# The World's Most Misunderstood Programming Language

A language of many contrasts.

The broadest range of programmer skills of any programming language.

From computer scientists

to cut-n-pasters

and everyone in between.

# Complaints

- "JavaScript is not a language I know."
- "The browser programming experience is awful."
- "It's not fast enough."
- "The language is just a pile of mistakes."

Hidden under a huge steaming pile of good intentions and blunders is an elegant, expressive programming language.

JavaScript has good parts.

JavaScript is succeeding very well in an environment where Java was a total failure.

# Influences

- Self
  - prototypal inheritance
  - dynamic objects
- Scheme
  - lambda
  - loose typing
- Java
  - syntax
  - conventions
- Perl
  - regular expressions

# Bad Parts

- Global Variables
- + adds and concatenates
- Semicolon insertion
- typeof
- with and eval
- phony arrays
- == and !=
- false, null, undefined, NaN

# Transitivity? What's That?

```
• 0 == ''                // true
• 0 == '0'               // true
• '' == '0'              // false
• false == ''            // true
• false == '0'           // true
• false == undefined     // false
• false == null          // false
• null == undefined      // true
• " \t\r\n " == 0        // true
• " \t\r\n " == ""       // false
```

# Good Parts

- Lambda
- Dynamic Objects
- Loose Typing

- ```
  (define foo (lambda (a b c)
        (body)
  )
  ```

- ```
  var foo = function (a, b, c) {
     return body;
  };
  ```

- `(foo a b c)`
- `foo(a, b, c)`

- **(cond (***p1 e1***) (***p2 e2***) ... (else** *en***))**
- *p1* **?** *e1* **:** *p2* **?** *e2* **:** ... *en*

- `(quote (a b c))`
- `['a', ['b', ['c']]]`

# Y Combinator

```
var Y = function (le) {
      return function (f) {
          return f(f);
      }(function (f) {
          return le(function (x) {
              return f(f)(x);
          });
      });
};
```

# Inheritance

- Inheritance is object-oriented code reuse.
- Two Schools:
  - Classical
  - Prototypal

# Prototypal Inheritance

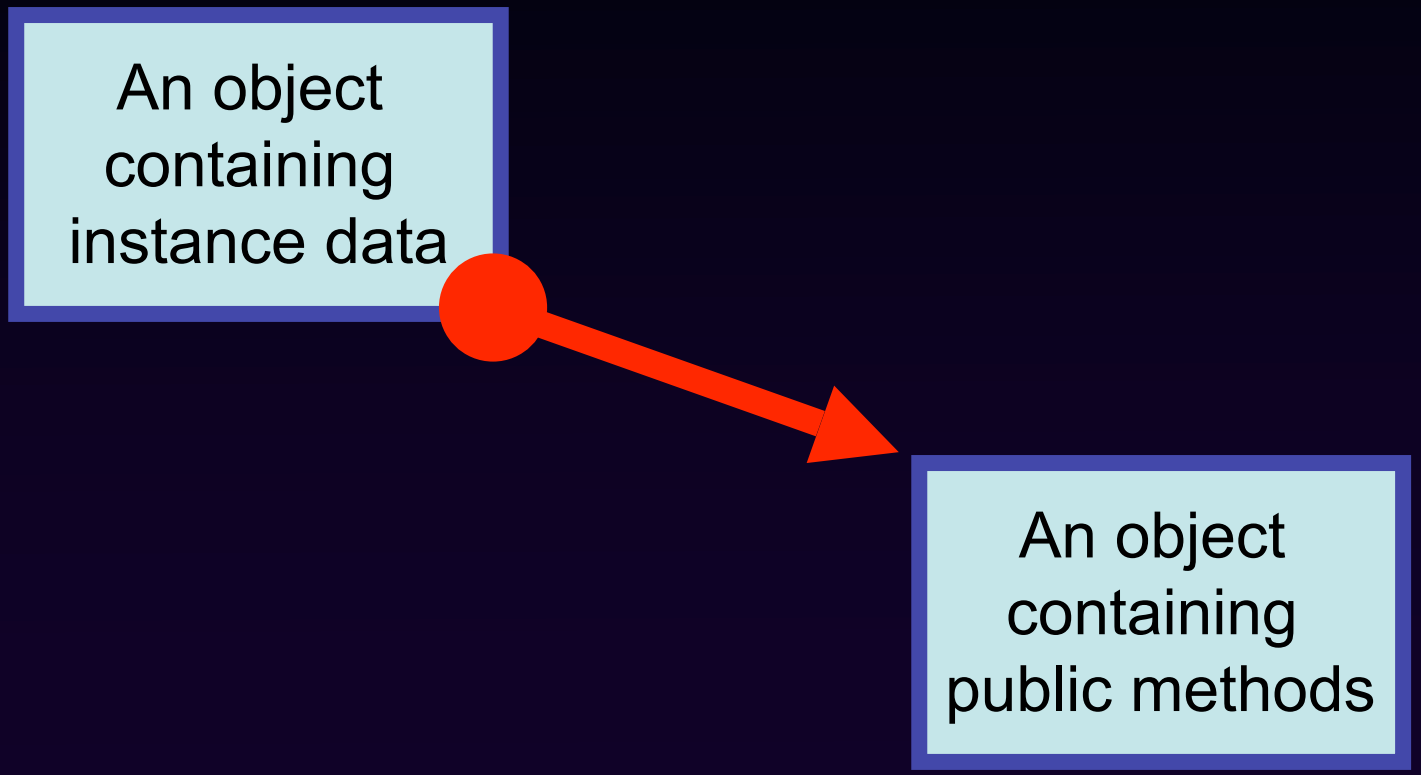
- Class-free.
- Objects inherit from objects.
- An object contains a link to another object: Delegation. Differential Inheritance.

```
var newObject =
        Object.create(oldObject);
```

