Jakob Mattsson

Developer • Entrepreneur • Crazy person

@jakobmattsson • jakob.mattsson@gmail.com

on

Forgotten funky functions

Burt.

Client-side JavaScript tracking

Touch&Tell

Quick and easy
customer feedback.

touch-and-tell.se


LEANMACHINE

FISHBRAIN

THE INFORMATION AND OPINIONS CONTAINED IN THESE SLIDES REPRESENT THE VIEWS OF THE AUTHOR AND MAY NOT NECESSARILY REPRESENT THE VIEWS OF THE AUTHOR'S EMPLOYER


JS

Forgotten funky functions

Plan

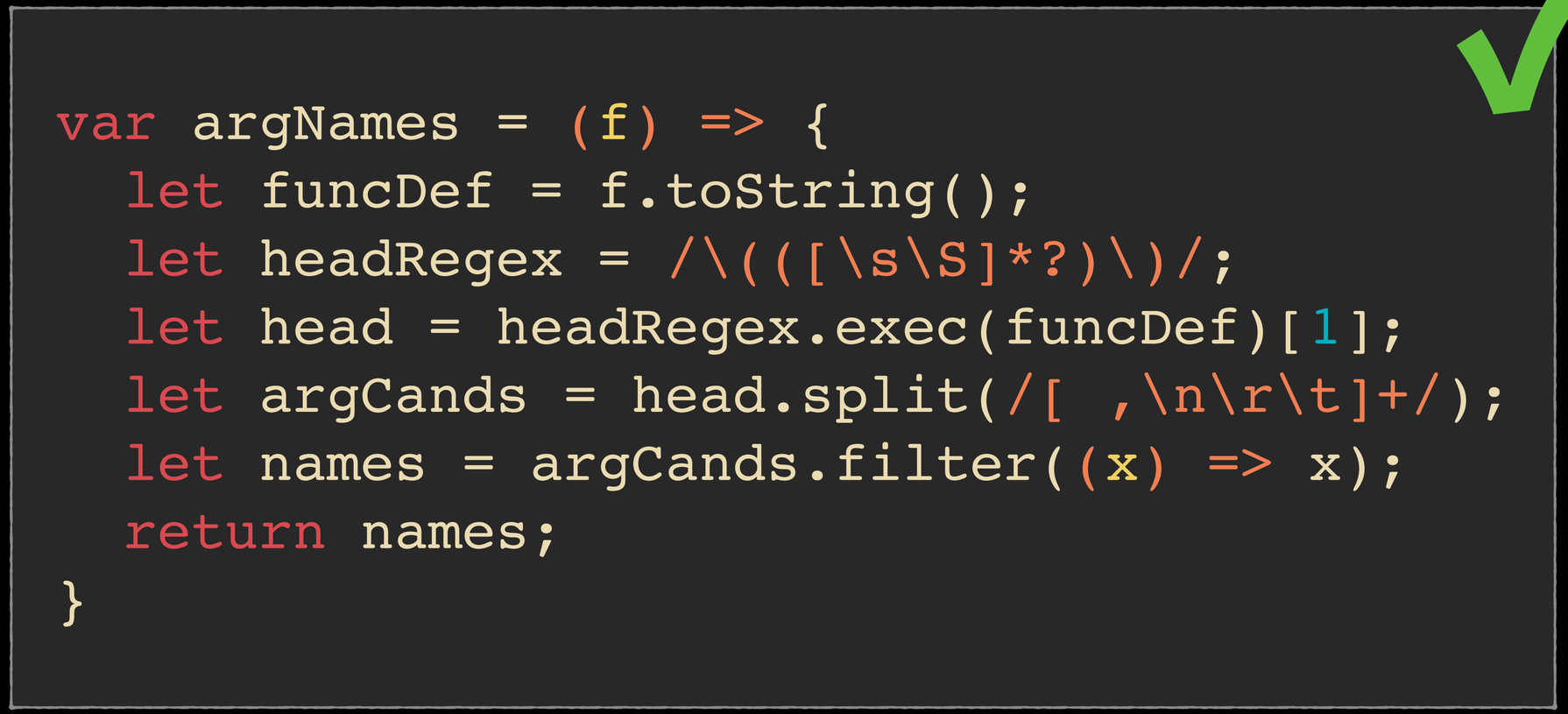
- Example to kick it off
- Four things that makes JavaScript different
- What is real power in a language
- How is ES6 in increasing this power

