BUSINESS PLUS
Solution Partner
SPECIALISED Adobe Experience Manager

netcentric

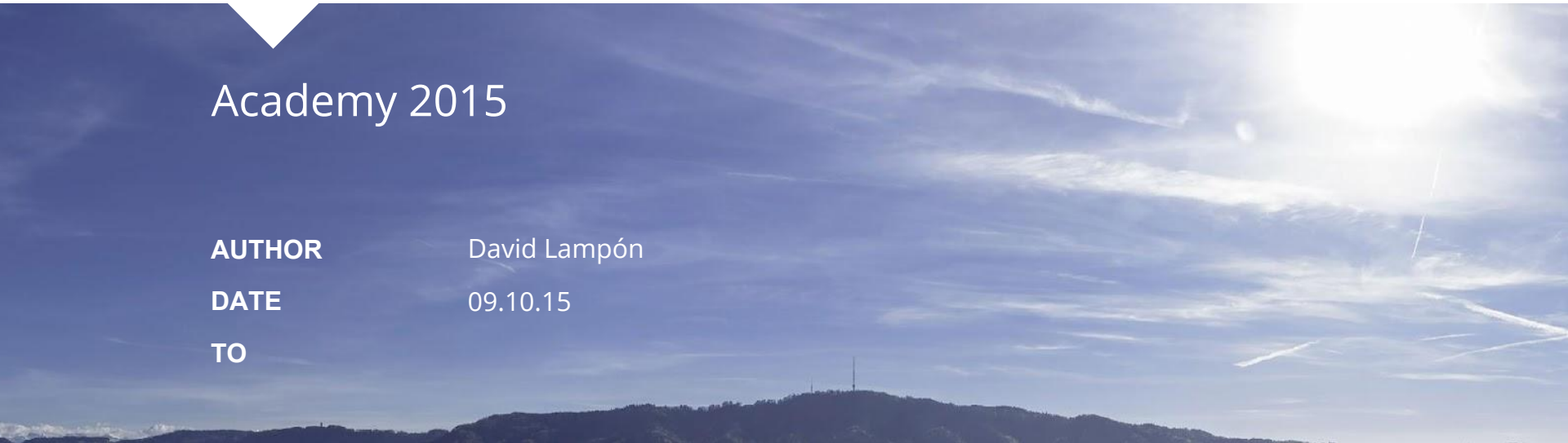
ES6

Academy 2015

AUTHOR David Lampón

DATE 09.10.15

TO



Agenda

01 **Welcome**

02 **History of JavaScript**

03 **What is all this expectation around ES6?**

04 **How can we use ES6 today? Babel rules!**

05 **How do we code ES5 today? The good parts**

06 **ES6 - warm up -**

07 **Variables and parameters + exercises**

08 **A little bit of functions + exercises**

09 **Modules + live code demo**

10 **Summary**

1

WELCOME

About the presenter…

David Lampon Diestre

- 36 years old --- 1979 best generation ever ---

Background:

- Telecommunication Engineering - UPC - 2001
- Architecture - La Salle - 2011

Previous working experience:

- Webmaster @NTT/Verio 2001
- PHP developer @DelClos Consultors 2002-2003
- Part-time freelancer web designer 2004-2006
- iOS developer @ThreeBytes 2012

@Netcentric (since January 2014)

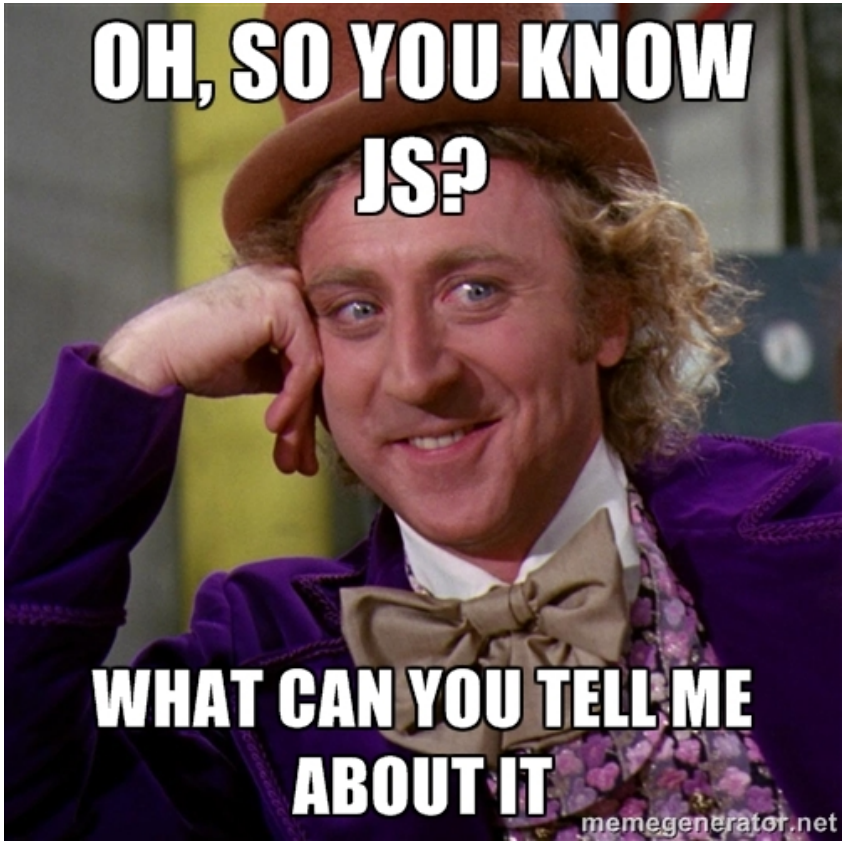
• Migration project for UBS Intranet	Jan ‘14 - Aug ‘14	CQ 5.6
• BMW Motorrad	Aug ‘14 - Aug ‘15	AEM 6.0
• Daimler Mercedes	Aug ‘15	AEM 6.1

Apart from that…

- Happily married
- Eric is coming (expected release on 30th of October)
- Currently crossfitter and road biker
- Current PS4 gamer (wide previous experience)
- Best movie ever: Shawnshack Redemption, Great Escape and Back to the Future
- Best band ever: Dave Matthews Band

http://david.lampon.net - @davidlampon - david.lampon@netcentric.biz

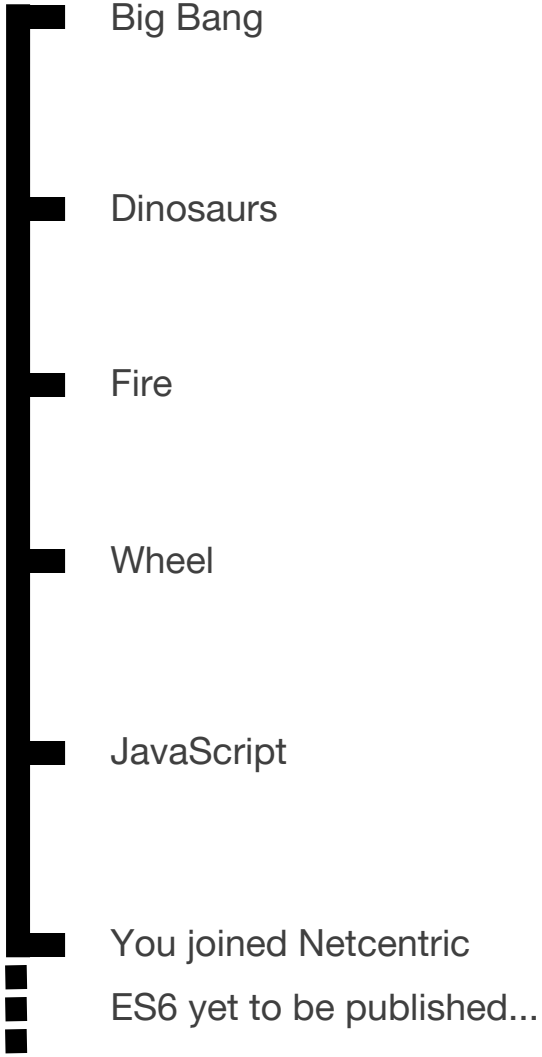
Before we start...



2

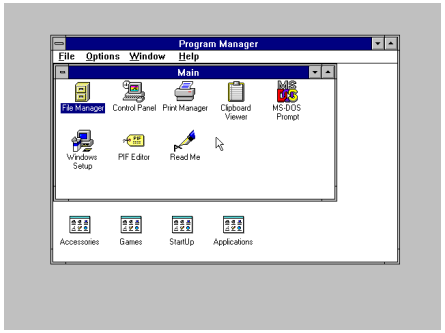
HISTORY OF JAVASCRIPT

History of JavaScript



History of JavaScript

v1.0 - 1981 => v7.0 - 1994



1992 - 2001

1995

Netscape announced in its first press release (13 October 1994) that it would make Navigator available without charge to all non-commercial users.

