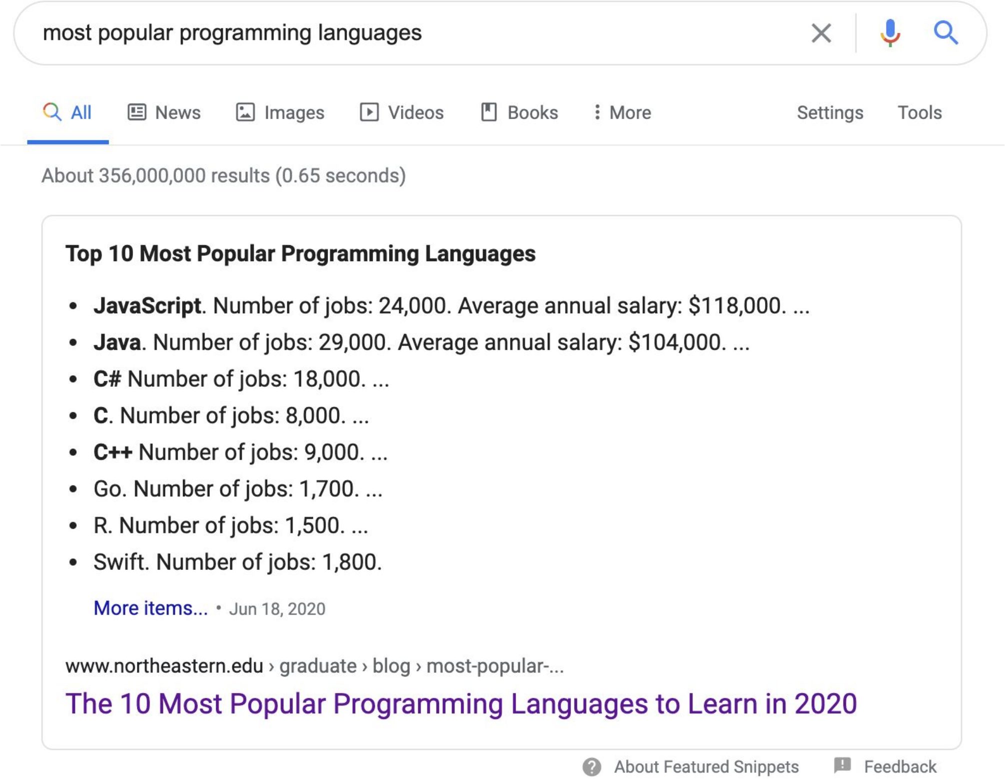
CS188
Scalable Internet Services

John Rothfels, 10/8/20

Motivation



most popular programming languages

About 356,000,000 results (0.65 seconds)

Top 10 Most Popular Programming Languages

- **JavaScript**. Number of jobs: 24,000. Average annual salary: $118,000. ...
- **Java**. Number of jobs: 29,000. Average annual salary: $104,000. ...
- **C#** Number of jobs: 18,000. ...
- **C**. Number of jobs: 8,000. ...
- **C++** Number of jobs: 9,000. ...
- Go. Number of jobs: 1,700. ...
- R. Number of jobs: 1,500. ...
- Swift. Number of jobs: 1,800.

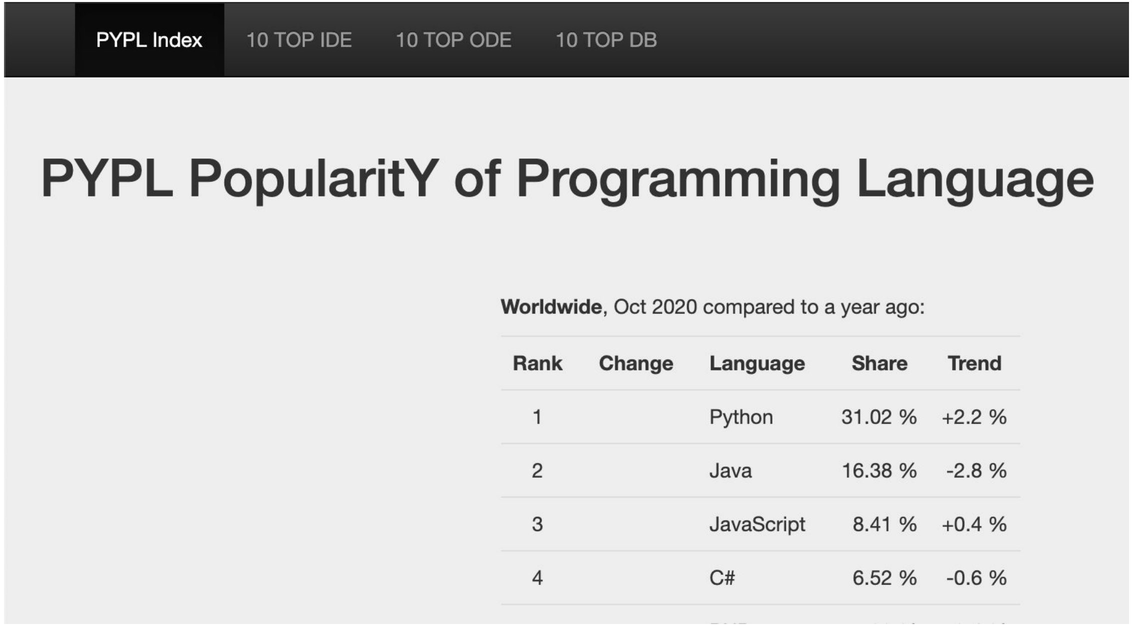
More items... • Jun 18, 2020

www.northeastern.edu › graduate › blog › most-popular-...

The 10 Most Popular Programming Languages to Learn in 2020

Motivation

http://pypl.github.io/PYPL.html



PYPL PopularitY of Programming Language

Worldwide, Oct 2020 compared to a year ago:

Rank	Change	Language	Share	Trend
1		Python	31.02 %	+2.2 %
2		Java	16.38 %	-2.8 %
3		JavaScript	8.41 %	+0.4 %
4		C#	6.52 %	-0.6 %

The browser wars

In the mid-90s, Netscape reigns supreme.

Microsoft releases initial version of Internet Explorer in 1995.

Competition between Netscape and Microsoft produces significant innovation in browsers.

- JavaScript
- Cookies
- CSS

The browser wars

Microsoft bundles Internet Explorer to Windows 98.

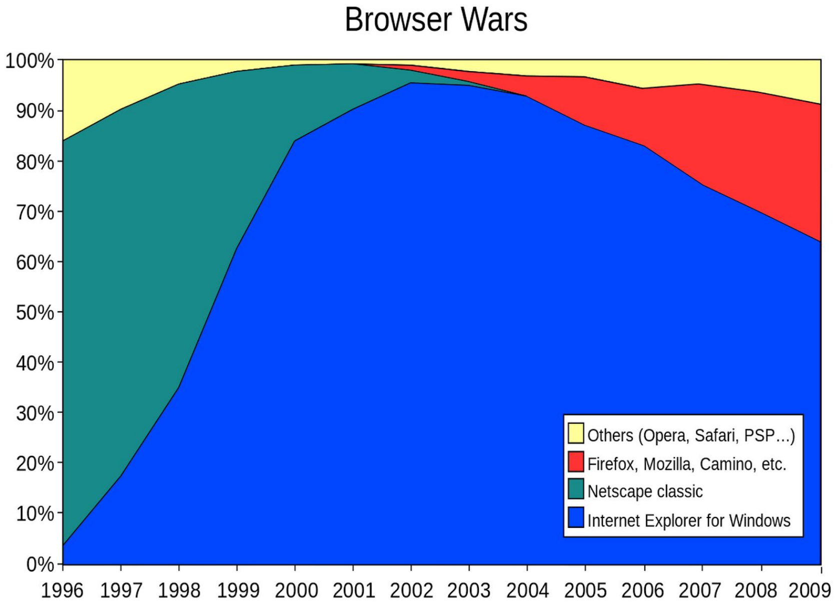
- Every file management window is a browser!
- Eventually triggers an antitrust lawsuit against Microsoft. 💣

Meanwhile, Netscape focuses on open-sourcing its browser.

- Eventually creates the Mozilla foundation. 🦊
- Gets acquired by AOL.

