

The Better Parts



Antoine de Saint-Exupéry

ANTOINE DE SAINT-EXUPÉRY

The Little Prince



Il semble que la perfection soit atteinte
non quand il n'y a plus rien à ajouter,
mais quand il n'y a plus rien à retrancher.

Antoine de Saint-Exupéry
Terre des Hommes, 1939

It seems that perfection is attained
not when there is nothing more to add,
but when there is nothing more to subtract.

Good Parts

It seems that perfection is attained
not when there is nothing more to add,
but when there is nothing more to subtract.

If a feature is sometimes
useful and sometimes
dangerous

and if there is a better option then
always use the better option.

We are not paid to use every
feature of the language.

We are not paid to use every
feature of the language.

We are paid to write
programs that work well
and are free of error.

A good programming
language should teach you.

I made every mistake with
JavaScript you could make.



JSLint

Unearthing the excellence in JavaScript



JavaScript: The Good Parts

O'REILLY*

| YAHOO!.PRESS

Douglas Crockford

Arguments against good parts

- Just a matter of opinion.
- Every feature is an essential tool.
- I have a right to use every feature.
- I need the freedom to express myself.
- I need to reduce my keystrokes.
- It is an insult to suggest that I would ever make a mistake with a dangerous feature.
- There is a good reason those features were added to the language.

“Foot Guns”

Brendan Eich

The purpose of a programming language is to aid programmers in producing error-free programs.

It is not possible to write good
programs in JavaScript.

It is not only possible to write
good programs in JavaScript,
it is necessary.

~~It is not possible to write good
programs in JavaScript.~~

The fantasy of infallibility.

The futility of faultlessness.

Danger Driven Development



A. The time it takes to write the code.

B. The time it takes to make the code work right.

Always take the time to code well.

New Good Parts in ES6

- Proper tail calls:

```
return func() ;
```

New Good Parts in ES6

- Proper tail calls
- Splat... aka ...rest aka ...spread

```
function curry(func, ...first) {  
    return function (...second) {  
        return func(...first, ...second);  
    };  
}
```

```
function curry(func) {  
    var slice = Array.prototype.slice, args = slice.call(arguments, 1);  
    return function () {  
        return func.apply(null, args.concat(slice.call(arguments, 0)));  
    };  
}
```

New Good Parts in ES6

- Proper tail calls
- Splat
- Module

New Good Parts in ES6

- Proper tail calls
- Splat
- Module
- Let

`let` is the new `var`

New Good Parts in ES6

- Proper tail calls
- Splat
- Module
- Let
- Destructuring

```
let {that, other} = some_object;
```

```
let that = some_object.that, other = some_object.other;
```

New Good Parts in ES6

- Proper tail calls
- Splat
- Module
- Let
- Destructuring
- WeakMap

Good Parts Reconsidered

- I stopped using `new` years ago.
Use `Object.create` instead.

Good Parts Reconsidered

- I stopped using `new` years ago.
- I have stopped using `Object.create`.

Good Parts Reconsidered

- I stopped using `new` years ago.
- I have stopped using `Object.create`.
- I have stopped using `this`. [ADsafe.org]



Loops Reconsidered

- I don't use **for**.
I now use *array*.**forEach** and its many brothers.
- I don't use **for in**. I now use **Object.keys(object) .forEach**.
- ES6 will have proper tail calls.
At that point I will stop using **while**.

```
function repeat(func) {  
    while (func() !== undefined) {  
    }  
}
```

```
function repeat(func) {  
    if (func() !== undefined) {  
        return repeat(func);  
    }  
}
```

The Next Language

Programmers are as
emotional and irrational
as normal people.

It took a generation to agree that
high level languages were a good
idea.

It took a generation to agree that
`goto` was a bad idea.

It took a generation to agree that
objects were a good idea.

It took two generations to agree that
lambdas were a good idea.