Wikipedia

In October 1997, Sun Microsystems, the creator of Java, sued Microsoft for incompletely implementing the Java 1.1 standard.

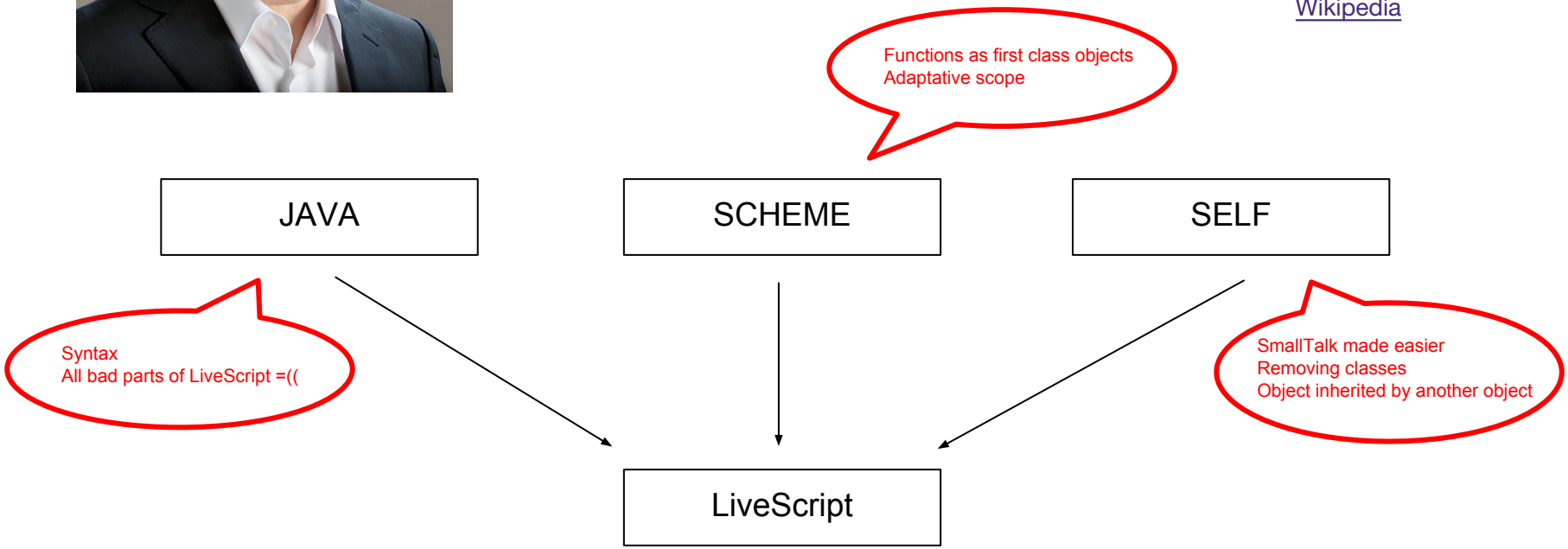
Wikipedia

History of JavaScript

Brendan Eich

The first version was completed in ten days in order to accommodate the Navigator 2.0 Beta release schedule, and was called Mocha, which was later renamed LiveScript in September 1995 and later JavaScript in the same month.

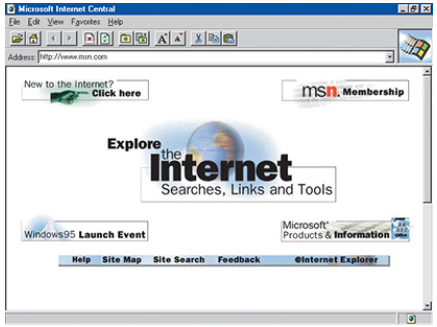
Wikipedia



History of JavaScript

- JAVA as a language was taking off at that time as we were writing directly to the VM instead of the OS so we could **escape from the control of Microsoft**. Sun represented the freedom for the developers who could program independently of the computer as long as it was able to run a Java Virtual Machine.
- **Sun and Netscape** as the Internet started to shape up become two main players in the IT war as they both allied could become an alternative to Microsoft or if they didn't play well their cards they could end up competing against each other.
- Sun had **two request** to sign up an alliance with Netscape:
  - Put Java language in the browser
  - Remove LiveScript form the browser (as Java intended to be the master language)
- First condition was ok. Second was ko:
  - Java could not talk to the DOM
  - They wanted a language for beginners (definitely not Java)
- As a joke on a meeting they proposed to change the name **from LiveScript to Javascript**: "interpreted Java", "Java's little brother"...

History of JavaScript



- At that time Microsoft was not paying attention to the Internet. They thought that the future would be the fax. When Netscape started to get hot they bought SpyGlass which after some refactoring became **Internet Explorer 1.0**.
- They crafted their own script language: **DBScript** with a perfect exercise of **reverse engineering**: they copied every flaw on the language even knowing they were flaws. So both were kind of compatible although they were inside different browsers.
- Sun took ownership of the **JavaScript** trademark and Microsoft did the same with **ECMAScript** (probably the worst name ever as it was written in the cover page of a draft document).
- They both tried to **standardize their language**. Microsoft went to the W3C who wanted to send Sun to hell and so they did. Netscape tried the same with ISO and other European standardization institution.
- When ECMAScript became standardised none of the bugs Brendan had detected were fixed.

WHAT IS ALL THIS EXPECTATION AROUND ES6?

Expectation around ES6

Edition	Date published	Changes from prior edition	Editor
1	June 1997	First edition	Guy L. Steele, Jr.
2	June 1998	Editorial changes to keep the specification fully aligned with ISO/IEC 16262 international standard	Mike Cowlishaw
3	December 1999	Added regular expressions, better string handling, new control statements, try/catch exception handling, tighter definition of errors, formatting for numeric output and other enhancements	Mike Cowlishaw
4	Abandoned	Fourth Edition was abandoned, due to political differences concerning language complexity. Many features proposed for the Fourth Edition have been completely dropped; some are proposed for ECMAScript Harmony.	
5	December 2009	Adds "strict mode", a subset intended to provide more thorough error checking and avoid error-prone constructs. Clarifies many ambiguities in the 3rd edition specification, and accommodates behaviour of real-world implementations that differed consistently from that specification. Adds some new features, such as getters and setters, library support for JSON, and more complete reflection on object properties.[8]	Pratap Lakshman, Allen Wirfs-Brock
5.1	June 2011	This edition 5.1 of the ECMAScript Standard is fully aligned with third edition of the international standard ISO/IEC 16262:2011.	Pratap Lakshman, Allen Wirfs-Brock
6	June 2015[9]	The Sixth Edition adds significant new syntax for writing complex applications, including classes and modules, but defines them semantically in the same terms as ECMAScript 5 strict mode. Other new features include iterators and `for/of` loops, Python-style generators and generator expressions, arrow functions, binary data, collections (maps, sets and weak maps), and proxies (metaprogramming for virtual objects and wrappers). As the first "ECMAScript Harmony" specification, it is also known as "ES6 Harmony".	Allen Wirfs-Brock
7	Work in progress	The Seventh Edition is in a very early stage of development, but is intended to continue the themes of language reform, code isolation, control of effects and library/tool enabling from ES6. New features proposed include promises/concurrency, number and math enhancements, guards and trademarks (an alternative to static typing), operator overloading, value types (first-class number-like objects), new record structures (records, tuples and typed arrays), pattern matching, and traits.[10]	

Wikipedia

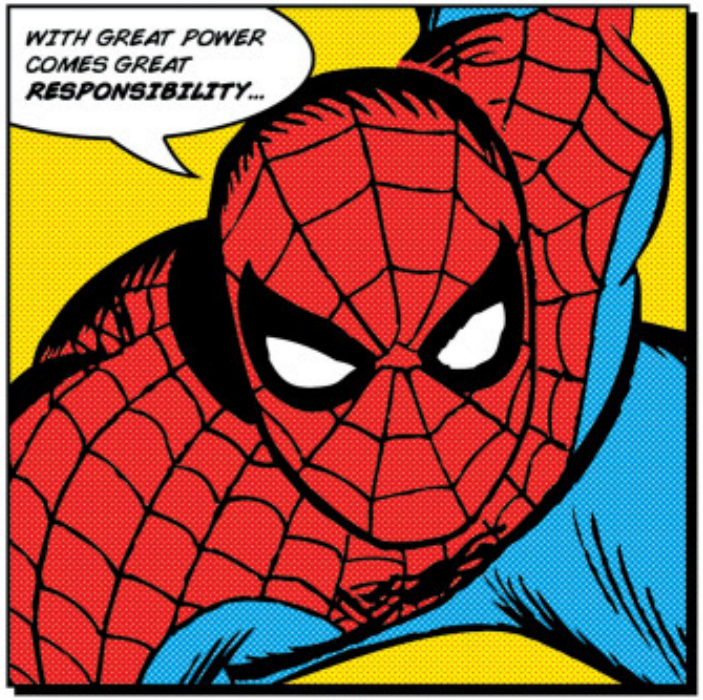
- 1999 ES3 (third edition), 2009 ES5 (fifth edition) - TC39 - A lot of time without big changes, it **became more stable** (which is the most important) despite it's bad parts.