The browser wars

Microsoft wins. Internet Explorer becomes the dominant browser for roughly a decade.



The browser wars

Lull in browser innovation commences. Lack of competition means no reason to innovate. Time between releases increases significantly. **The dark ages**.

Version	Release Year
IE1	1995
IE2	1995
IE3	1996
IE4	1997
IE5	1999
IE6	2001
IE7	2006
IE8	2009

The browser wars

Due to a variety of factors, Microsoft slowly loses market share to Firefox (Mozilla).

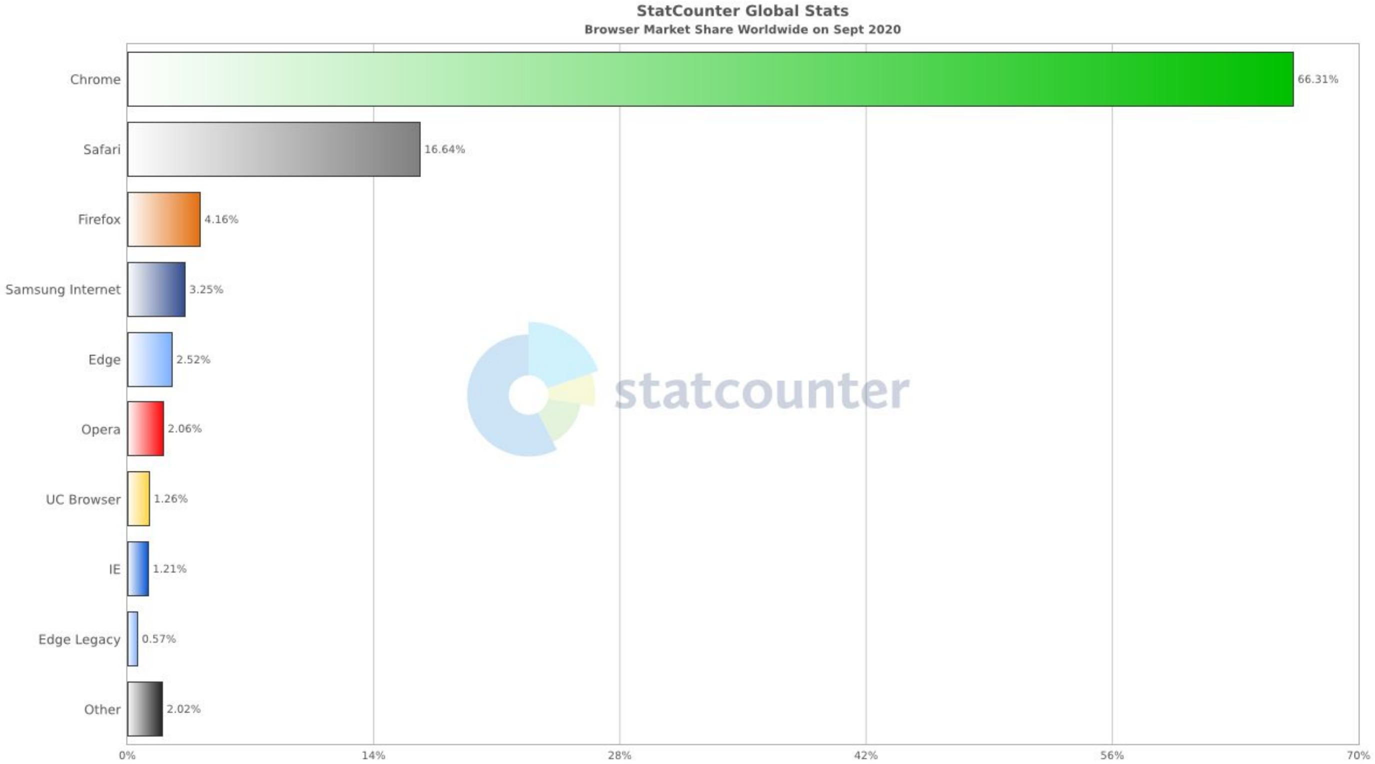
- Better security
- Better performance

As Microsoft is slowly bleeding market share, Google announces Chrome.

Browser innovation reignites! 🔥🧨

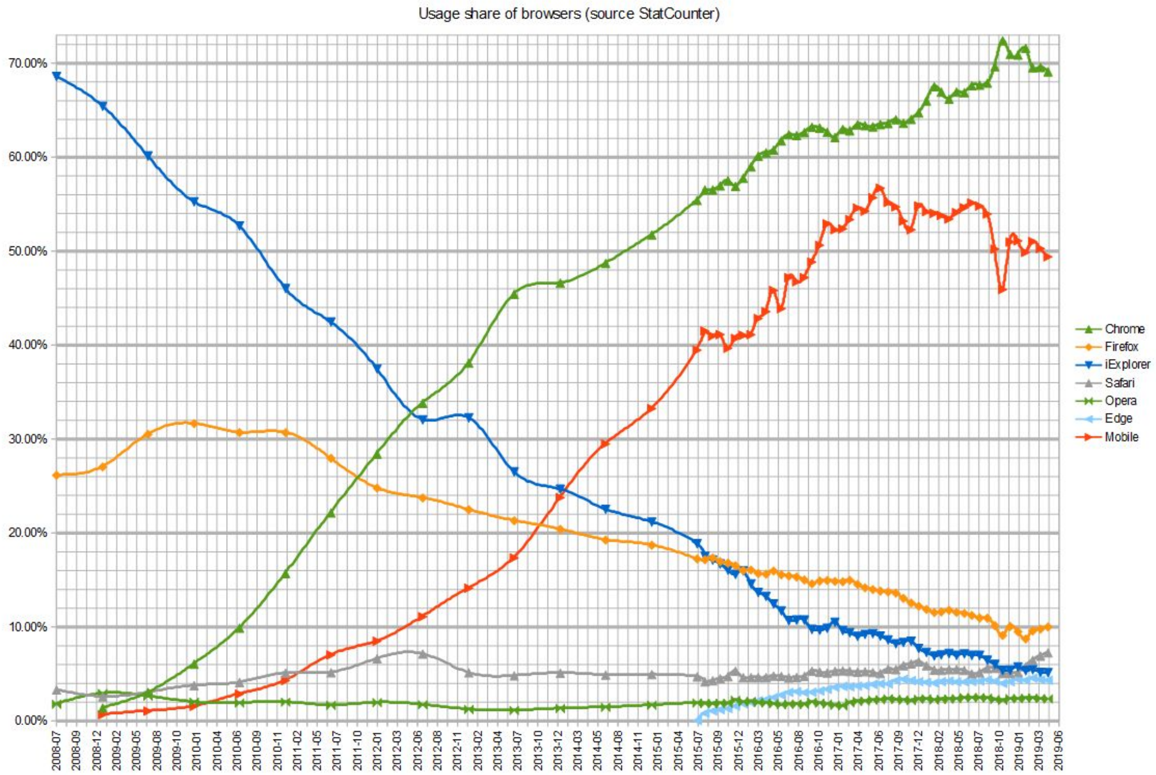
Today there are at least 4 viable browsers.