Examples & fun stuff

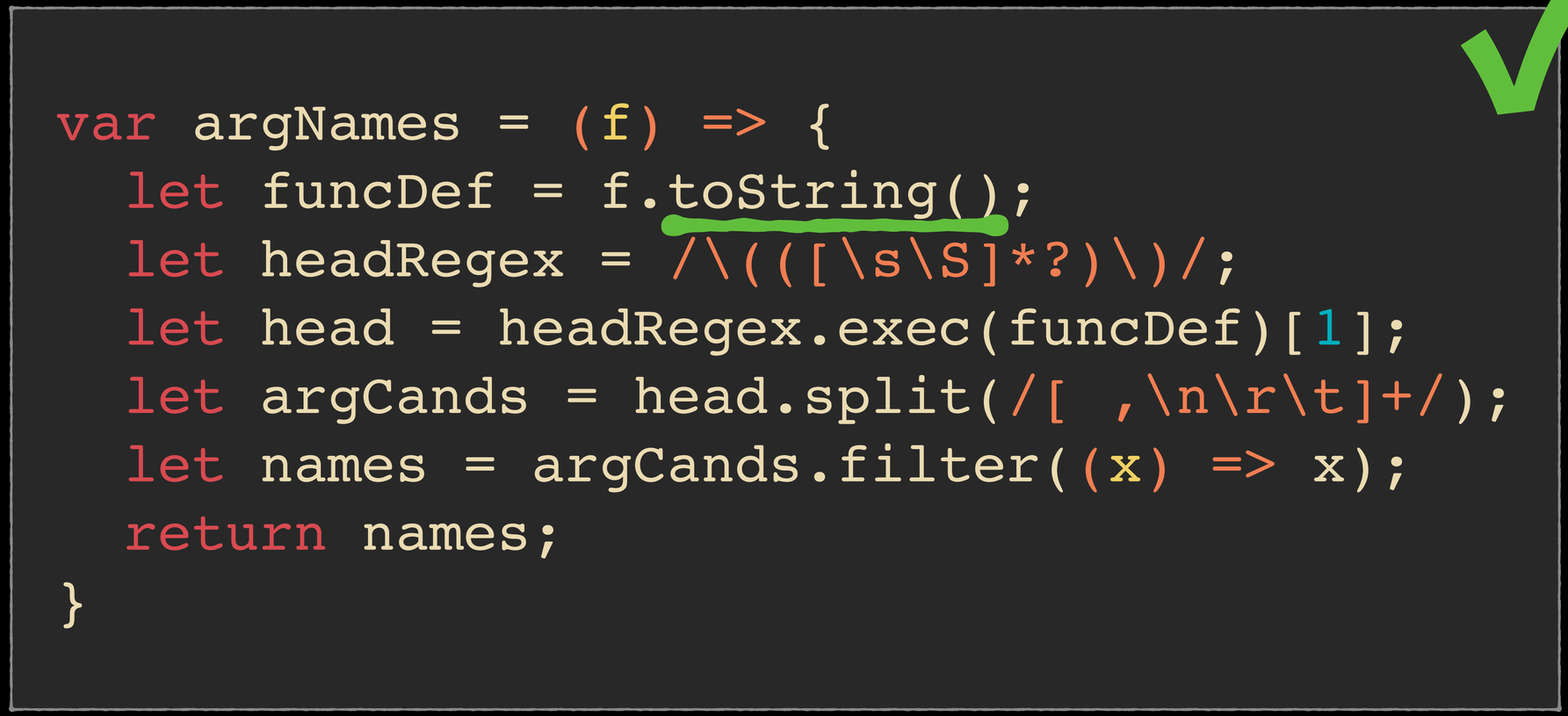
If you have a function,
how do you get the names
of the arguments for it?

If you have a function, how do you get the names of the arguments for it?

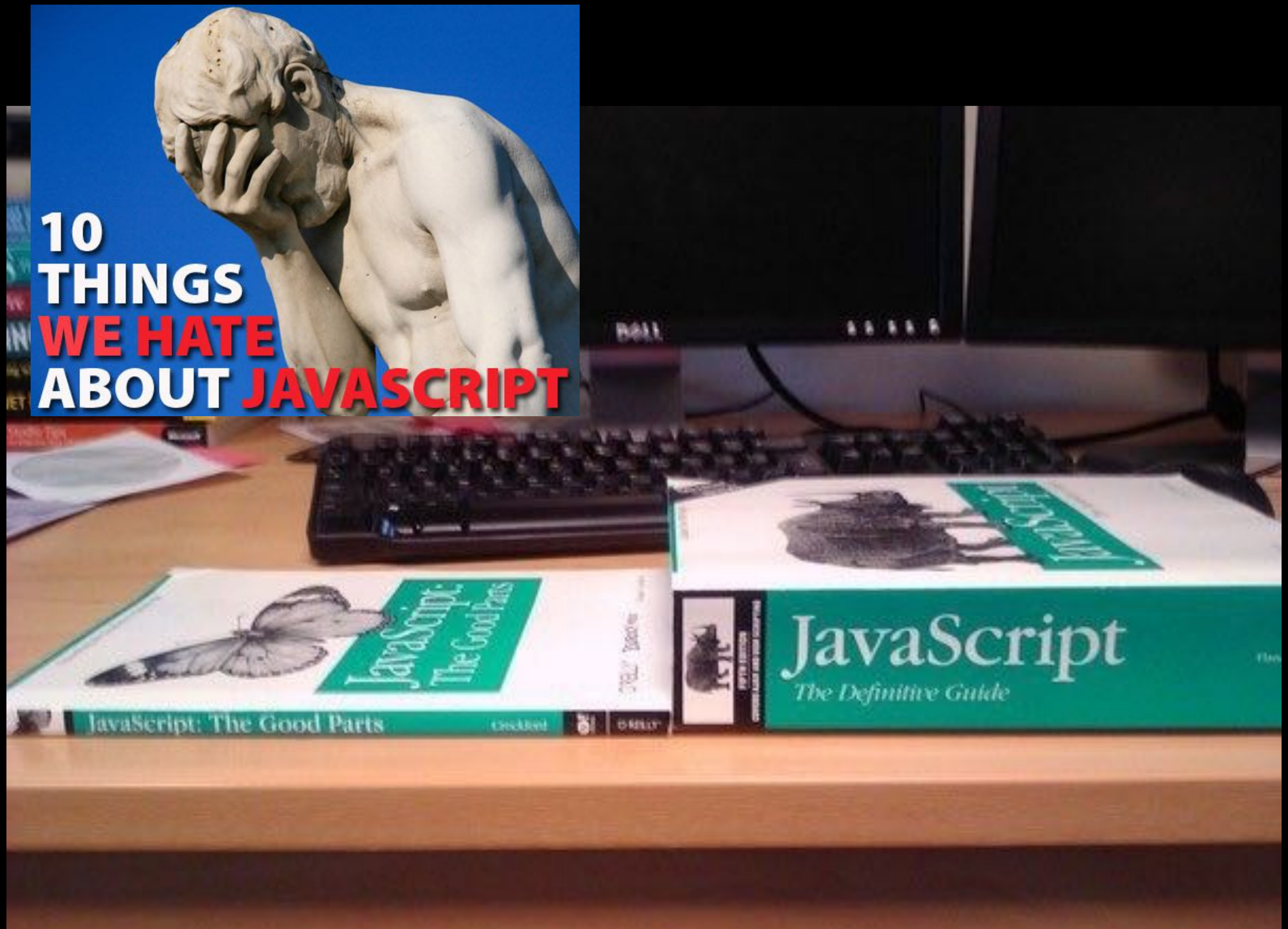
```
var fullName = (firstName, lastName) =>
  firstName + " " + lastName

console.log(argNames(fullName));

// ["firstName", "lastName"]
```