Expectation around ES6

- Biggest change of the language since its release
- Additions to make its syntax more clear
- Capabilities to improve its performance on the front and back end sides
- More mature and robust as a language
- More easy to jump in for people who come from other languages
- A big change of coding style but maintaining the legacy
- ...

List of new features

Expectation around ES6



A big change is coming…

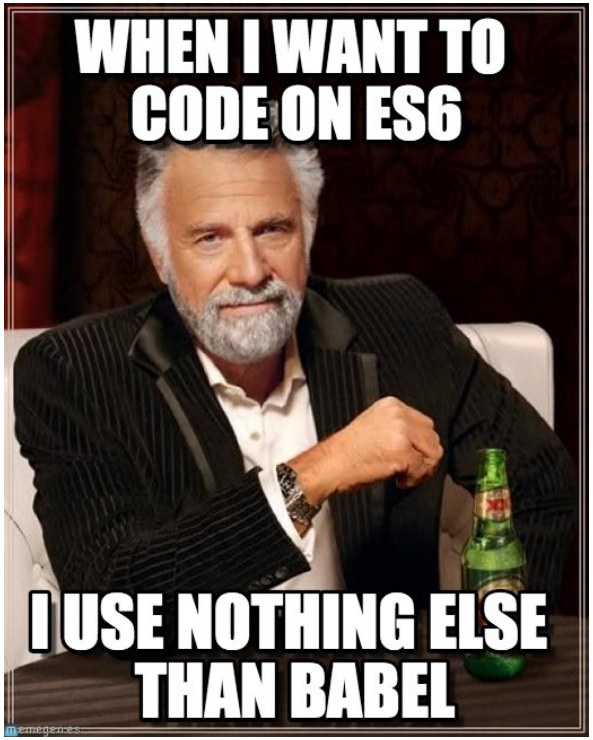
- Good for the **language** to be more mature, accessible but stronger and richer...
- Good for the **IT sector** to move forward and have better tools to develop...
- Good for **frontend engineers** to be able to develop even more awesome products...
- Good for **newbies** because this is sort of a fresh start for the whole industry...

4

HOW CAN WE USE ES6 TODAY? BABEL RULES!

How can we use ES6 today?

Problem: ES6 is not yet published (no, not yet)



Solution: Transpilers

How can we use ES6 today?

Options:

1. Execution environment: browser / node
2. Polyfills / shims: add new functionality to an existing environment
3. Transpiler: translator for a language to another (from ES6 to ES5)
4. And of course any combination of the three above

Currently in production:

- Using ES6 in production is becoming more and more normal
- Angular 2.0 and Ember are being developed in ES6 already

Addy Osmani - @addyosmani

ES6 tools repo

How can we use ES6 today?

Browsers

- All browsers are currently working hard on implementing new ES6 features on their JS engines as fast as possible.
- Browsers are ok to test, try and experiment but are not suitable for commercial purposes as you rely on your clients to have installed a specific model and version of their browser.
- It will change overtime but if we are still supporting IE9 (2011) we can't really wait for browsers and community to adopt new browsers to start develop on ES6. Remember the flexbox drama.

Chrome release channels
(chrome://flags)

Firefox nightly builds

IE / Safari...

How can we use ES6 today?

Polyfills / Shims

- JS libraries to enable new capabilities
- They close the technology gap with newer and older browsers
- If the browser already has the feature, it will use the native one
- The least disruptive method
- The whole ES6 can't be implemented in this way

Repos

- Paul Miller - ES6 Shims
- WebReflection/es6-collection
- ECMAScript Shims

- ES6 - arrays extras (raw file)

How can we use ES6 today?

Transpilers

Traceur:

- Google project to enable developers to test ES6 new features and provide some feedback for the TC39 group
- Speed is secondary, the most important aspect is stability
- Not 100% of ES6 is implemented in Traceur as completely new features can't be coded in ES5.

How to:

- Traceur REPL (read - evaluate / print / loop)
- Plunker (remember to activate traceur on the libraries tab)

  * We can include external js files including *type="module"* in the <script> tag
  * All transpilation takes place in runtime in the browser: not good for production
  * For experimental features include: *<script>traceur.options.experimental = true;</script>*

- NPM package + gulp task

How can we use ES6 today?

Transpilers

ESNext:

- Similar to Traceur
- Supports a small set of features / syntax but doesn't require runtime library to be included in the code
- NPM package + gulp task

BABEL(website):

- The default ES6 transpiler for the community
- Transpiles ES6, some ES7 and some other *crazy* features
- NPM package + gulp task

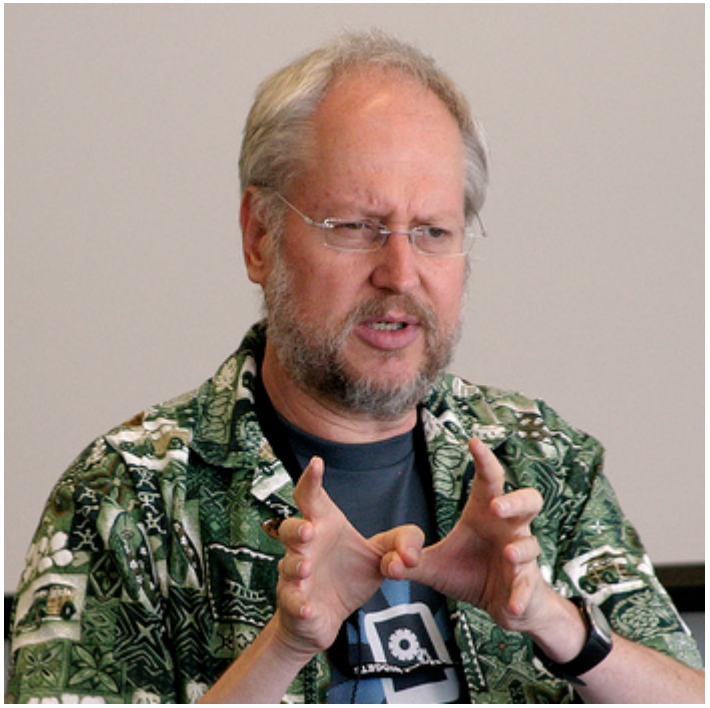
NODE.JS:

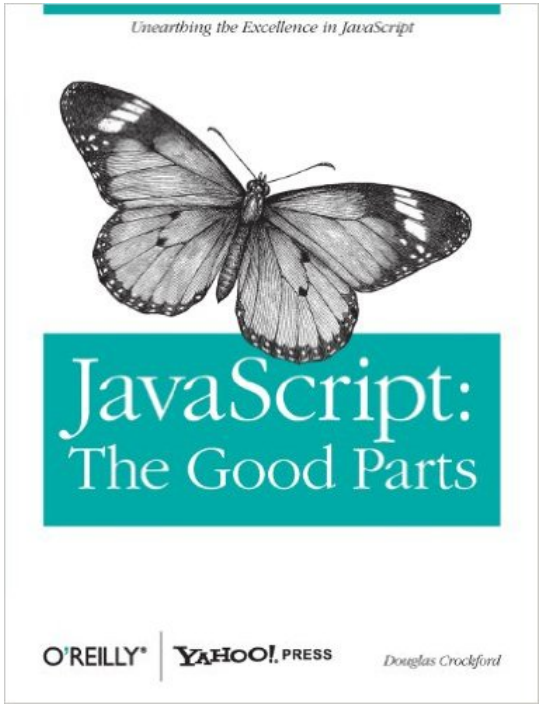
- 0.10 doesn't support any ES6
- ES6 support starts at 0.11
- What is supported in Node?

HOW DO WE CODE ES5 TODAY? THE GOOD PARTS

How do we code ES5 today?

Exercises *stolen* from Douglas Crockford and his master class about **JavaScript, The Good parts**





How do we code ES5 today?

```
function funky(o) {
    o = null;
}
var x = [];
funky(x);
alert(x);
```

A. null
B. []
C. undefined
D. throw

What is x?