The browser wars



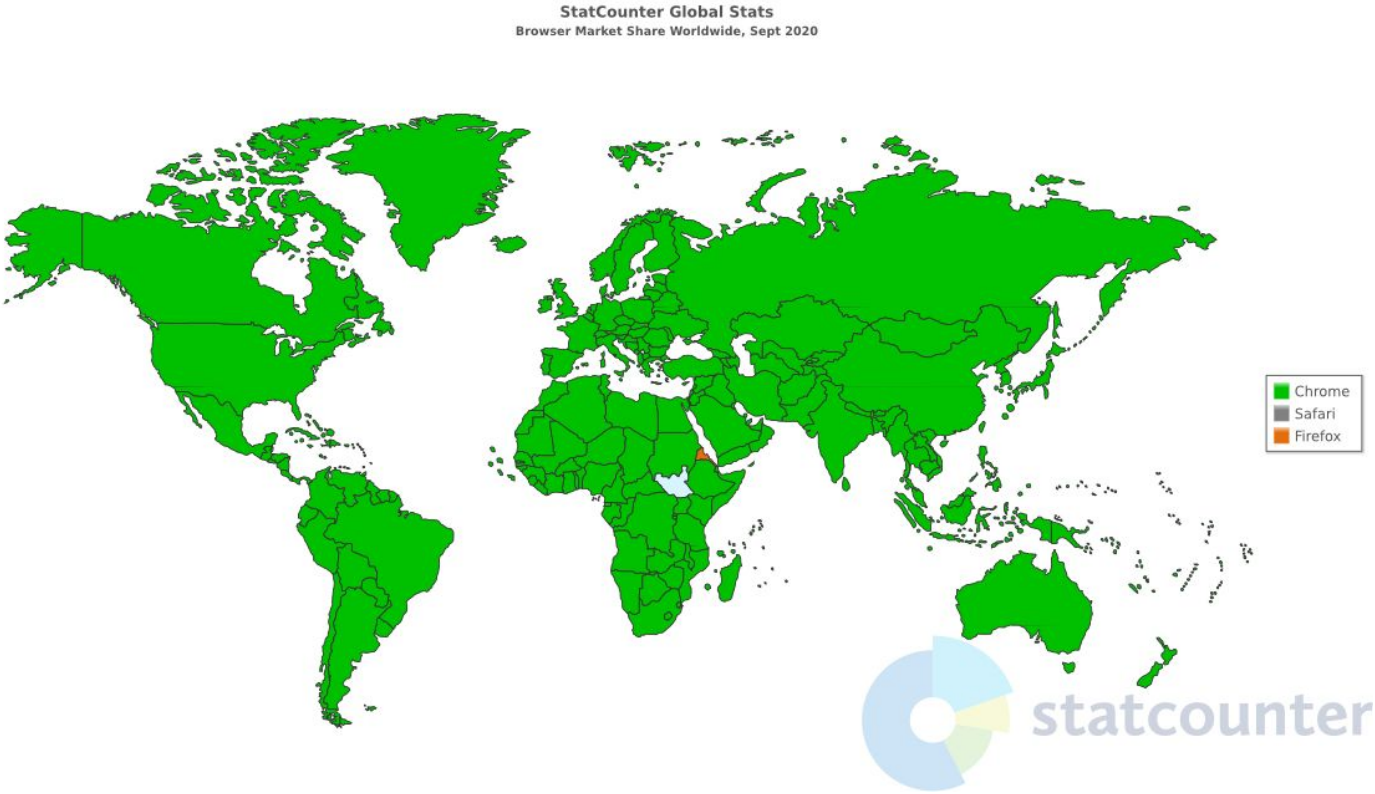
source

The browser wars



source

The browser wars



source

JavaScript renaissance

During the browser dark ages, three things eventually spur the JavaScript renaissance:

- **`XMLHTTPRequest`**
- DOM manipulation
- V8

JavaScript renaissance

`XMLHttpRequest` (Ajax) allows JavaScript on the page to asynchronously request resources from the server.

```
var req = new XMLHttpRequest();
req.onload = function() {
  console.log(“I’m a callback!”);
};
req.open("get", "/comments", true);
req.send();
```

Originally added to IE to enable the Outlook Web Access team to asynchronously communicate with the server (year ~2000). Other browsers implement it and it becomes the de facto standard by ~2004.

JavaScript renaissance

`XMLHttpRequest` (Ajax) allows JavaScript on the page to asynchronously request resources from the server.

Prior to this, two-way communication with the server meant a full page refresh.

- A link was clicked, or a form was submitted.

This method allows JavaScript in your browser to send requests and receive responses completely programmatically.

- E.g. as you type, google.com will show you intermediate results.

Most people use a library abstraction (e.g. jQuery) instead of XHR directly.

JavaScript renaissance

The **D**ocument **O**bject **M**odel (DOM) is a standardized way of representing the structure of a web page as a tree of in-memory objects (HTML elements). These objects can be accessed via JavaScript and can be queried and manipulated.

```
var newDiv = document.createElement("div");
var newContent = document.createTextNode("Hello World!");
newDiv.appendChild(newContent);
document.body.appendChild(newDiv);
```

JavaScript renaissance

September 2008, Google releases Chrome. In addition to other novel features, it includes the **V8 JavaScript engine**.

V8 applies modern, state of the art VM techniques to JavaScript:

- Not interpreted: JavaScript gets dynamically compiled to machine code
- Garbage collector is fast
    - Generational: separates allocated memory into young and old groups, treats them differently'
    - Incremental: doesn't need to perform all GC at once
- Applies other modern VM optimizations:
    - Inlining, code elision, inline caching, etc.

JavaScript renaissance

V8 treats JavaScript performance seriously, and triggers other browsers to do the same.

- Safari’s JavaScriptCore
- IE’s Chakra
- Firefox’s SpiderMonkey

Today, these VMs are all roughly evenly matched. The performance leader goes back and forth between Chrome and Firefox (usually).

ℹ️ V8 was designed to also work well outside the browser. It is the execution engine that Node.JS is built on and we will be using it for class projects.

JavaScript renaissance

By 2008, we have all the ingredients ready for a JavaScript renaissance:

- Globally installed virtual machines
- …that can present content that can communicate via Ajax to the internet service that originated it
- …with full programmatic control of the user interface (DOM)
- …that use modern, high performance VM techniques
- …that exist in a competitive marketplace
    - Four viable browsers available on multiple operating systems
    - All competing to stay ahead of the back
    - Standards compliance is a competitive advantage

These are things we enjoy today we did not always have. 🥺

JavaScript renaissance

What does the renaissance give us?

- Better JavaScript (language features)!
- JavaScript running on the server!
- Web apps rendered in the browser (client-side) instead of the on the server
    - Instead of being a series of pages requested from a web server, we can serve a JavaScript application which draws the app entirely in the browser
    - This application is regularly fetching new data from the server, sending user input over Ajax
    - This application is regularly receiving structured data (JSON) instead of rendered markup (HTML).
    - Clicks don’t result in page refreshes, they execute JavaScript to change how the current page is displayed
    - Communication with the server may be decoupled from user interaction: for example, while the browser sits open, a javascript timer can check for new data and update the page as needed

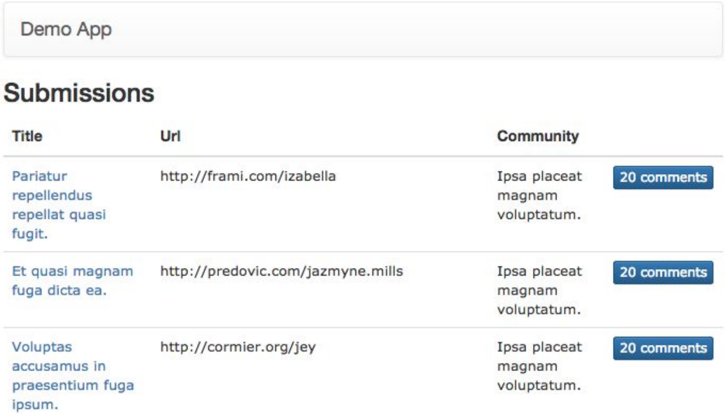
JavaScript renaissance

Consequences of shift to JavaScript-heavy client-side apps:

- Client side logic is much more complex and full page refreshes are more rare
- It’s possible to build applications that work “offline”
- It’s possible to build effective “push” mechanisms
- The “running application” is much more static and cacheable
- The APIs you build to serve up structured data can be used by mobile applications and other internet services
- JavaScript VMS enable very ambitious use of CPU resources 🎉

JavaScript renaissance

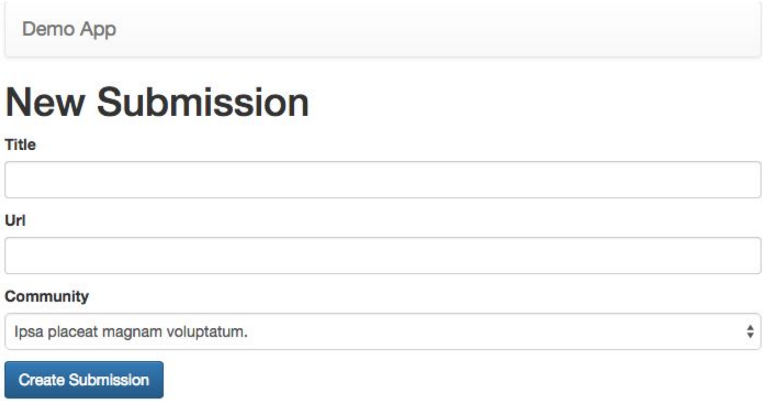
Consider an app that may be rendered by the server or rendered on the client.