# Objects

- Objects are general containers.
- Key/value pairs.
- Keys are strings.
- Values are any value.
- Objects can be modified at any time.
- Objects are passed by reference.
- An object can inherit from another object.

# Prototypes


An object containing instance data
An object containing public methods

- Public methods are functions.
- A pseudoparameter `this` is bound to the invoked object.

# Object literals

- Simple quasiliteral constructor for objects.
- ```
  {
      name : value,
      name : value
  }
  ```
- Inspiration for the JSON Data Interchange Format. **`www.JSON.org/`**
  ```

# Closure

```
var digit_name = function () {
    var names = ['zero', 'one', 'two',
        'three', 'four', 'five', 'six',
        'seven', 'eight', 'nine'];

    return function (n) {
        return names[n];
    };
}();
alert(digit_name(3));    // 'three'
```

# A Module Pattern

```
var singleton = function () {
    var privateVariable;
    function privateFunction(x) {
        ...privateVariable...
    }
    return {
        firstMethod: function (a, b) {
            ...privateVariable...
        },
        secondMethod: function (c) {
            ...privateFunction()...
        }
    };
}();
```

Module pattern is easily transformed into a powerful constructor pattern.

# Power Constructors

1. Make an object.
   - Object literal
   - `new`
   - `Object.create`
   - call another power constructor

# Power Constructors

1. Make an object.
   - Object literal, `new`, `Object.create`, call another power constructor
2. Define some variables and functions.
   - These become private members.

# Power Constructors

1. Make an object.
   - Object literal, `new`, `Object.create`, call another power constructor
2. Define some variables and functions.
   - These become private members.
3. Augment the object with privileged methods.

# Power Constructors

1. Make an object.
   - Object literal, `new`, `Object.create`, call another power constructor
2. Define some variables and functions.
   - These become private members.
3. Augment the object with privileged methods.
4. Return the object.

# Step One

```
function myPowerConstructor(x) {
    var that = otherMaker(x);
}
```

# Step Two

```
function myPowerConstructor(x) {
    var that = otherMaker(x);
    var secret = f(x);
}
```

# Step Three

```
function myPowerConstructor(x) {
    var that = otherMaker(x);
    var secret = f(x);
    that.priv = function () {
        ... secret x that ...
    };
}
```

# Step Four