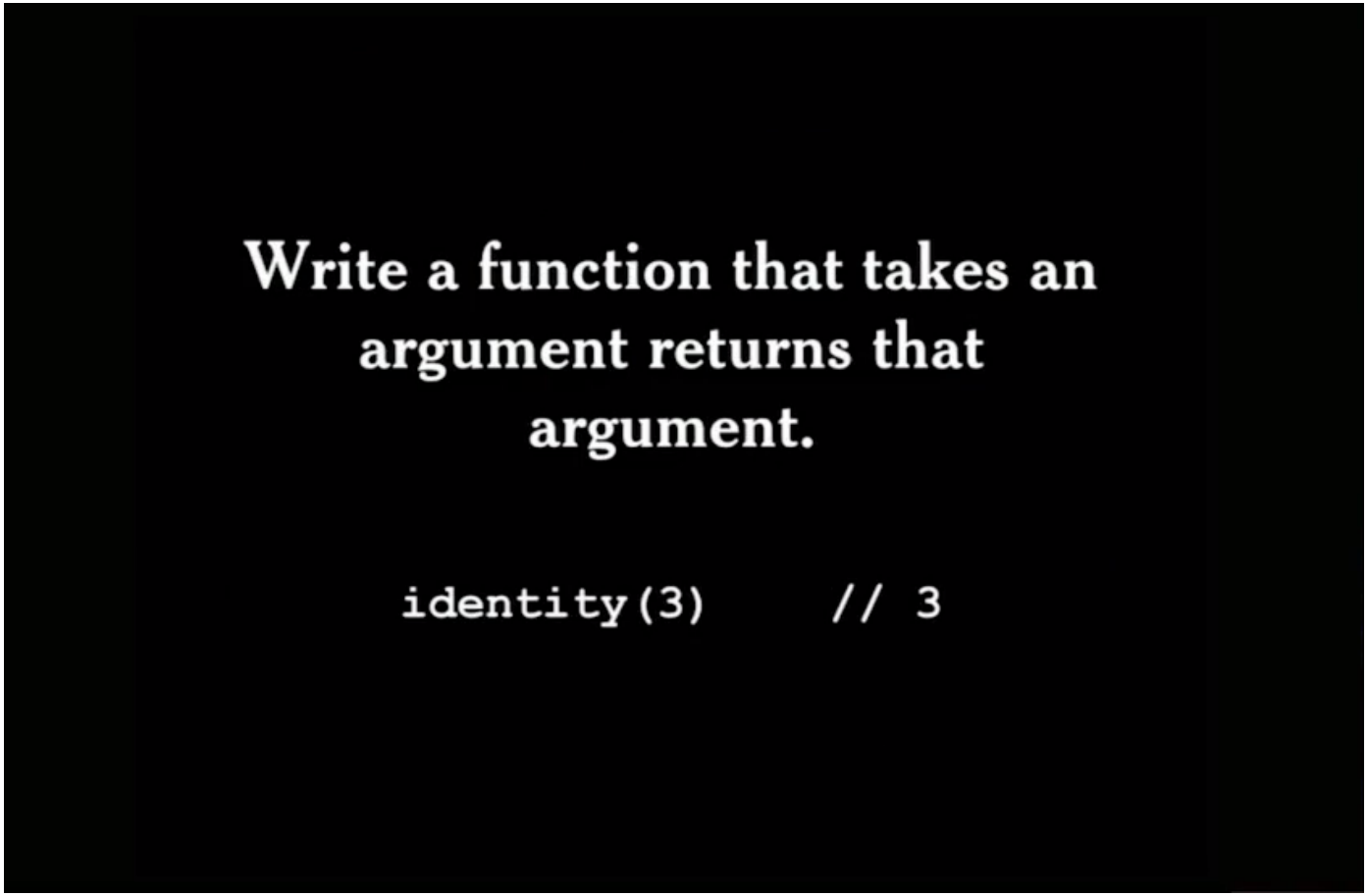
How do we code ES5 today?

```
function swap(a, b) {
    var temp = a;
    a = b;
    b = temp
}
var x = 1, y = 2;
swap(x, y);
alert(x);
```

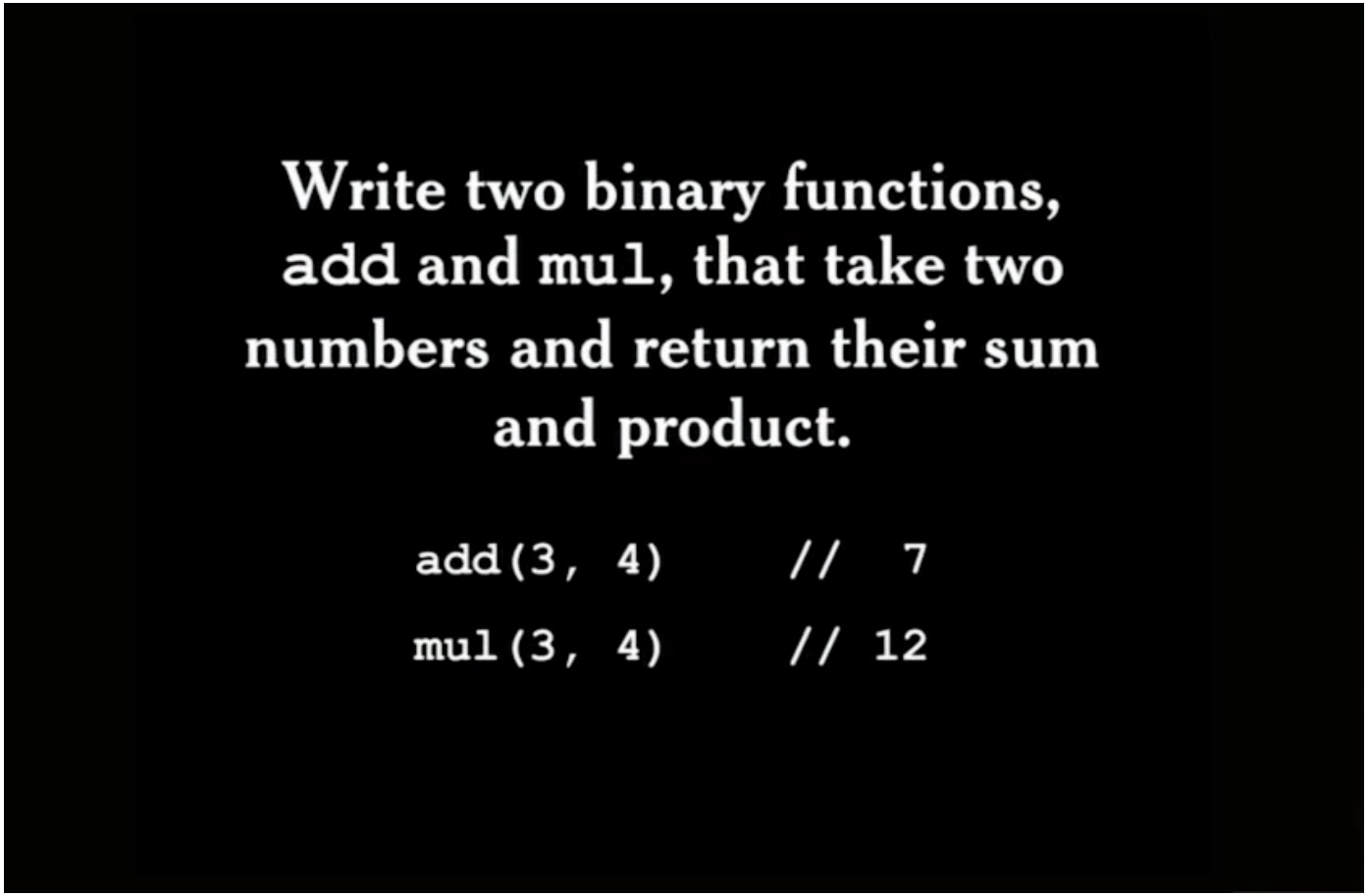
A. 1
B. 2
C. undefined
D. throw

What is x?

How do we code ES5 today?



How do we code ES5 today?