Demo App

Submissions

Title	Url	Community	
Pariatur repellendus repellat quasi fugit.	http://frami.com/izabella	Ipsa placeat magnam voluptatum.	20 comments
Et quasi magnam fuga dicta ea.	http://predovic.com/jazmyne.mills	Ipsa placeat magnam voluptatum.	20 comments
Voluptas accusamus in praesentium fuga ipsum.	http://cormier.org/jey	Ipsa placeat magnam voluptatum.	20 comments



Demo App

New Submission

Title

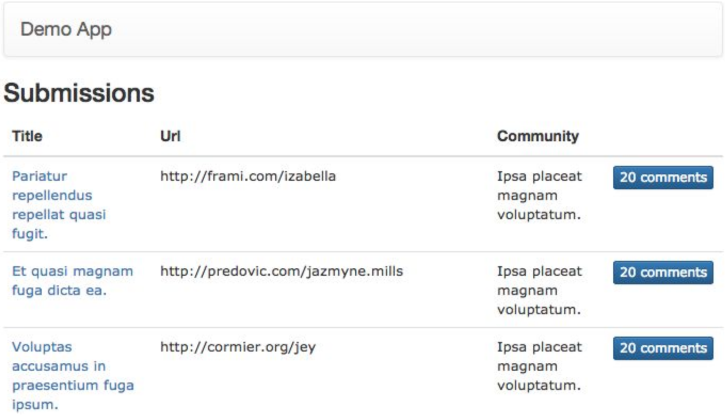
Url

Community

Ipsa placeat magnam voluptatum.

JavaScript renaissance

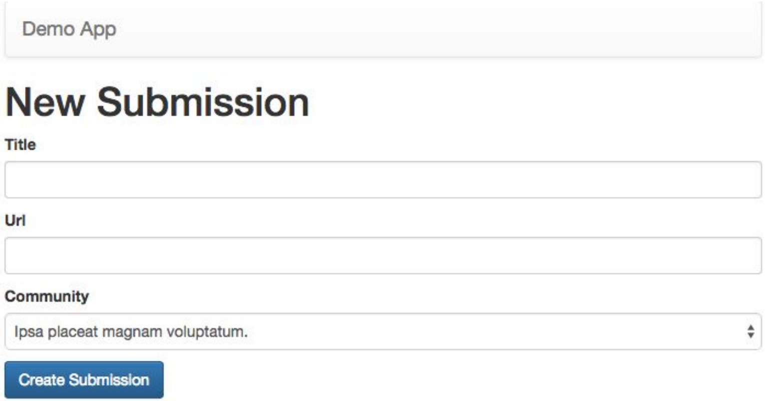
Consider an app that may be rendered by the server or rendered on the client.



Demo App

Submissions

Title	Url	Community	
Pariatur repellendus repellat quasi fugit.	http://frami.com/izabella	Ipsa placeat magnam voluptatum.	20 comments
Et quasi magnam fuga dicta ea.	http://predovic.com/jazmyne.mills	Ipsa placeat magnam voluptatum.	20 comments
Voluptas accusamus in praesentium fuga ipsum.	http://cormier.org/jey	Ipsa placeat magnam voluptatum.	20 comments



Demo App

New Submission

Title

Url

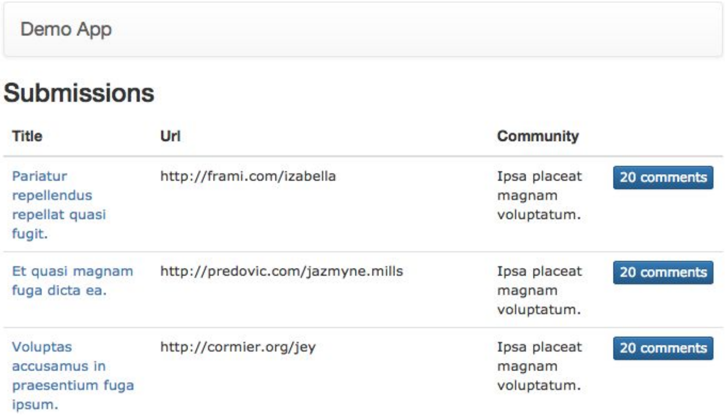
Community

Ipsa placeat magnam voluptatum.

In a server rendered app, when you click the “Create Submission” button (which is inside a `<form>`), the browser makes an HTTP **`POST`** request and loads the response. The response is a web page (HTML) with a list of submissions that includes the new submission you created. The list only changes when you refresh the page even if there was a new submission on the server.

JavaScript renaissance

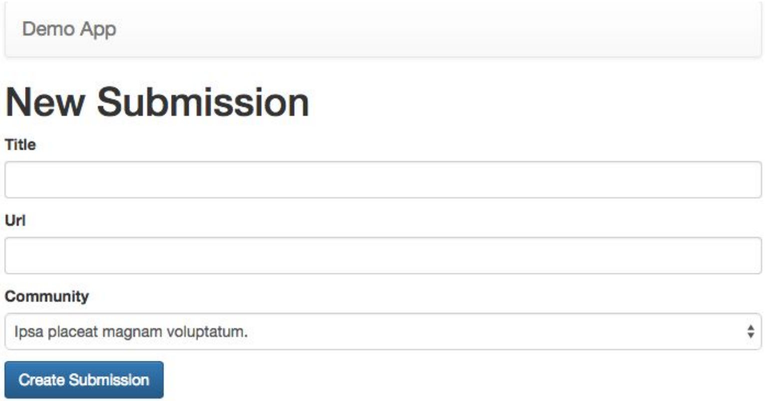
Consider an app that may be rendered by the server or rendered on the client.



Demo App

Submissions

Title	Url	Community	
Pariatur repellendus repellat quasi fugit.	http://frami.com/izabella	Ipsa placeat magnam voluptatum.	20 comments
Et quasi magnam fuga dicta ea.	http://predovic.com/jazmyne.mills	Ipsa placeat magnam voluptatum.	20 comments
Voluptas accusamus in praesentium fuga ipsum.	http://cormier.org/jey	Ipsa placeat magnam voluptatum.	20 comments



Demo App

New Submission

Title

Url

Community

Ipsa placeat magnam voluptatum.

Create Submission

In a client rendered app, when you click the “Create Submission” button ~~(which is inside a `<form>`)~~, ~~the browser~~ JavaScript makes an HTTP **`POST`** request and loads the response. The response is ~~HTML~~ a JSON of new submission you created. JavaScript navigates the browser URL to the submissions list and a cache of data provides the list. JavaScript adds your new submission to the cache/list. Periodically JavaScript asks the server if there’s a new submission….

JavaScript renaissance

Consider an app that may be rendered by the server or rendered on the client.

Benefits to client-side rendering?

- UI is extremely responsive
- Less network traffic
- Live updates!
- ...

Costs:

- Client code is much more complex. 😢

JavaScript renaissance

Client-side HTML rendering w/ JavaScript is usually more complex than server-side rendering. But you can do more with it! The client-side JavaScript must:

- Know the relationship between input events and corresponding DOM updates
- Know enough application logic to distinguish valid input from invalid input
- Keep a connection to the server and display updates as they come in

ℹ️ Applications can be made to be rendered on *both* client and server using the same code. Your starter project / the course website is one such example.

JavaScript renaissance

How should our application design adjust to this increase in complexity on the client?