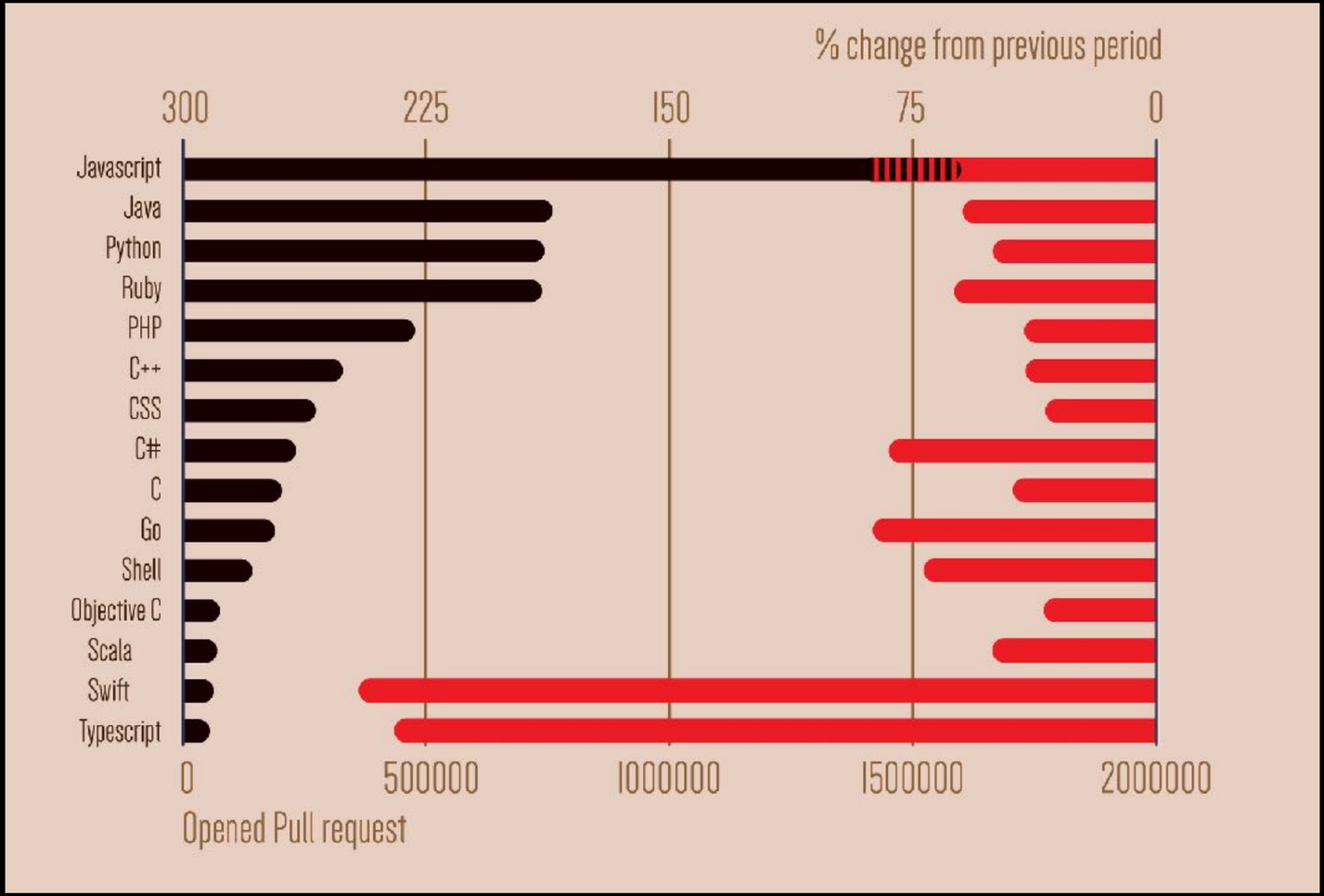
```
var argNames = (f) => {
  let funcDef = f.toString();
  let headRegex = /\(([\s\S]*?)\)/;
  let head = headRegex.exec(funcDef)[1];
  let argCands = head.split(/[ ,\n\r\t]+/);
  let names = argCands.filter((x) => x);
  return names;
}
```



```
var argNames = (f) => {
  let funcDef = f.toString();
  let headRegex = /\(([\s\S]*?)\)/;
  let head = headRegex.exec(funcDef)[1];
  let argCands = head.split(/[ ,\n\r\t]+/);
  let names = argCands.filter((x) => x);
  return names;
}
```


10
THINGS
WE HATE
ABOUT JAVASCRIPT
JavaScript: The Good Parts
JavaScript
The Definitive Guide

Github: the most popular languages



Two conflicting questions

Which language should we use in the browser?