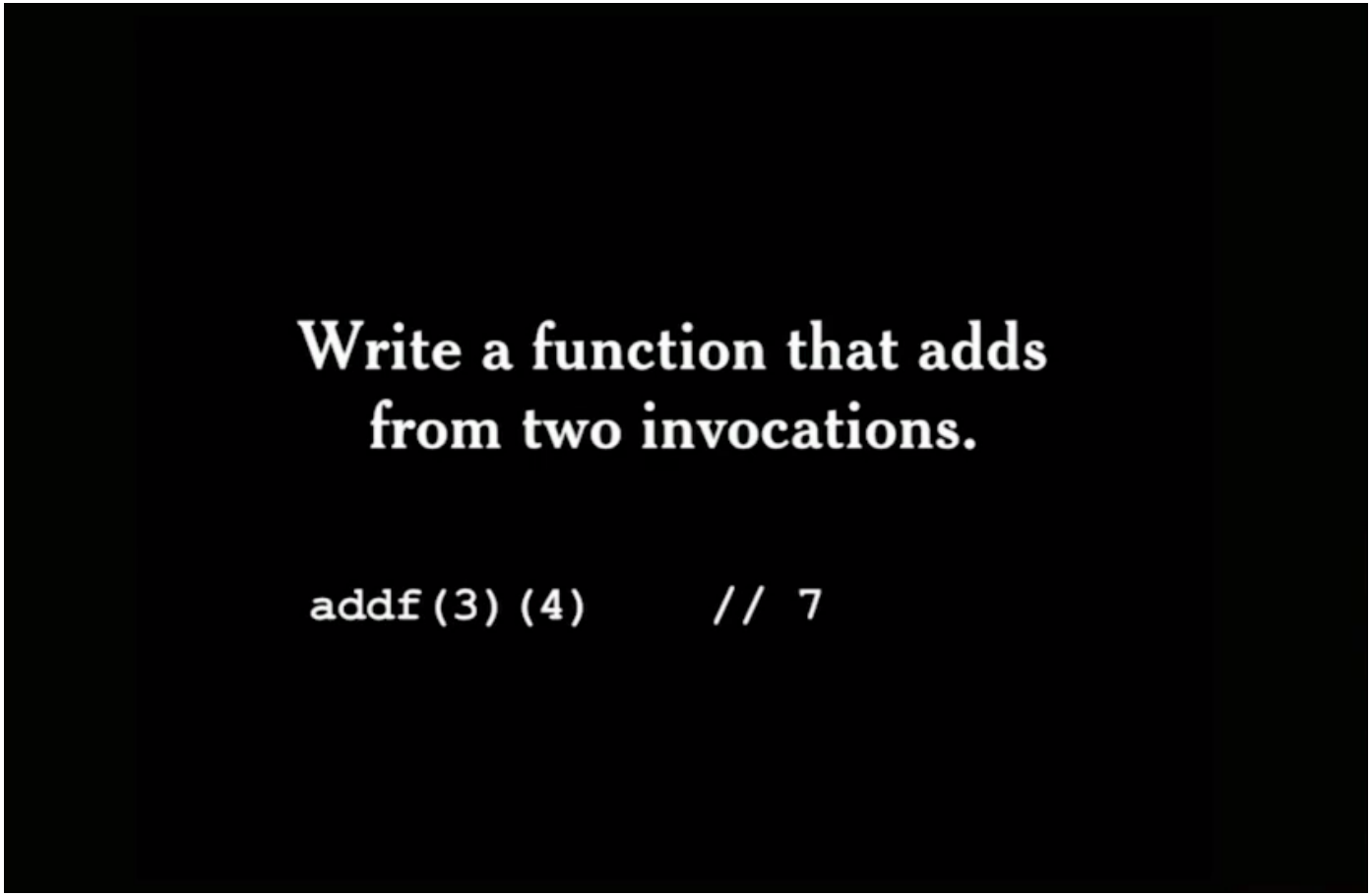


How do we code ES5 today?

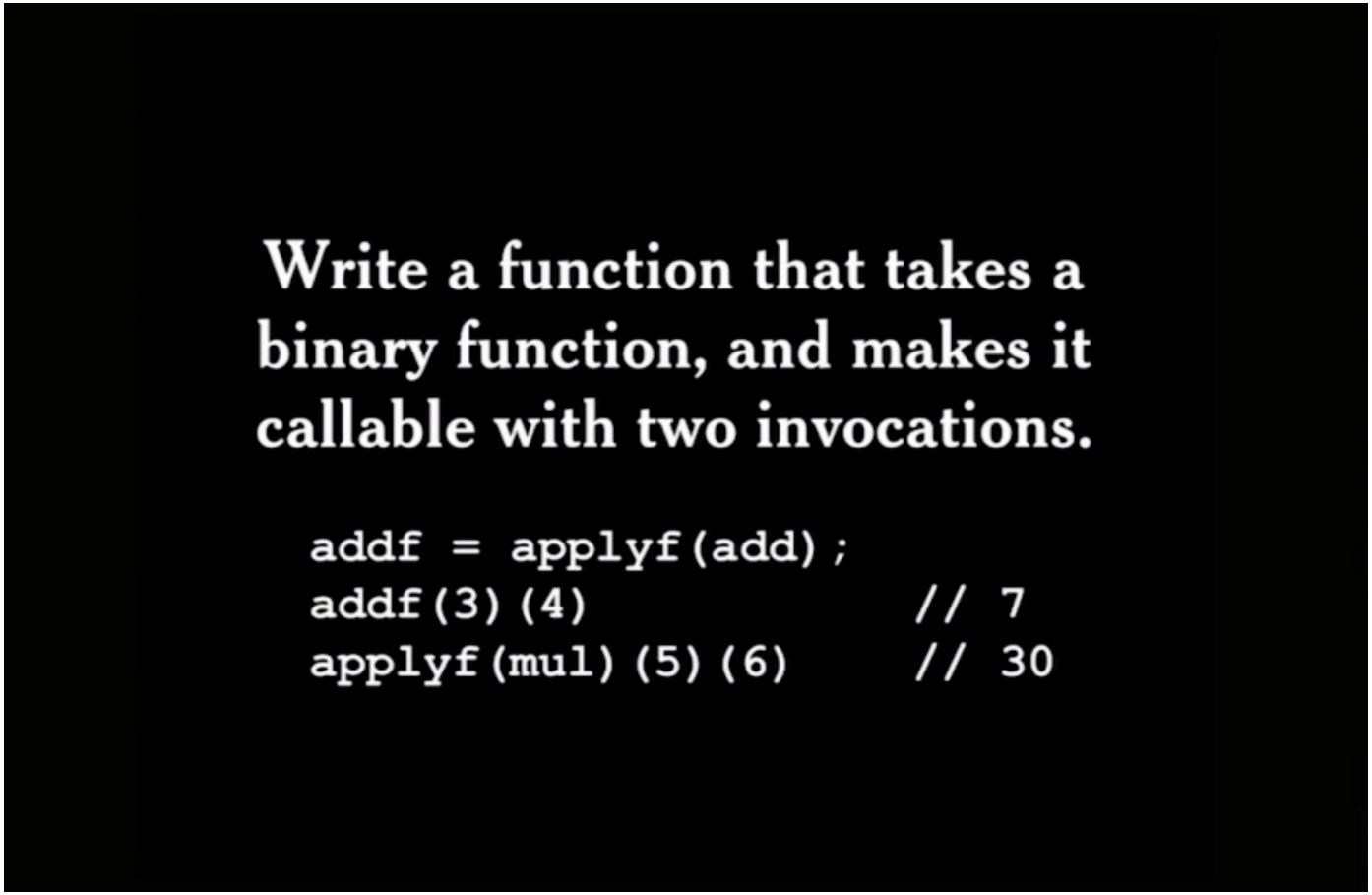
Write a function that takes an argument and returns a function that returns that argument.

```
idf = identityf(3);
idf()     // 3
```

How do we code ES5 today?



How do we code ES5 today?



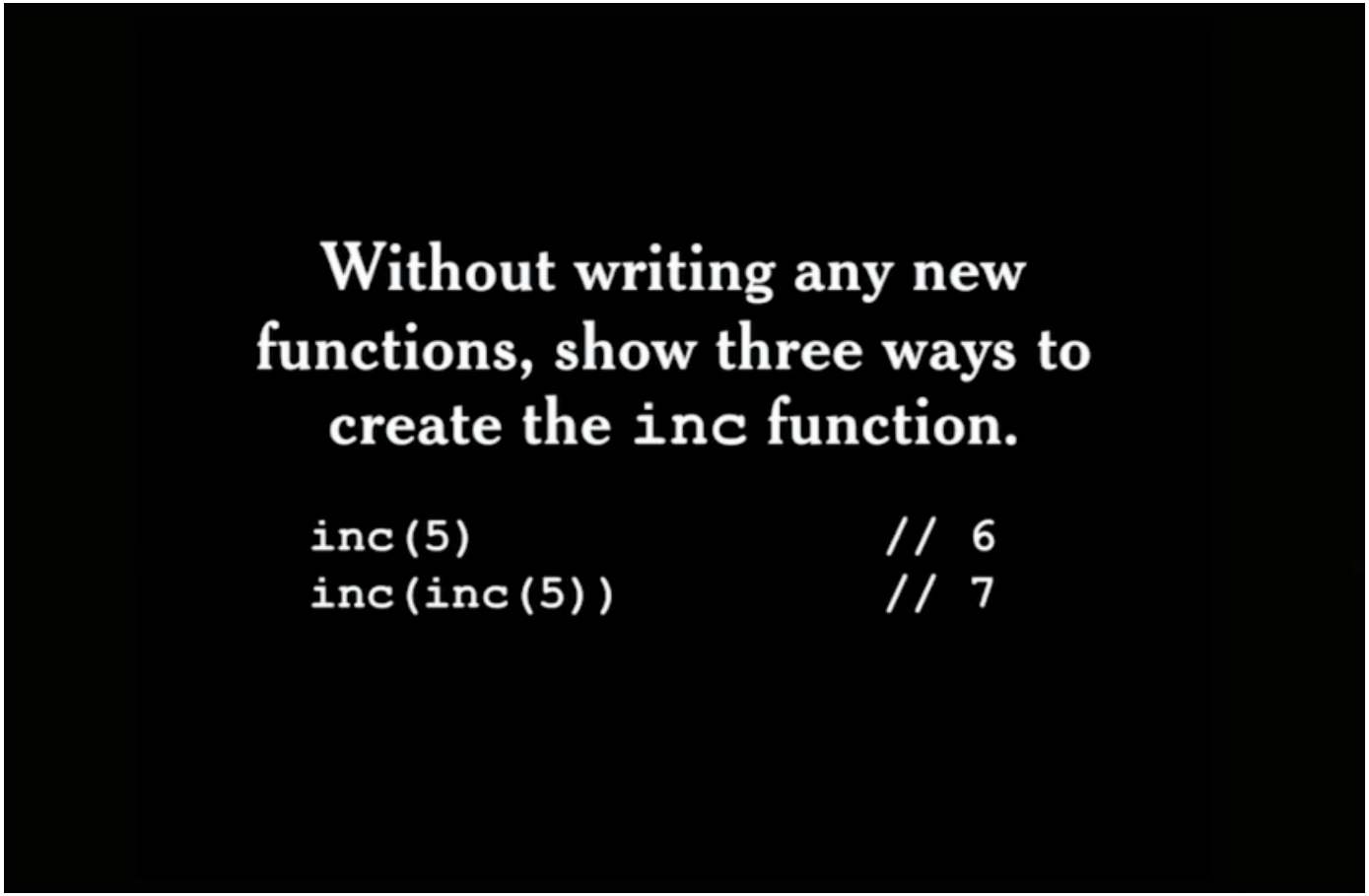
How do we code ES5 today?