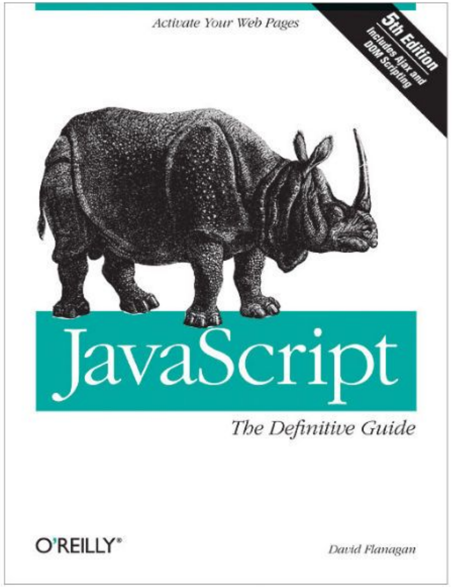
One approach: a tangled pile of jQuery selectors and callbacks, all trying frantically to keep data in sync between the HTML UI, your JavaScript logic, and the database on your server. 😅

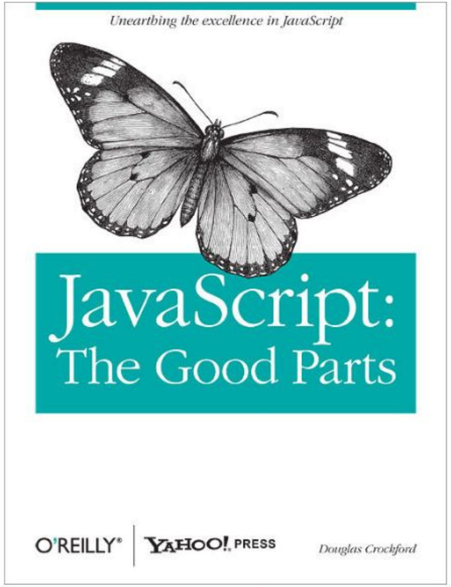
A better approach: some kind of MVC (“model-view-controller) framework

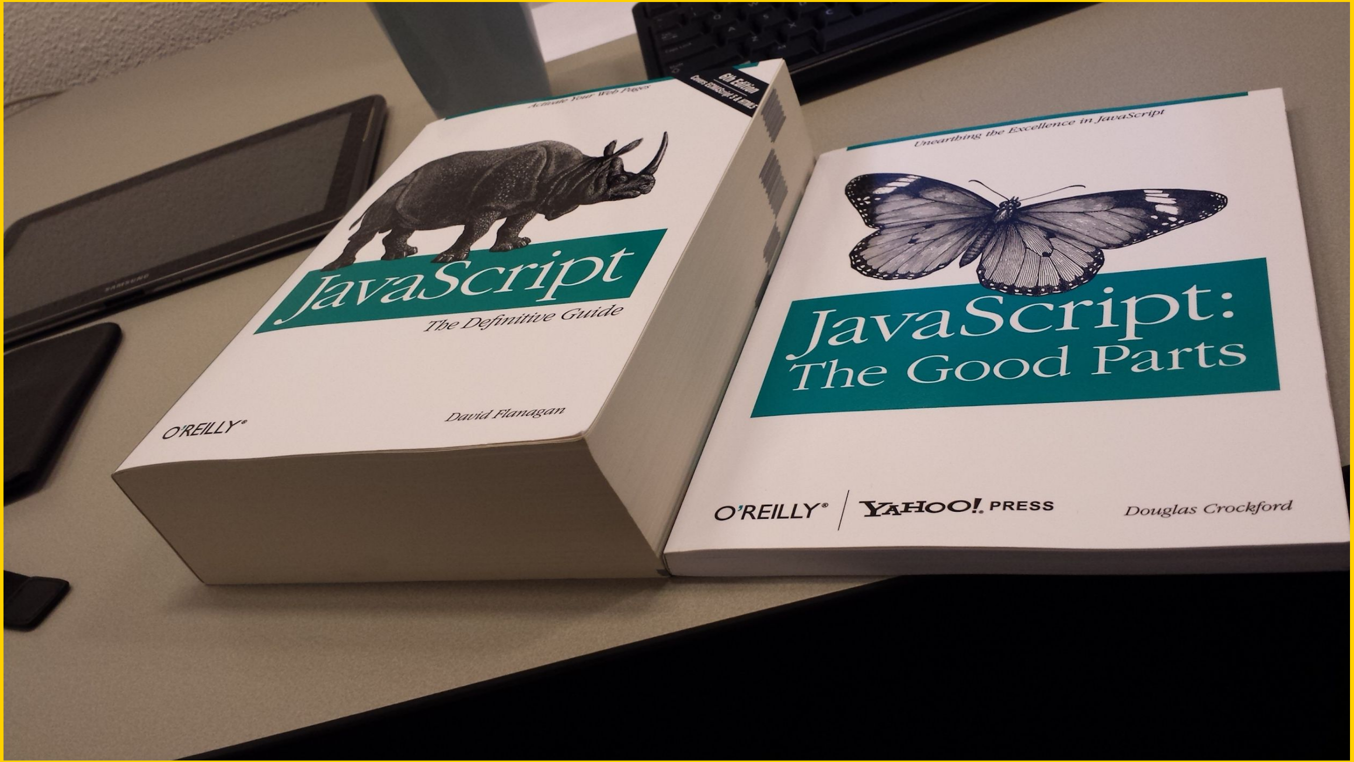
- React, Vue, Angular

JavaScript renaissance

How should our application design adjust to this increase in complexity on the client?






Activate Your Web Pages
6th Edition
Covers ECMAScript 5 & HTML5
JavaScript
The Definitive Guide
O'REILLY®
David Flanagan
Unearthing the Excellence in JavaScript
JavaScript:
The Good Parts
O'REILLY®
YAHOO! PRESS
Douglas Crockford

JavaScript renaissance

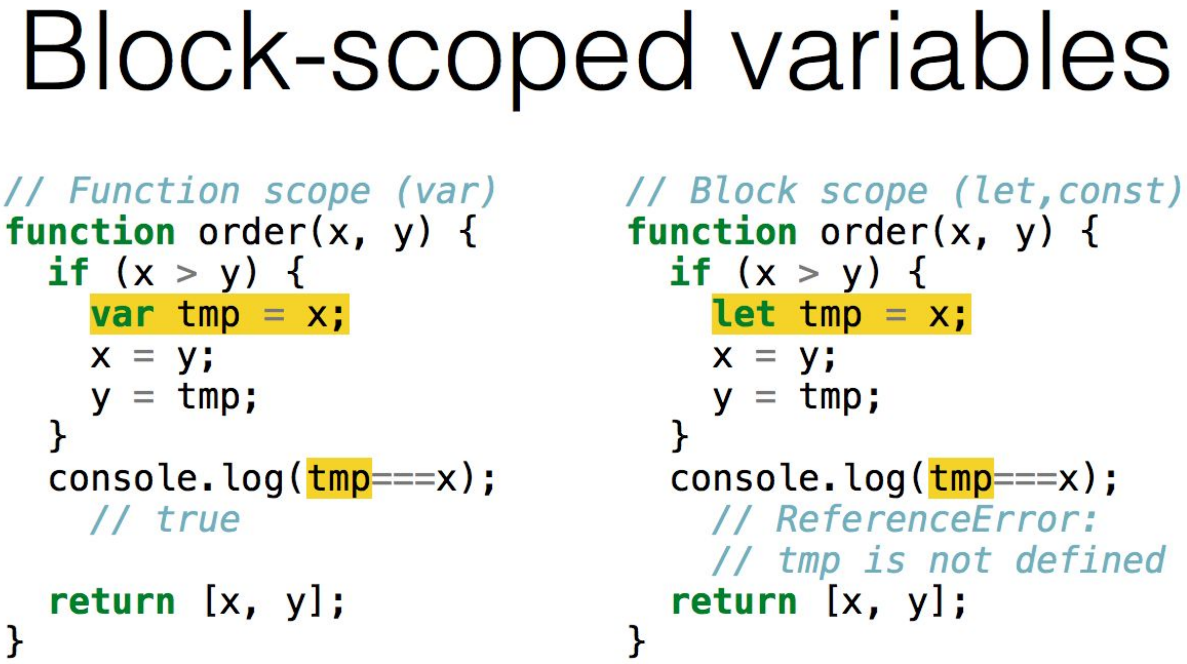
JavaScript has grown up a lot. New language features (e.g. **`Promise`**, **`async`**/**`await`**) make the language *significantly* easier to work with.

What's challenging about JavaScript these days is that there's so many different versions of it, and not everything can run the newest versions with the hottest features.

You'll typically always be using some sort of build tool (e.g. webpack, parcel, babel) to transform the JavaScript you write into the JavaScript which browser engines can actually run.

If you're interested, read more about the current state of JavaScript.