Systems languages

Application languages

Classical Inheritance

Prototypal Inheritance

Classification Taxonomy

Prototypal Inheritance

- Memory conservation.
- May have made sense in 1995.
- Confusion: Own vs inherited.
- Retroactive heredity.
- Performance inhibiting.

Prototypal Inheritance

Class Free

Block Scope

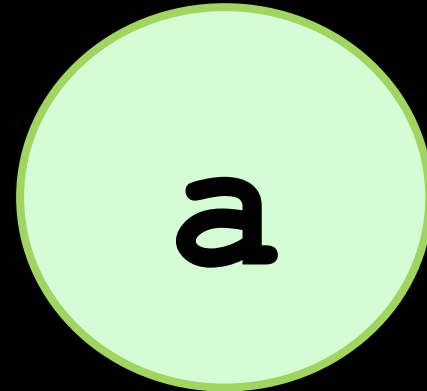
```
{  
  let a;  
  {  
    let b;  
    ... a ...  
    ... b ...  
  }  
  ... a ...  
}
```

Function Scope

```
function green() {  
  let a;  
  function yellow() {  
    let b;  
    ... a ...  
    ... b ...  
  }  
  ... a ...  
}
```

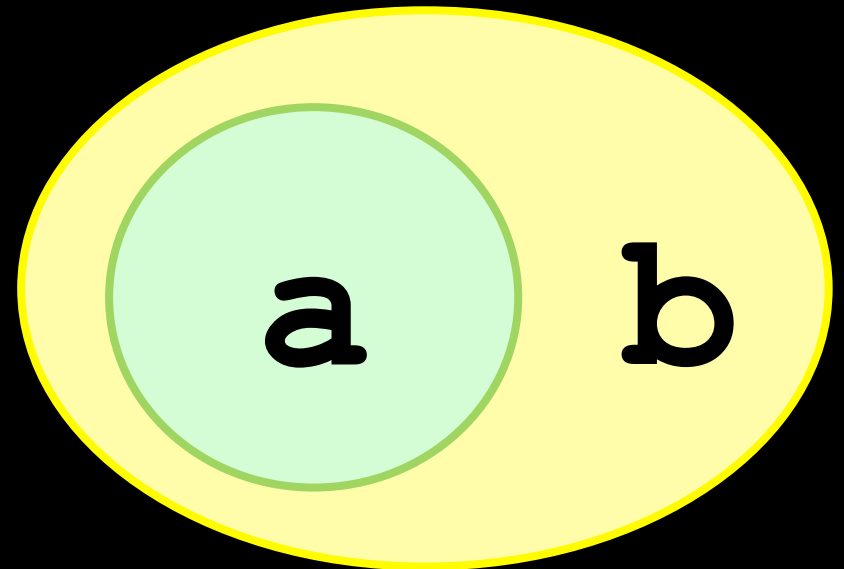
Function Scope

```
function green() {  
  let a;  
  function yellow() {  
    let b;  
    ... a ...  
    ... b ...  
  }  
  ... a ...  
}
```



Function Scope

```
function green() {  
  let a;  
  function yellow() {  
    let b;  
    ... a ...  
    ... b ...  
  }  
  ... a ...  
}
```



Inner survives the outer

```
function green() {  
  let a;  
  return function yellow() {  
    let b;  
    ... a ...  
    ... b ...  
  };  
  ... a ...  
}
```

```
function constructor(spec) {  
    let {member} = spec;
```

```
}
```

```
function constructor(spec) {  
    let {member} = spec,  
        {other} = other_constructor(spec);  
  
}
```

```
function constructor(spec) {  
    let {member} = spec,  
        {other} = other_constructor(spec),  
        method = function () {  
            // member, other, method  
        };  
  
}
```

```
function constructor(spec) {  
    let {member} = spec,  
        {other} = other_constructor(spec),  
        method = function () {  
            // member, other, method  
        };  
  
    return Object.freeze({  
        method,  
        other  
    });  
}
```

A Bug Story

```
private int index;
```

int

- Overflow errors
- Complement: saving gates in subtraction
- Memory used to be really expensive and scarce

Number types

Java: byte char short int long float double

Wastes the programmer's time by having to select the right type.