Write a function that takes a function and an argument, and returns a function that can supply a second argument.

```
add3 = curry(add, 3);
add3(4)                    // 7

curry(mul, 5)(6)           // 30
```

How do we code ES5 today?

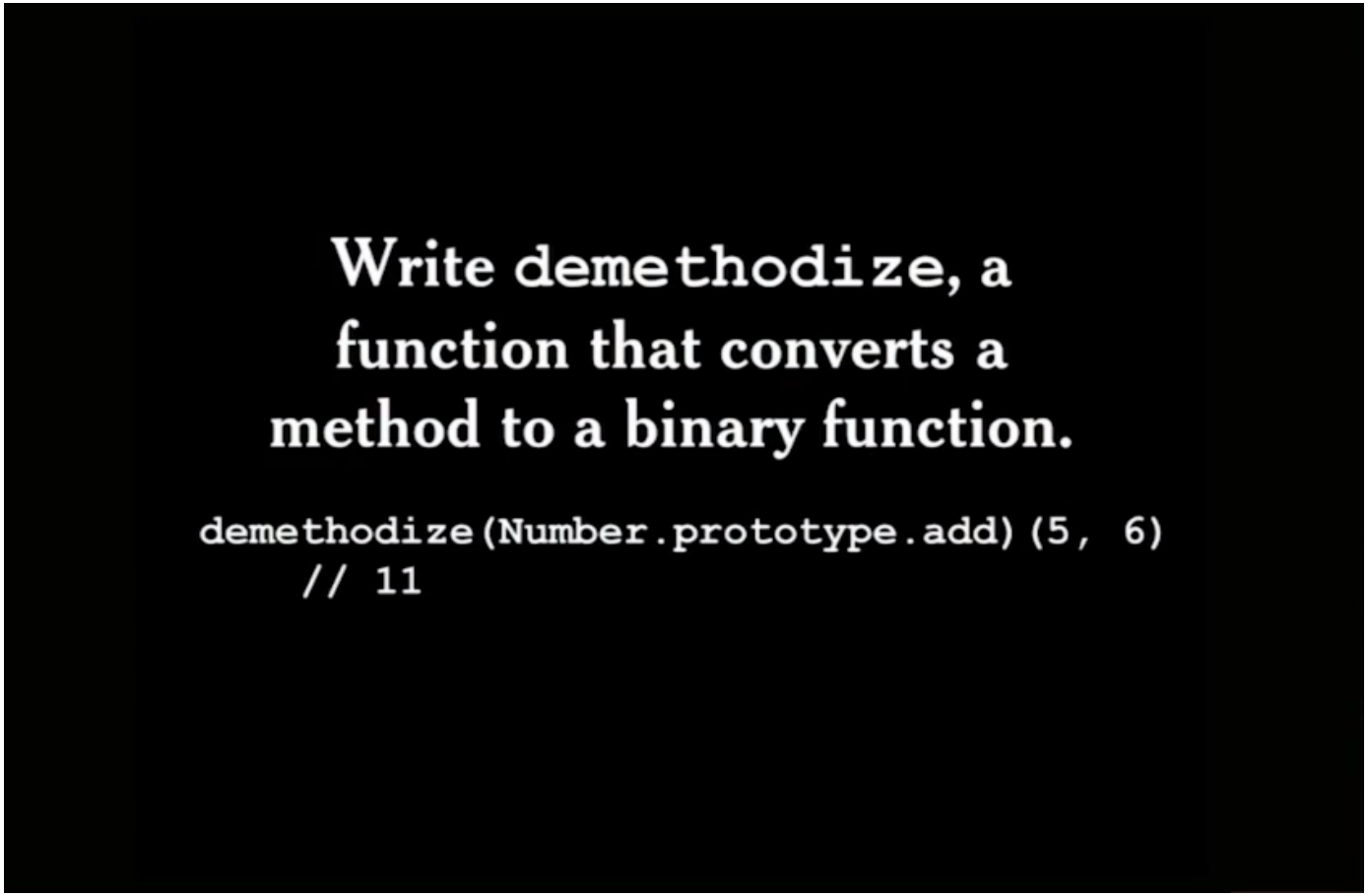


How do we code ES5 today?

Write `methodize`, a function that converts a binary function to a method.

```
Number.prototype.add =
    methodize(add);
(3).add(4)     // 7
```

How do we code ES5 today?

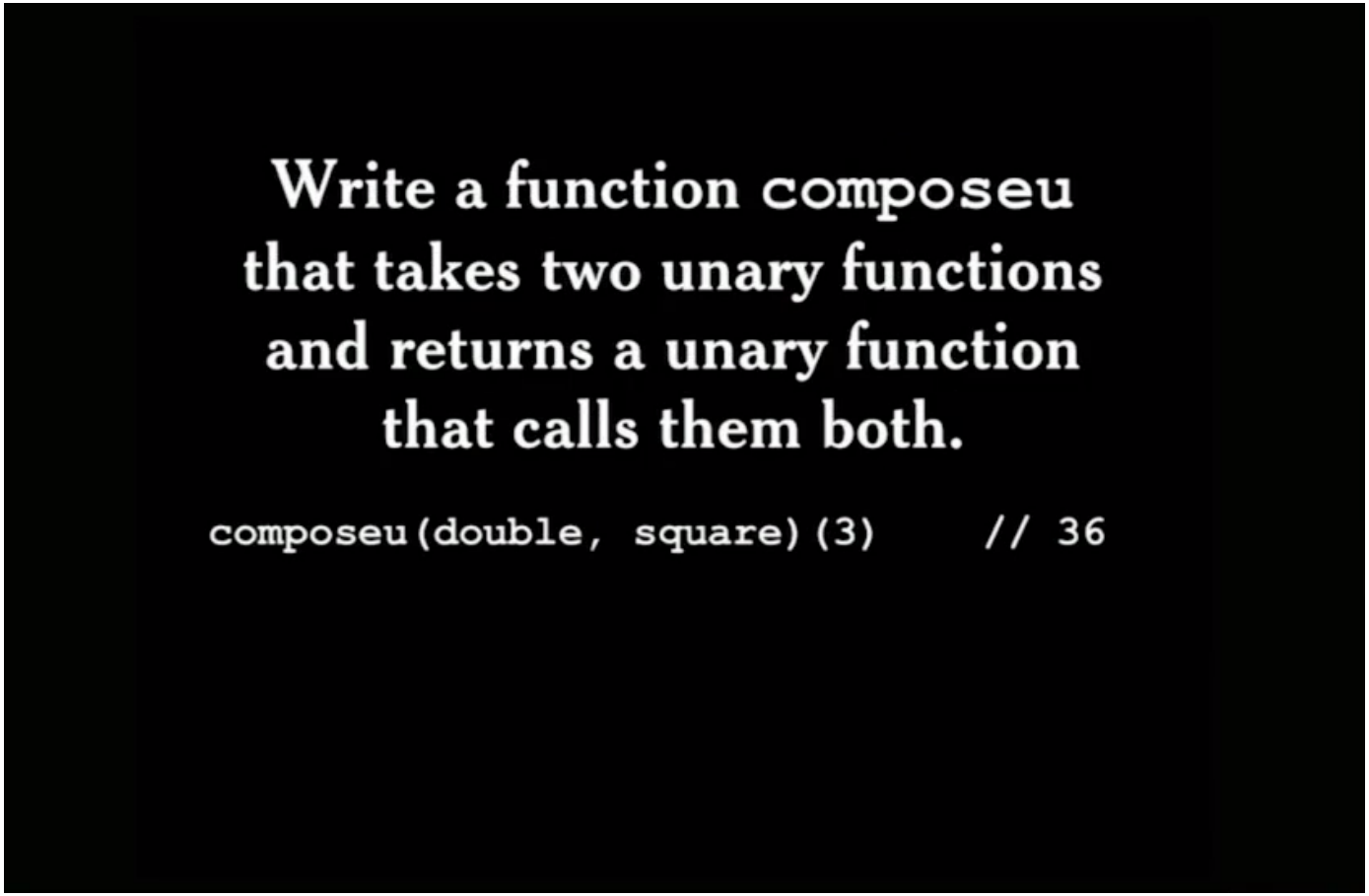


How do we code ES5 today?