ECMAScript 6

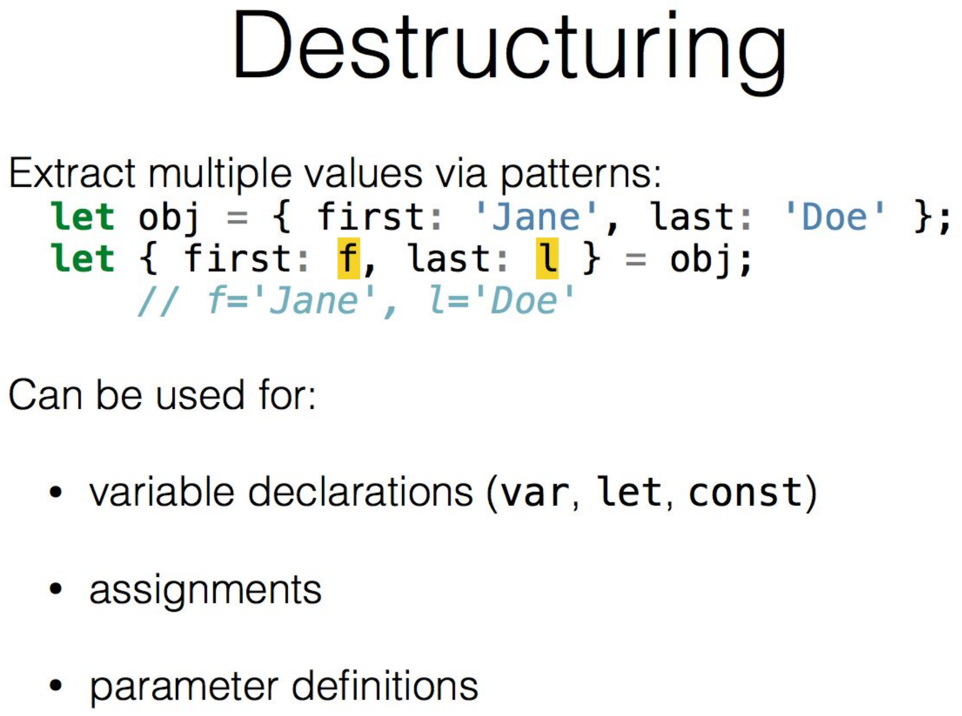


Block-scoped variables

```
// Function scope (var)
function order(x, y) {
  if (x > y) {
    var tmp = x;
    x = y;
    y = tmp;
  }
  console.log(tmp===x);
    // true

  return [x, y];
}
```

```
// Block scope (let,const)
function order(x, y) {
  if (x > y) {
    let tmp = x;
    x = y;
    y = tmp;
  }
  console.log(tmp===x);
    // ReferenceError:
    // tmp is not defined
  return [x, y];
}
```

ECMAScript 6



Destructuring

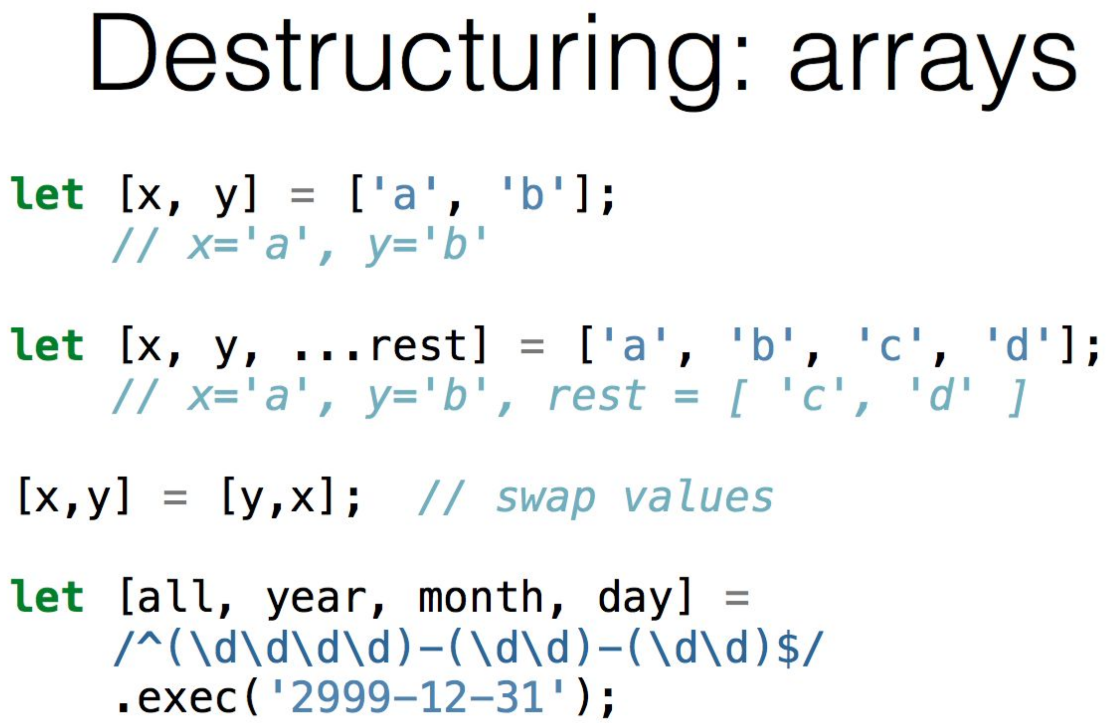
Extract multiple values via patterns:

```
  let obj = { first: 'Jane', last: 'Doe' };
  let { first: f, last: l } = obj;
      // f='Jane', l='Doe'
```

Can be used for:

- variable declarations (`var`, `let`, `const`)
- assignments
- parameter definitions

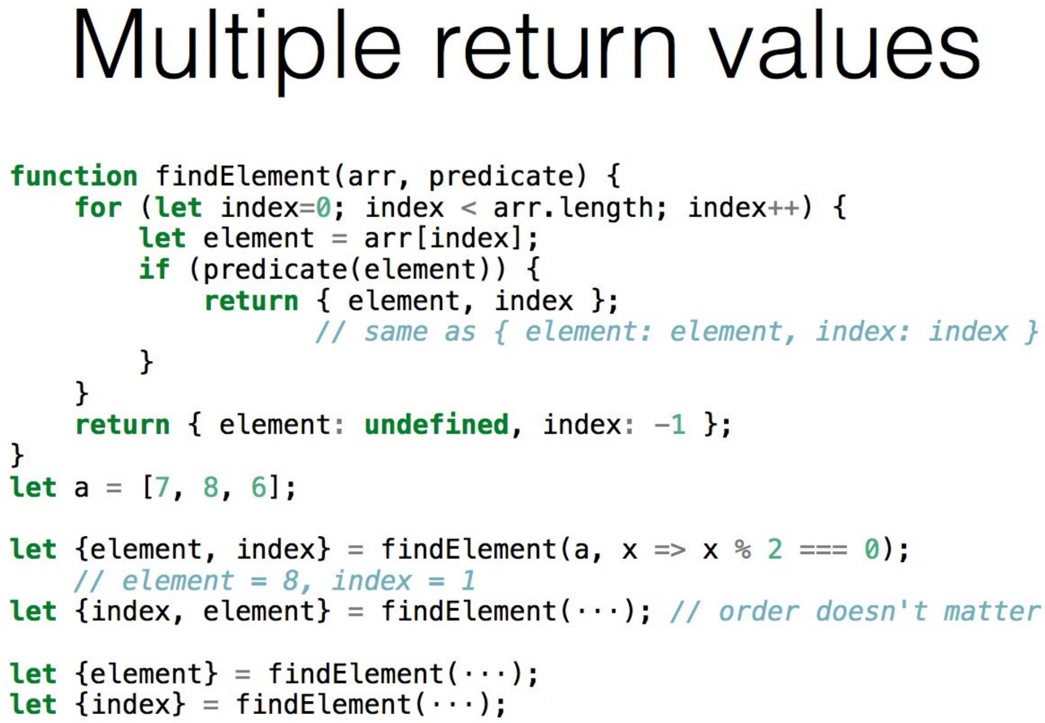
ECMAScript 6



Destructuring: arrays

```
let [x, y] = ['a', 'b'];
    // x='a', y='b'

let [x, y, ...rest] = ['a', 'b', 'c', 'd'];
    // x='a', y='b', rest = [ 'c', 'd' ]

[x,y] = [y,x];  // swap values

let [all, year, month, day] =
    /^(\d\d\d\d)-(\d\d)-(\d\d)$/
    .exec('2999-12-31');
```