Two conflicting questions

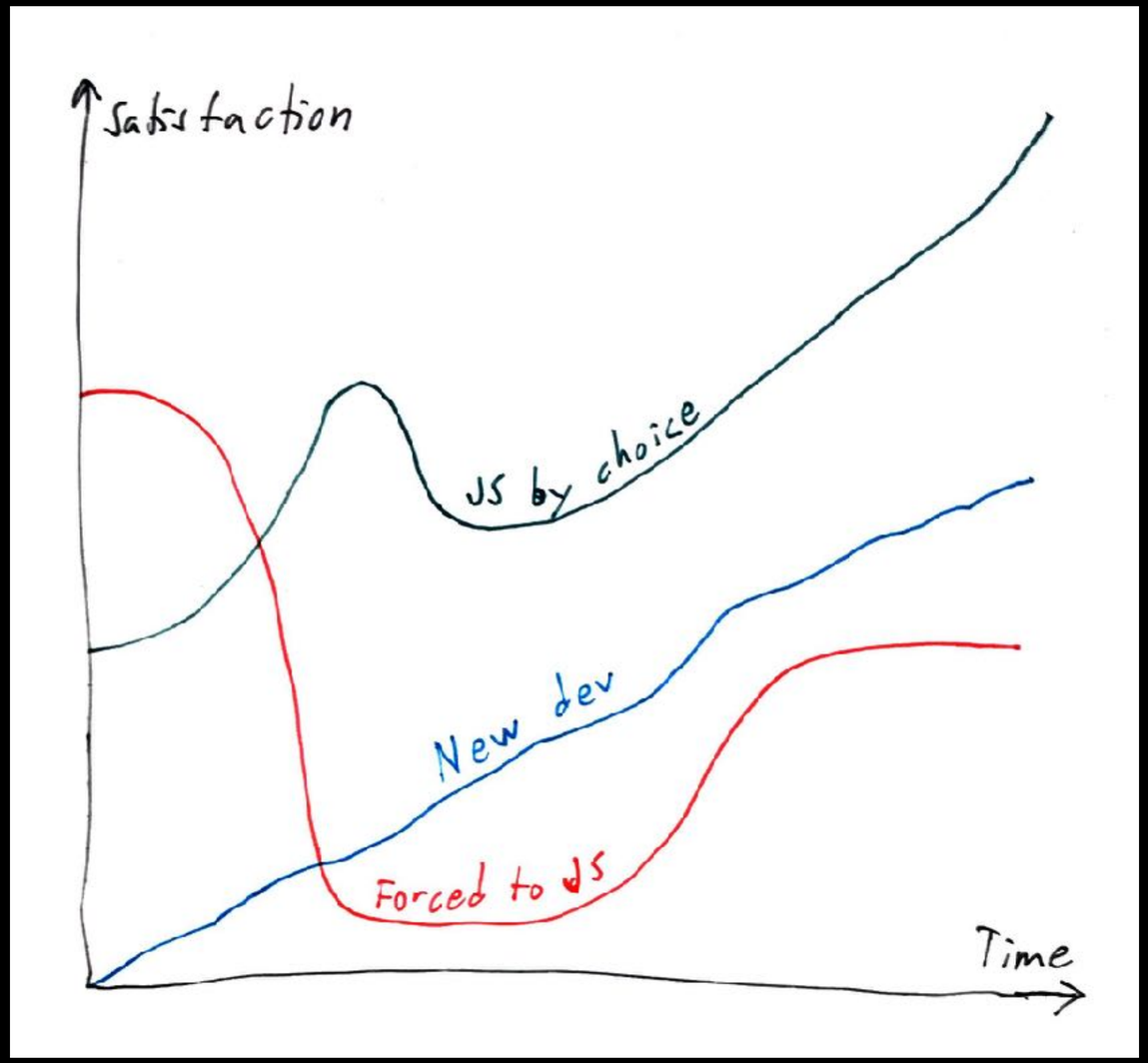
Which language should we use in the browser?

What do we want from this JS language?

Why do we want yet another language ?

Diversity is good!

Being different is good!


Satisfaction
JS by choice
New dev
Forced to JS
Time

Back to the original question:

How is it any different?

Dynamically typed

Dynamically typed

Other languages are too!


R


php

JavaScript

Brendan Eich writes Mocha, a scripting language for Netscape

May 1995

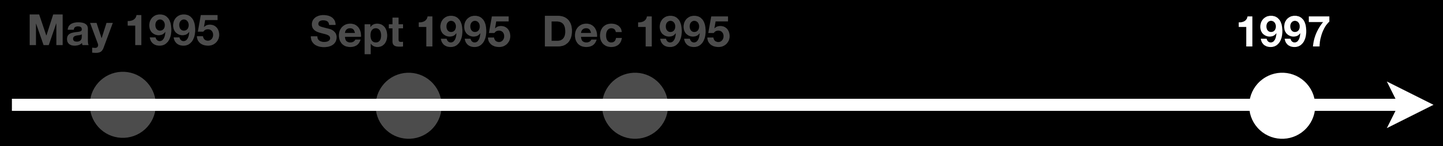
Name changed to LiveScript. Included in Netscape Navigator 2.

May 1995 Sept 1995

Name changed to JavaScript.

May 1995 Sept 1995 **Dec 1995**

Standardization time!
Renamed EMCAScript.



Four cases for scripting

Four cases for scripting

Beautiful inline data: JSON

Four cases for scripting

Beautiful inline data: JSON

First-class functions with closures

”In 1970 Guy Steele and Gerald Sussman created Scheme. Their work lead to a series of lambda papers but still lambdas were relegated to relative obscurity until Java made them popular by not having them.”

- A Brief, Incomplete, and Mostly Wrong History of Programming Languages

”If you have a problem, you just need another function”