Errors result from choosing the right type.

No real benefit.

JavaScript: number (same as double)

Having only one type is a huge improvement.

Number types

Java: byte char short int long float double

Wastes the programmer's time by having to select the right type.

Errors result from choosing the right type.

No real benefit.

JavaScript: number (same as double)

Having only one type is a huge improvement.

Unfortunately, it is the wrong type.

0.1 + 0.2 == 0.3

false

Binary Floating Point

Made a lot of sense in the 1950s.

Scientific | Business

DEC64



$$\text{Number} = \text{Coefficient} * 10^{\text{Exponent}}$$

dec64 . com

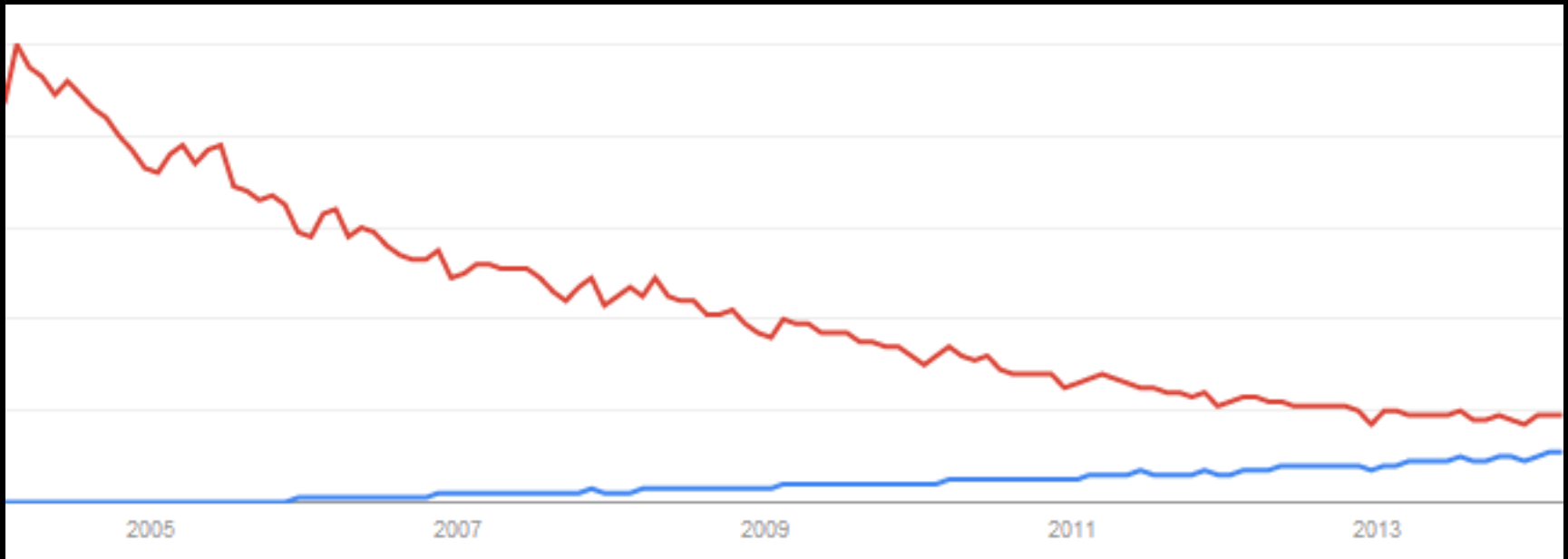


<https://github.com/douglascrockford/DEC64/>

The JSON Data Interchange Format



JSON,XML



Advice to data format standard designers

- Don't break JSON.
- Make it significantly better.
- Get a better name.

JSON

Java-

Script

Object

Notation

Responsibility

1. The People
2. The Team
3. Management

And finally,
one piece of advice:

Don't make bugs.

Thank you and good night.