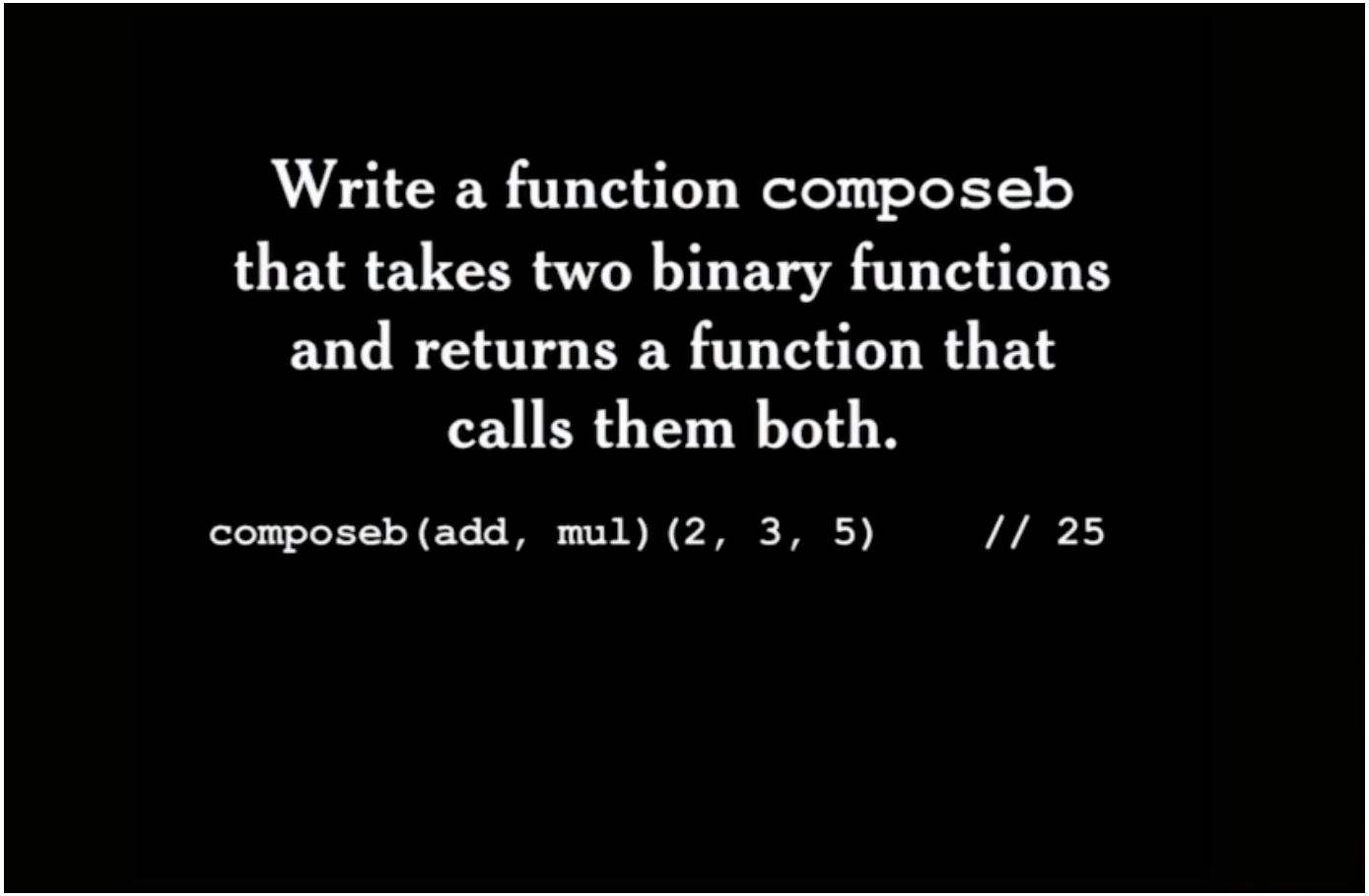
Write a function `twice` that takes a binary function and returns a unary function that passes its argument to the binary function twice.

```
var double = twice(add);
double(11)     // 22
var square = twice(mul);
square(11)     // 121
```

How do we code ES5 today?



How do we code ES5 today?



How do we code ES5 today?

Write a function that allows another function to only be called once.

```
add_once = once(add);
add_once(3, 4)     // 7
add_once(3, 4)     // throw!
```

How do we code ES5 today?

Write a factory function that returns two functions that implement an up/down counter.

```
counter = counterf(10);
counter.inc()     // 11
counter.dec()     // 10
```

How do we code ES5 today?

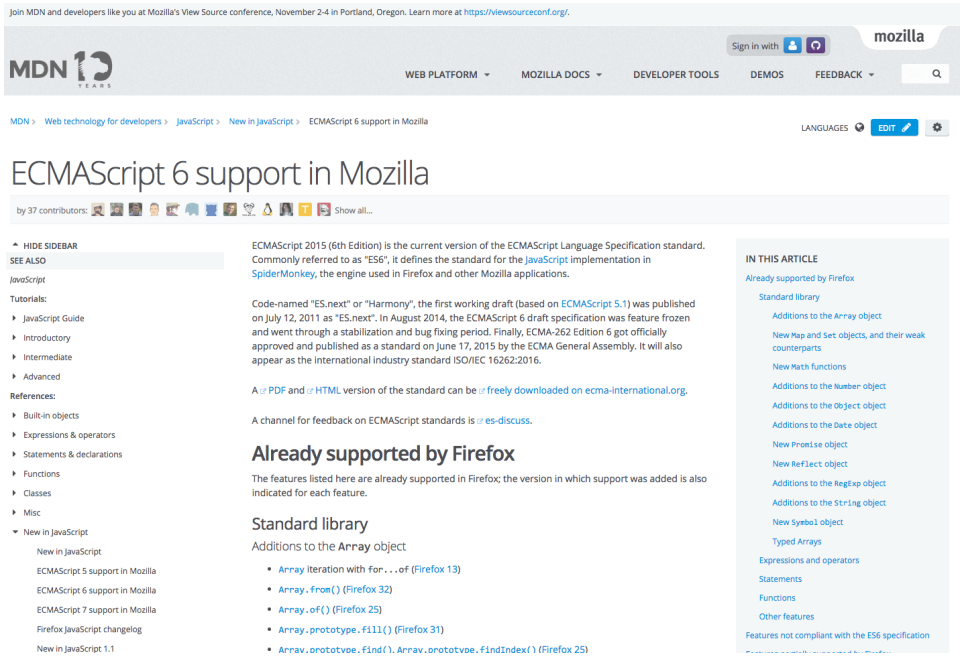
Make a revocable function that takes a nice function, and returns a `revoke` function that denies access to the nice function, and an `invoke` function that can invoke the nice function until it is revoked.

```
temp = revocable(alert);
temp.invoke(7);     // alert: 7
temp.revoke();
temp.invoke(8);     // throw!
```

ES6 - WARM UP -

ES6 - Warm up -

- Retrocompatibility is mandatory (some bad parts)
- Thanks to some new features JS is pushing to be the language of the future
- JavaScript is an implementation of ECMAScript specifications (TC39)
- Browsers are updating their JS engines with the new features and syntax