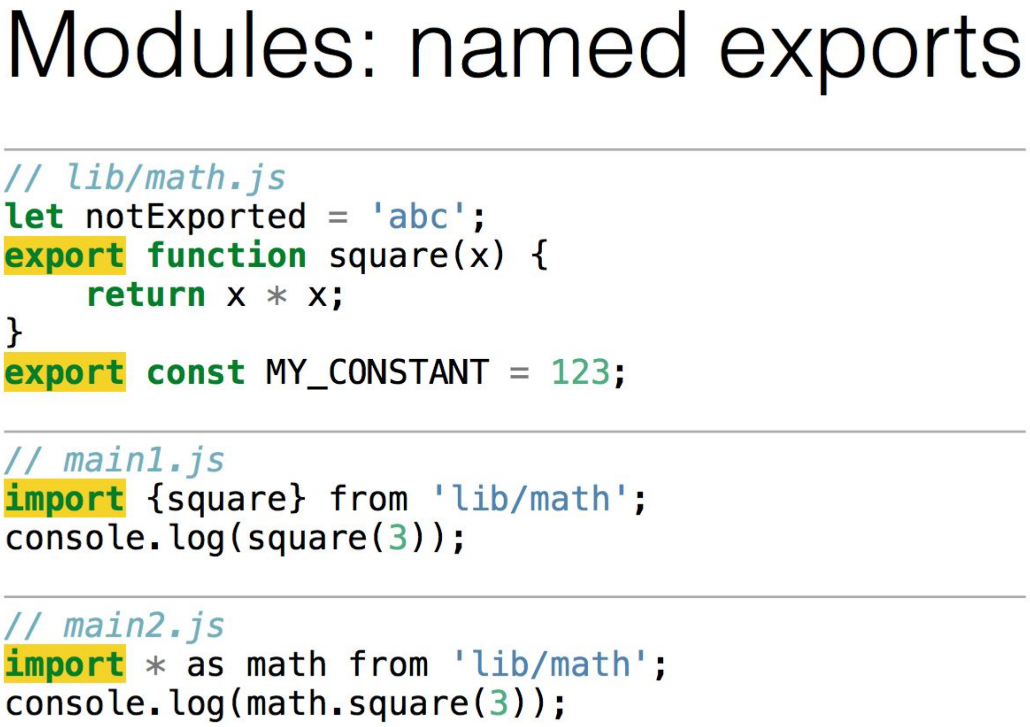
ECMAScript 6



Multiple return values

```
function findElement(arr, predicate) {
    for (let index=0; index < arr.length; index++) {
        let element = arr[index];
        if (predicate(element)) {
            return { element, index };
                   // same as { element: element, index: index }
        }
    }
    return { element: undefined, index: -1 };
}
let a = [7, 8, 6];

let {element, index} = findElement(a, x => x % 2 === 0);
    // element = 8, index = 1
let {index, element} = findElement(···); // order doesn't matter

let {element} = findElement(···);
let {index} = findElement(···);
```

ECMAScript 6

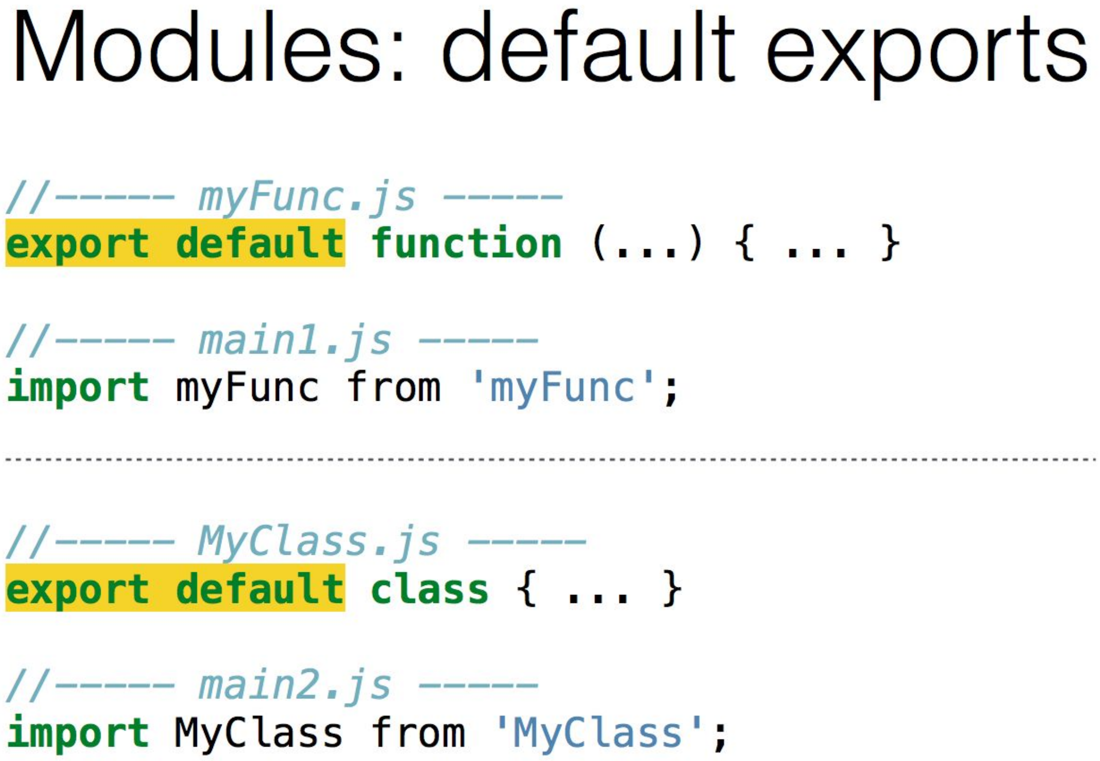


Modules: named exports

```
// lib/math.js
let notExported = 'abc';
export function square(x) {
    return x * x;
}
export const MY_CONSTANT = 123;
```

```
// main1.js
import {square} from 'lib/math';
console.log(square(3));
```

```
// main2.js
import * as math from 'lib/math';
console.log(math.square(3));
```

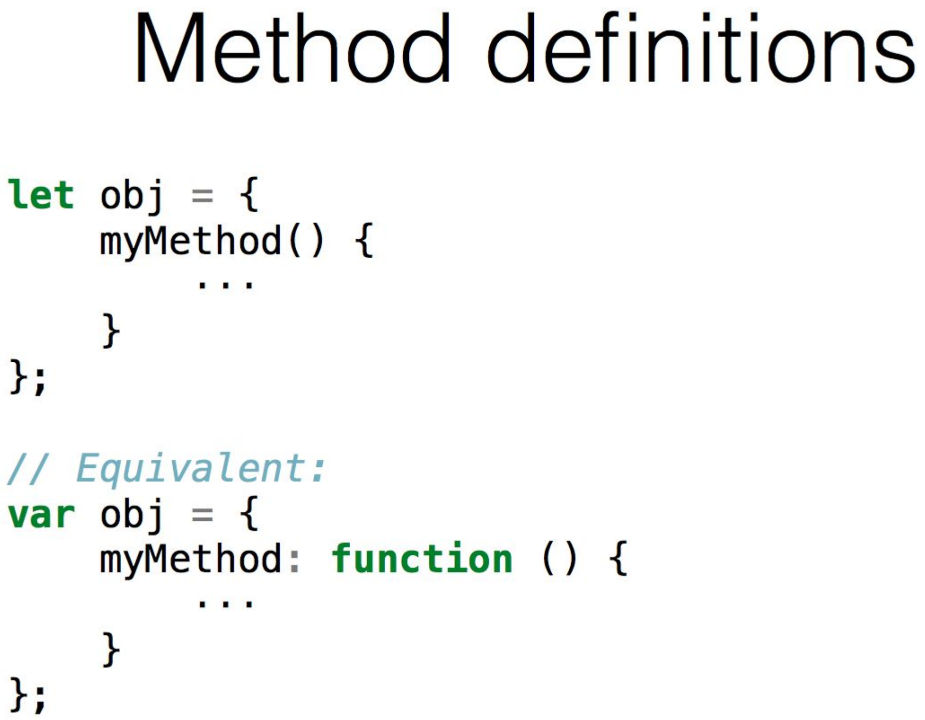


Modules: default exports

```
//------ myFunc.js ------
export default function (...) { ... }

//------ main1.js ------
import myFunc from 'myFunc';
```

```
//------ MyClass.js ------
export default class { ... }

//------ main2.js ------
import MyClass from 'MyClass';
```