- Unknown conference

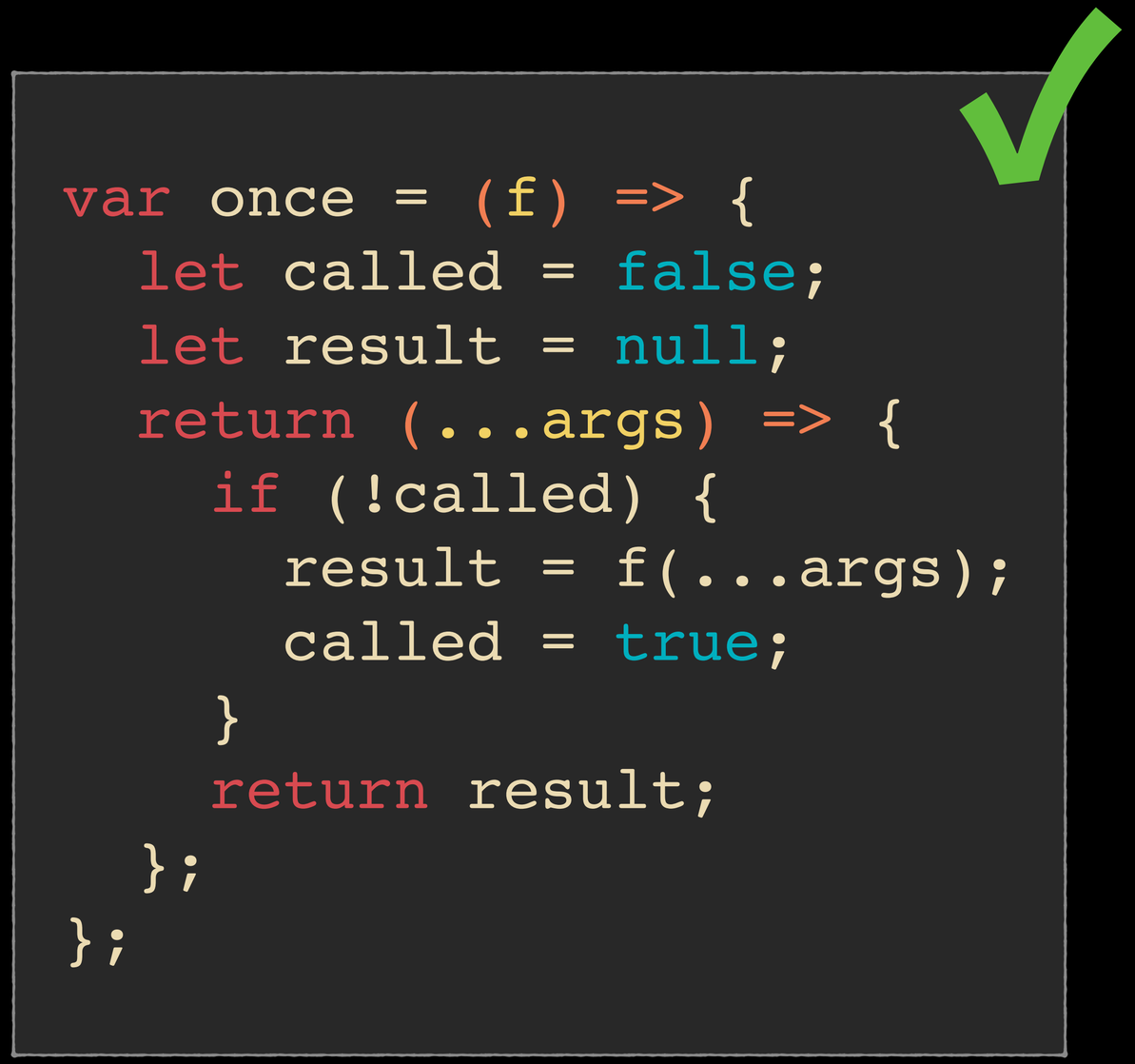
Example:

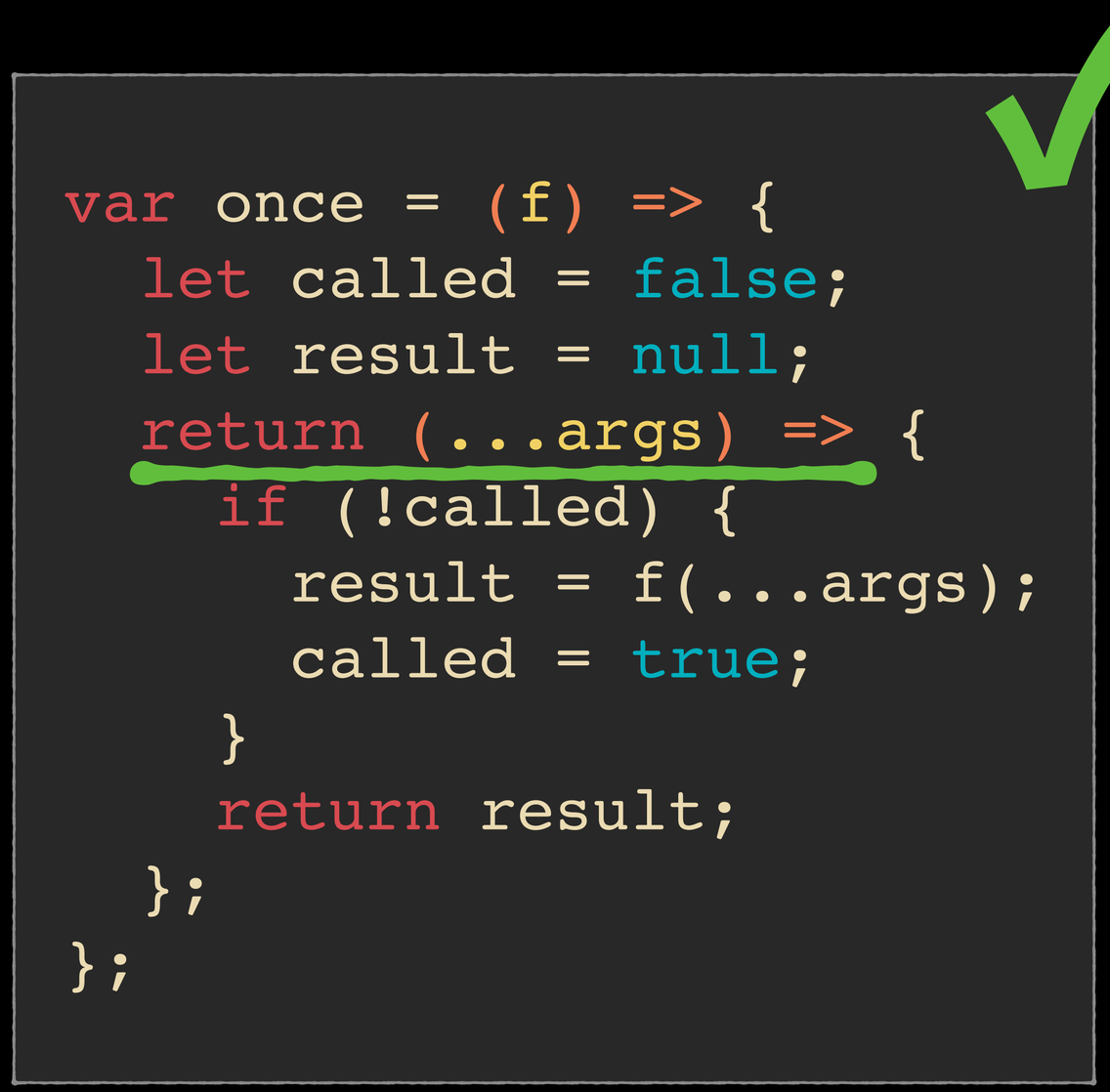
once

Example:

once

```
var initOnce = once(init);

initOnce();
initOnce();
initOnce();
```


var once = (f) => {
let called = false;
let result = null;
return (...args) => {
if (!called) {
result = f(...args);
called = true;
}
return result;
};
};

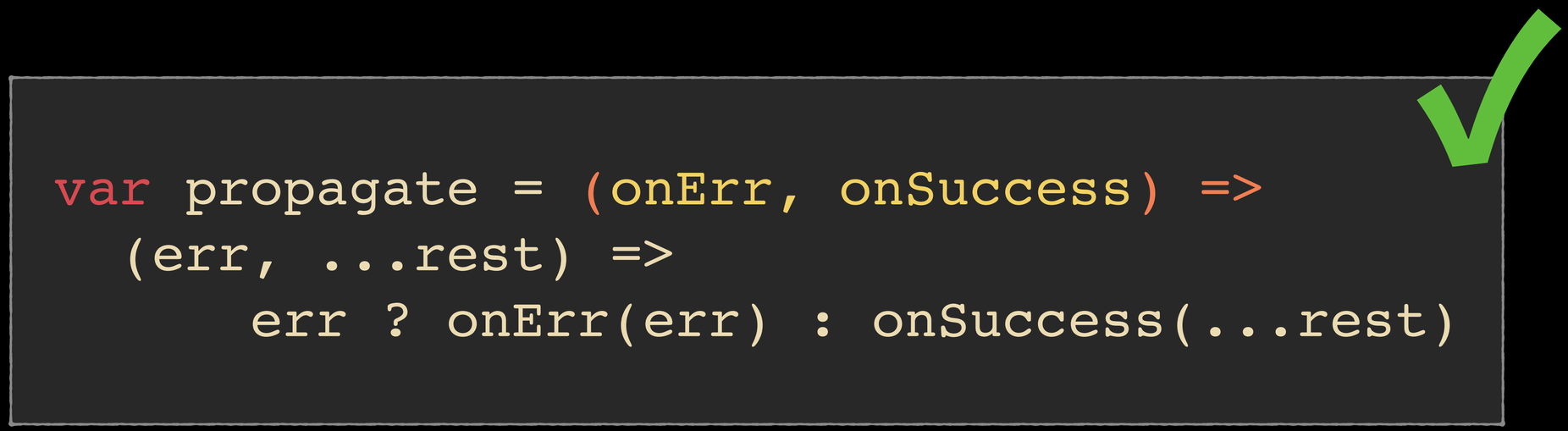

var once = (f) => {
let called = false;
let result = null;
return (...args) => {
if (!called) {
result = f(...args);
called = true;
}
return result;
};
};

Example:

propagate

```
var getUserFirstName = (id, cb) => {
  get('/users/' + id, (err, response) => {
    if (err) {
      cb(err);
      return;
    }
    cb(null, response.firstName);
  });
```

```
var getUserFirstName = (id, cb) => {
  get('/users/' + id, propagate(cb, (response) => {
    cb(null, response.firstName);
  }));
};
```


var propagate = (onErr, onSuccess) =>
(err, ...rest) =>
err ? onErr(err) : onSuccess(...rest)

Don’t let it be magic to you!

Make sure you understand it!

Recommended reading:

paulgraham.com/avg.html

Four cases for scripting

Beautiful inline data: JSON

First-class functions with closures

Async nature

It started out async

setTimeout

xhr

etc…

Four cases for scripting

Beautiful inline data: JSON

First-class functions with closures

Async nature

Built-in inspection and interpreter

JavaScript source access

.toString()

eval

```
var fullName = (firstName, lastName) =>
  firstName + " " + lastName

var fullName2 = argsAsObject(fullName);

fullName2({
  firstName: 'Jakob',
  lastName: 'Mattsson'
});
```

```
var fullName = (firstName, lastName) =>
  firstName + " " + lastName

var fullName2 = argsAsObject(fullName);

fullName2({
  firstName: 'Jakob',
  lastName: 'Mattsson'
});
```

```
var argsAsObject = (f) => {
  let names = argNames(f);
  return (args) => {
    let argValues = names.map((name) => args[name]);
    return f.apply(this, argValues);
  };
};
```

```
var argsAsObject = (f) => {
  let names = argNames(f);
  return (args) => {
    let argValues = names.map((name) => args[name]);
    return f.apply(this, argValues);
  };
};
```

```
var fullName = (firstName, lastName) =>
  firstName + " " + lastName

var fullNameSnakeCase = renameArgs(fullName, [
  'first_name',
  'last_name'
]);

var fullName2 = argsAsObject(fullNameSnakeCase);

fullName2({
  first_name: 'Jakob',
  last_name: 'Mattsson'
});
```

```
var fullName = (firstName, lastName) =>
  firstName + " " + lastName

var fullNameSnakeCase = renameArgs(fullName, [
  'first_name',
  'last_name'
]);

var fullName2 = argsAsObject(fullNameSnakeCase);

fullName2({
  first_name: 'Jakob',
  last_name: 'Mattsson'
});
```

```
var renameArgs = (f, argNames) => {
  var parts = [
    "(function(",
      argNames.join(', ')
    ") { "
      "return f.apply(this, arguments);"
    "});"
  ];
  return eval(parts.join(''));
};
```