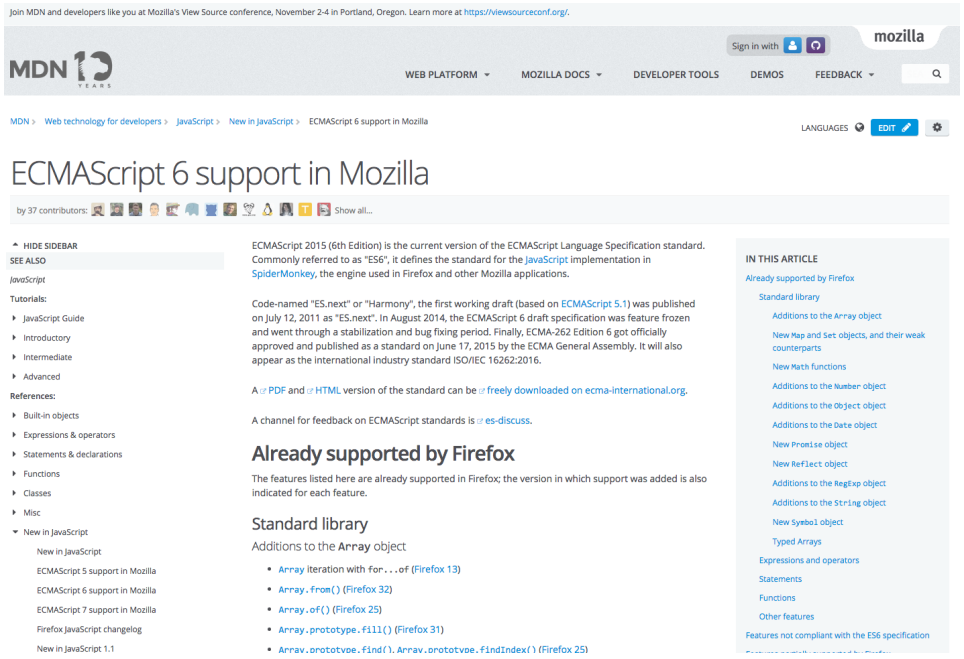


MDN - Mozilla Development Network

netcentric

ES6 - Warm up -

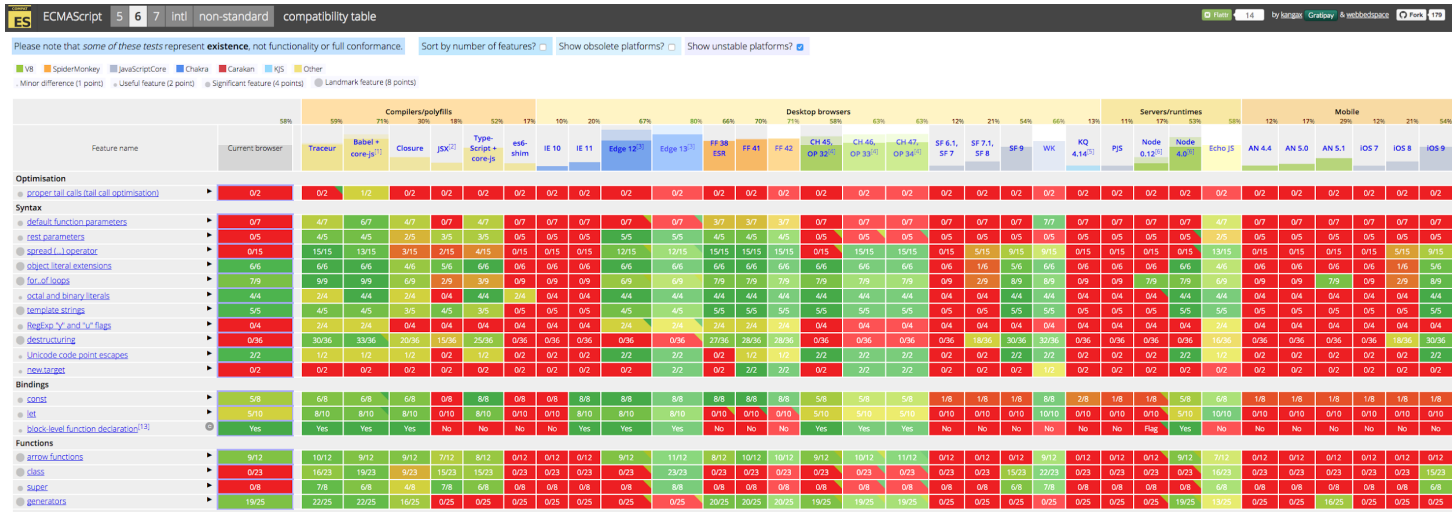
- ES6 brings the biggest amount of changes since the language was first released
- We don’t have an official release but we can try it already to help engine developers and TC39 and get a clean-bug-free final release.



MDN - Mozilla Development Network

netcentric

ES6 - Warm up -



Kangax compatibility list

netcentric

VARIABLES AND PARAMETERS

Variables and parameters

ES6 makes JavaScript:

- Easier to read
- Easier to write
- Safer (code wise)

Variables and parameters

let

- variable declaration expression var had scope limitations (function scope)
- let enables block scope for variables
- we should use let to avoid variable hoisting and its strange behaviour

- global scope: var outside a function or assignment without var
- function scope: var inside a function
- block scope: safer and very common in many other languages

coming up next:
“let” exercises

Variables and parameters

const

- read only property
- it can't be assigned any value after the first time
- block scoping

- "const" keyword already existed in the past

coming up next:
"const" exercises

Variables and parameters

destructuring

- operation to assign values to a set of variables by tearing apart a pattern or a structure
- it's not the construction of an array literal although the notation is the same
- it's just working with individual variables
- as parameters: we pass an object as argument and destructure it in variables instead of using *object.property* notation

coming up next:
"destructuring" exercises

Variables and parameters

default

- parameters now can have an implicit default state

- parameter: variable which is part of the method's signature (method declaration)
- argument: an expression used when calling the method

coming up next:
"default" exercises

Variables and parameters

rest

- allows working with a variable or an unknown number of arguments in a function
- three dot notation “...”

- in the past (ES5) we had an implicit array like structure called arguments

coming up next:
“rest” exercises

Variables and parameters

spread

- exactly the same notation as rest: three dot notation "..."
- in this case it's used outside the function argument list
- allows to spread an array into individual parameters

coming up next:
"spread" exercises

Variables and parameters

template literals and tags

- templates to replace string concatenation
- we can have expressions inside templates
- we can associate a tag with a template
- tags are invoked in run time

coming up next:
“spread” exercises

A LITTLE BIT OF FUNCTIONS

A little bit of functions

functional programming

- (fat) arrow function: short and expressive statement
- single parameter doesn't require ()
- single return statement doesn't require {}
- no parameter requires empty ()

asynchronous behaviour and lexical this

- arrow functions capture the context they are in and where *this* is defined
- we can it: lexical binding of *this* but call/apply can't overwrite it anymore

coming up next:
"function" exercises

A little bit of functions

iterators

- any iterable object can give us its iterator
- an iterator is an object with the method: *next()*
- it has a property: *done* which is always *false* until it reaches the end of the iterable when it becomes *true*
- an iterator allows us to work with data forgetting about the source of the data (map, array, object, DOM…)
- ES5 loop: for in - using keys
- ES6 loop: for of - using the element itself
- generator: a function that generates iterators (yield, filters…)

coming up next:
“iterators” exercises

CLASSES AND MODULES

Classes and modules

class

- a blueprint for creating objects
- we create an object when we instantiate a class
- prototype + constructor === classes (under the hood is the same)
- syntax is clearer
- constructor() method is called when a class is instantiated (it may not be defined)
- we can define getters and setters instead of normal functions to get and set
- if we create a getter but not a setter we have a read only property
- constructors can use getters and setters instead of accessing the properties

Classes and modules

class

- three pillars of OOP: polymorphism, encapsulation and inheritance (reuse)
- *class Employee extends Person* => employee is a person
- deep inheritance can be complex to follow and understand
- super() allows calling the superclass method of the same name
- super().method allows calling any method of the super class
- we can call a method of the own class using this.method
- every class extends Object (implicit) so we can override default methods like toString()

coming up next:
"class" examples

Classes and modules

module

- enables: code organization and visibility control
- Till now we used:
  - IIFE: immediately invoked function expression - allowed to have everything scoped inside the function and privately implemented although we can attach elements to the windows object
  - common.js (2009): exports (makes object public) + require (gets the object), widely adopted and very popular. It's self contained like IIFE but without the ugly syntax. No need to work on global scope to share the code and yet being able to hide implementation.
  - AMD (require.js): asynchronous model definition, like IIFE but without being executed immediately. AMD syntax favours browser environment.

Classes and modules

module

- ES6 modules: very similar to common.js and AMD
- It's an attempt to unify a module API but it's not yet final
- common.js and require.js are popular, accepted and stable, they are not going away anytime soon
- we need webpack to run ES6 modules (
  - Netcentric's complex frontend structure
  - Mercedes frontend setup explained
  - ES6 Module Boilerplate

coming up next:
"module" super awesome live code demo

SUMMARY

Summary

Pluralsight courses: JavaScript Fundamentals for ES6
JS.Next: ES6

Thank you!

netcentric