```
function myPowerConstructor(x) {
    var that = otherMaker(x);
    var secret = f(x);
    that.priv = function () {
        ... secret x ...
    };
    return that;
}
```

# Closure

- A function object contains

  A function (name, parameters, body)

  A reference to the environment in which it was created (context).

- This is a very good thing.

# Values

- Numbers
- Strings
- Booleans
- Objects & Arrays
- Functions
- Falsy values:
  false, 0, "", null, undefined, NaN

# History

Thirteen years ago in a valley
30 miles to the south...

# Working with the Grain

# A Personal Journey

Beautiful Code

# JSLint

- JSLint defines a professional subset of JavaScript.
- It imposes a programming discipline that makes me much more confident in a dynamic, loosely-typed environment.
- **`http://www.JSLint.com/`**

# WARNING!

JSLint will hurt your feelings.

# Unlearning Is Really Hard

Perfectly Fine == Faulty

It's not ignorance does so much damage; it's knowin' so derned much that ain't so.

Josh Billings

The Very Best Part:

# Stability

No new design errors

since 1999!

# Coming Soon

- [ES3.1] ECMAScript Fourth Edition
- Corrections
- Reality
- Support for object hardening
- Strict mode for reliability
- Waiting on implementations

# Not Coming Soon

- [ES4] This project has been cancelled.

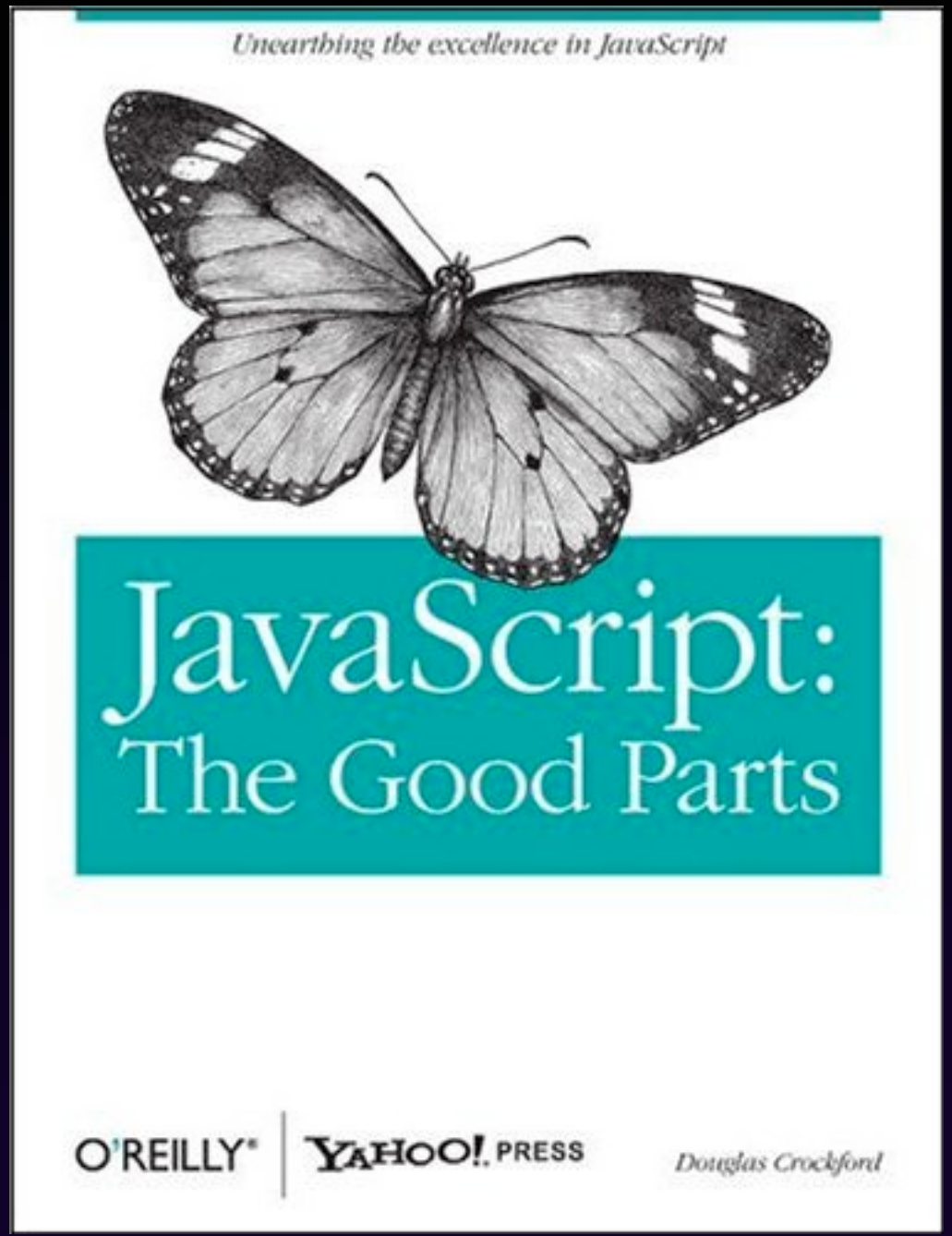
- Instead, [ES-Harmony].
- So far, this project has no defined goals or rules.

# Safe Subsets

- The most effective way to make this language better is to make it smaller.
- FBJS
- Caja & Cajita
- ADsafe
- These subsets will be informing the design of a new secure language to replace JavaScript.

# The Good Parts

- Your JavaScript application can reach a potential audience of billions.
- If you avoid the bad parts, JavaScript works really well. There is some brilliance in it.
- It is possible to write good programs with JavaScript.


Unearthing the excellence in JavaScript
JavaScript:
The Good Parts
O'REILLY
YAHOO! PRESS
Douglas Crockford