Eval can do truly magic things
otherwise impossible

Eval can do truly magic things
otherwise impossible

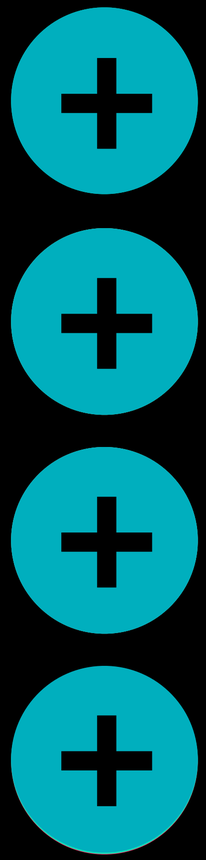
Another one: capability testing

```
var hasArrowFunctions = function() {
  try {
    eval("( () => {} )");
    return true;
  }
  catch (err) {
    return false;
  }
};
```

A little challenge:

Tell me about your eval story,
in the break or over lunch.
Any creative use of it!

Four cases for (java)scripting



Beautiful inline data: JSON

First-class functions with closures

Async nature

Built-in inspection and interpreter


ES6

```
var once = (f) => {
  let called = false;
  let result = null;
  return (...args) => {
    if (!called) {
      result = f(...args);
      called = true;
    }
    return result;
  };
};
```

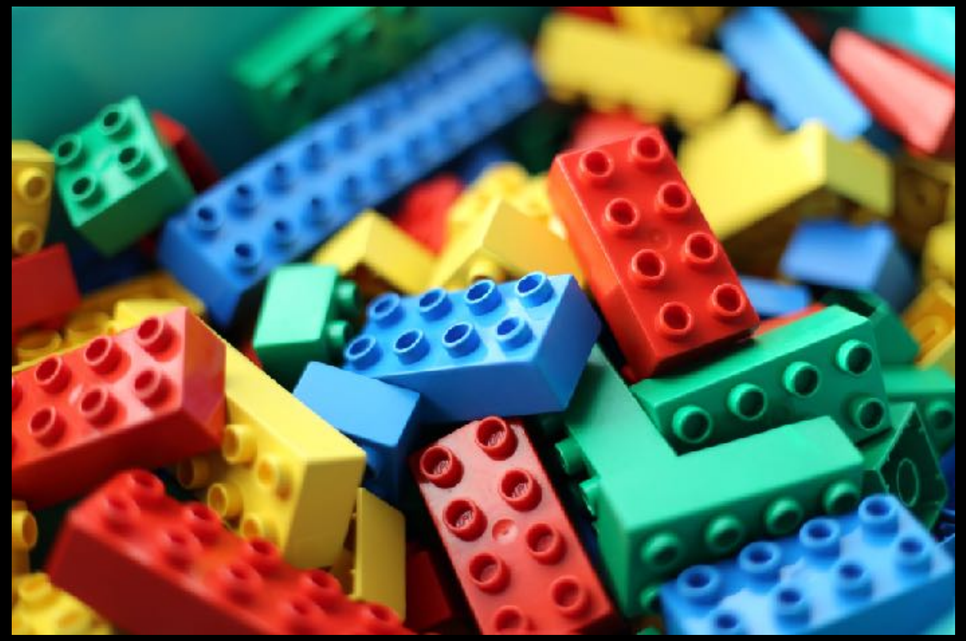
Truly private variables!

More so than traditionally ”private”

Created a stone so heavy even I can’t lift it!

Am I still all powerful?

What is power?



This!

Not this!


ES6

To me it is (primarily) about one thing:

Make the core more powerful!

Too much of this
is distracting:

Proxy

```
var target = {};

var proxy = new Proxy(target, {
  get(target, key) {
    return target[key] * 2;
  }
});

proxy.x = 5

console.log(proxy.x)   // 10
```

```
var target = {};

var proxy = new Proxy(target, {
  get(target, key) {
    return target[key] * 2;
  }
});

proxy.x = 5

console.log(proxy.x)   // 10
```

```
var target = {};

var proxy = new Proxy(target, {
  get(target, key) {
    return target[key] * 2;
  }
});

proxy.x = 5

console.log(proxy.x)   // 10
```

```
var target = {};

var proxy = new Proxy(target, {
  get(target, key) {
    return target[key] * 2;
  }
});

proxy.x = 5

console.log(proxy.x)    // 10
```

Other handlers

- `defineProperty(target, propKey, propDesc)`
- `deleteProperty(target, propKey)`
- `get(target, propKey, receiver)`
- `getOwnPropertyDescriptor(target, propKey)`
- `getPrototypeOf(target)`
- `has(target, propKey)`
- `isExtensible(target)`
- `ownKeys(target)`
- `preventExtensions(target)`
- `set(target, propKey, value, receiver)`
- `setPrototypeOf(target, proto)`

- `apply(target, thisArgument, argumentsList)`
- `construct(target, argumentsList, newTarget)`

1

Negative index

```
var arr = ['foo', 'bar', 'baz'];
console.log(arr[-1]); // baz
```

```
var handlers = {
  get(tgt, key, ctx) {
    const index = Number(key);
    if (index < 0)  {
      return ctx[ctx.length + index];
    }
    return undefined;
  }
}
var oldP = Object.getPrototypeOf(Array.prototype)
var proxy = new Proxy(oldP, handlers);
Object.setPrototypeOf(Array.prototype, proxy)
```

```
var handlers = {
  get(tgt, key, ctx) {
    const index = Number(key);
    if (index < 0)  {
      return ctx[ctx.length + index];
    }
    return undefined;
  }
}
var oldP = Object.getPrototypeOf(Array.prototype)
var proxy = new Proxy(oldP, handlers);
Object.setPrototypeOf(Array.prototype, proxy)
```

```
var handlers = {
  get(tgt, key, ctx) {
    const index = Number(key);
    if (index < 0)  {
      return ctx[ctx.length + index];
    }
    return undefined;
  }
}
var oldP = Object.getPrototypeOf(Array.prototype)
var proxy = new Proxy(oldP, handlers);
Object.setPrototypeOf(Array.prototype, proxy)
```

2

No such method

```
var x = strictObject({
  name: "jakob",
  foo() { console.log(”name: ", this.name) }
});

x.name = "saint petersburg";
x.foo();
x.nane = "holyjs";  // Error: No property with the name nane
x.fooo();           // Error: No property with the name fooo
```

```
var x = strictObject({
  name: "jakob",
  foo() { console.log(”name: ", this.name) }
});

x.name = "saint petersburg";
x.foo();
x.nane = "holyjs";  // Error: No property with the name nane
x.fooo();           // Error: No property with the name fooo
```

```
var strictObject = (obj) => {
  return new Proxy(obj, {
    get(tgt, key, ctx) {
      if (Reflect.has(tgt, key))
        return Reflect.get(tgt, key, ctx);
      throw `No property with the name ${key}`;
    },
    set(tgt, key, val, ctx) {
      if (Reflect.has(tgt, key))
        return Reflect.set(tgt, key, val, ctx);
      throw `No property with the name ${key}`;
    }
  });
};
```

```
var strictObject = (obj) => {
  var pobj = new Proxy({}, {
    get(target, key) {
      throw `No property with the name ${key}`
    },
    set(target, key) {
      throw `No property with the name ${key}`
    }
  });
  Object.setPrototypeOf(obj, pobj);
  return obj;
};
```

Remember:

It is impossible to determine whether an object is a proxy or not
(aka transparent virtualization)

You can’t access a handler via its proxy
(aka handler encapsulation)

3

Proxy vs non-proxy

```
const proxies = new WeakSet();

export function createProxy(obj, handler) {
    const proxy = new Proxy(obj, handler);
    proxies.add(proxy);
    return proxy;
}

export function isProxy(obj) {
    return proxies.has(obj);
}
```

```
const proxies = new WeakSet();

export function createProxy(obj, handler) {
    const proxy = new Proxy(obj, handler);
    proxies.add(proxy);
    return proxy;
}

export function isProxy(obj) {
    return proxies.has(obj);
}
```

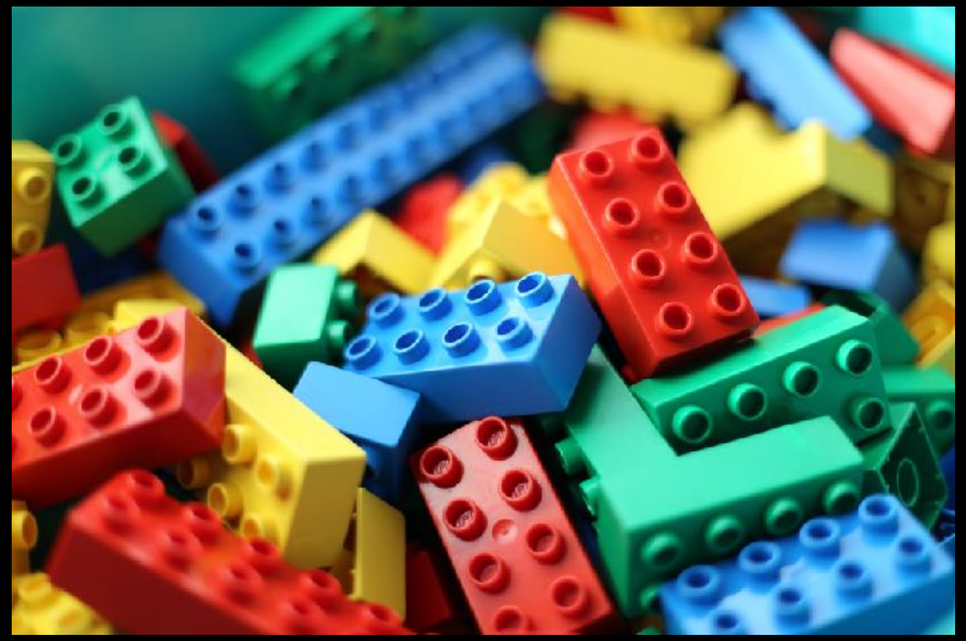
```
const proxies = new WeakSet();

export function createProxy(obj, handler) {
    const proxy = new Proxy(obj, handler);
    proxies.add(proxy);
    return proxy;
}

export function isProxy(obj) {
    return proxies.has(obj);
}
```


TIME FOR
ACTION
0:53 47
0:27 08

Remember
three principles

1. Love JS because of its own strengths

2. Powerful generic building blocks



This!

Not this!

3. Keep it simple. Don’t do everything.

Remember three practices

Remember three practices

1. **Introspection:**

   Reading code from code is powerful and eval can do otherwise impossible things.

Remember three practices

1. **Introspection:**

   Reading code from code is powerful and eval can do otherwise impossible things.

2. **Functional:**

   Functions for creating functions is a source of amazingness in JavaScript

Remember three practices

1. **Introspection:**

   Reading code from code is powerful and eval can do otherwise impossible things.

2. **Functional:**

   Functions for creating functions is a source of amazingness in JavaScript

3. **Prototypes and proxies:**

   Use prototypes and proxies to take objects to the next level

Jakob Mattsson

Developer • Entrepreneur • Crazy person

@jakobmattsson • jakob.mattsson@gmail.com

Q&A