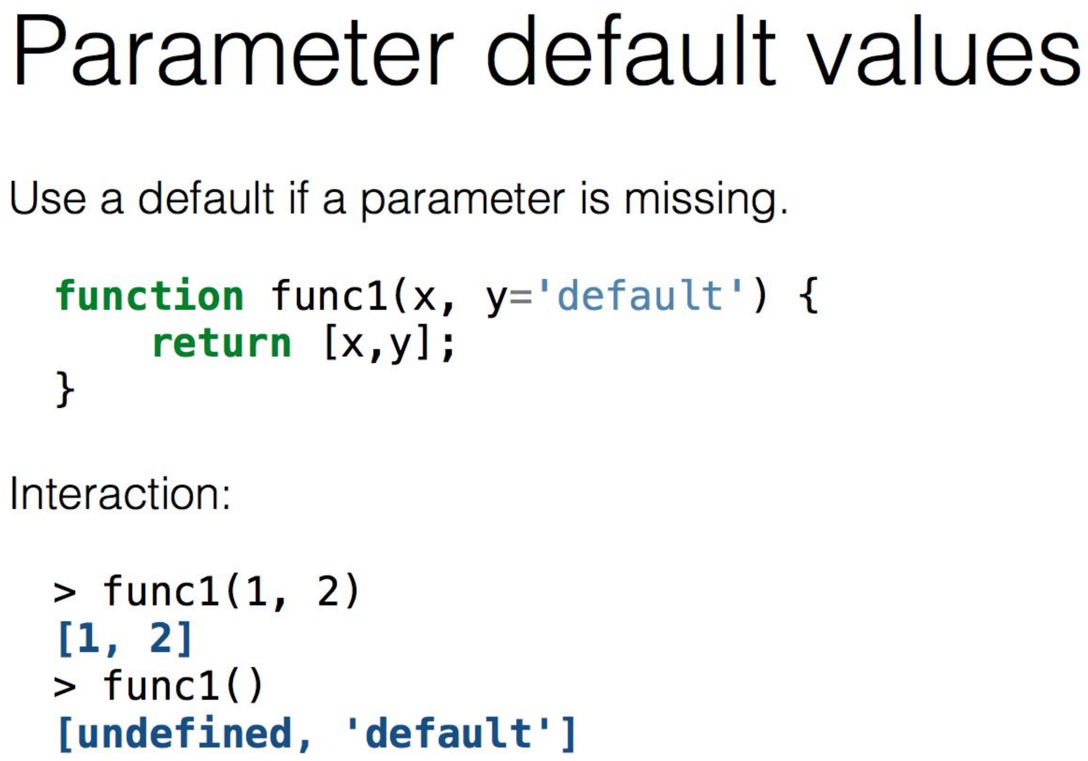
ECMAScript 6



Method definitions

```
let obj = {
    myMethod() {
        ...
    }
};

// Equivalent:
var obj = {
    myMethod: function () {
        ...
    }
};
```

ECMAScript 6



Parameter default values

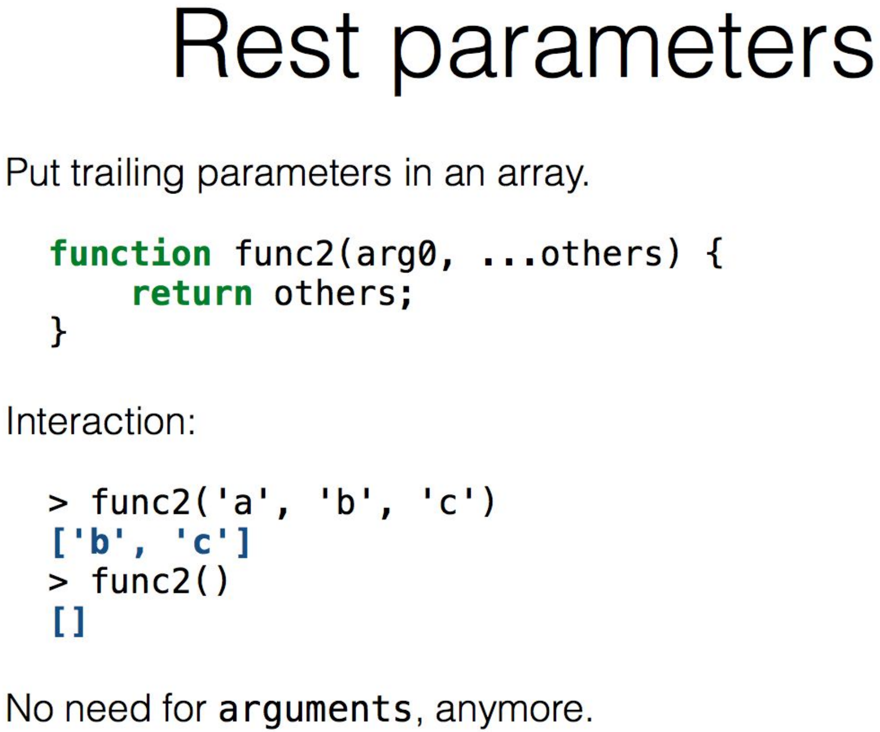
Use a default if a parameter is missing.

```
function func1(x, y='default') {
    return [x,y];
}
```

Interaction:

```
> func1(1, 2)
[1, 2]
> func1()
[undefined, 'default']
```

ECMAScript 6



Rest parameters

Put trailing parameters in an array.

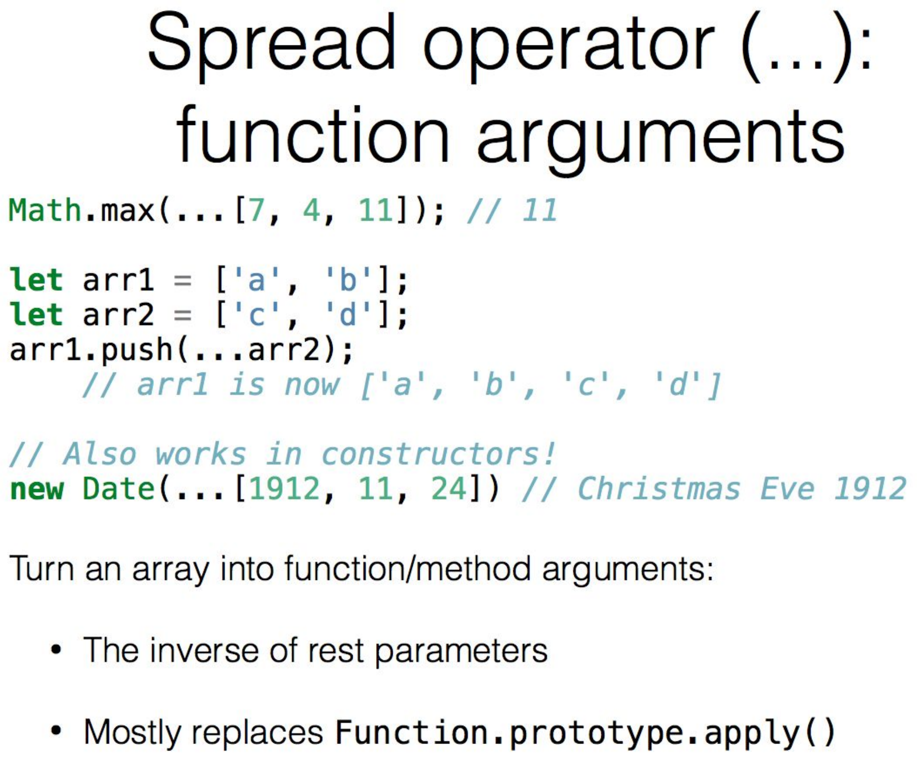
```
function func2(arg0, ...others) {
    return others;
}
```

Interaction:

```
> func2('a', 'b', 'c')
['b', 'c']
> func2()
[]
```

No need for `arguments`, anymore.

ECMAScript 6



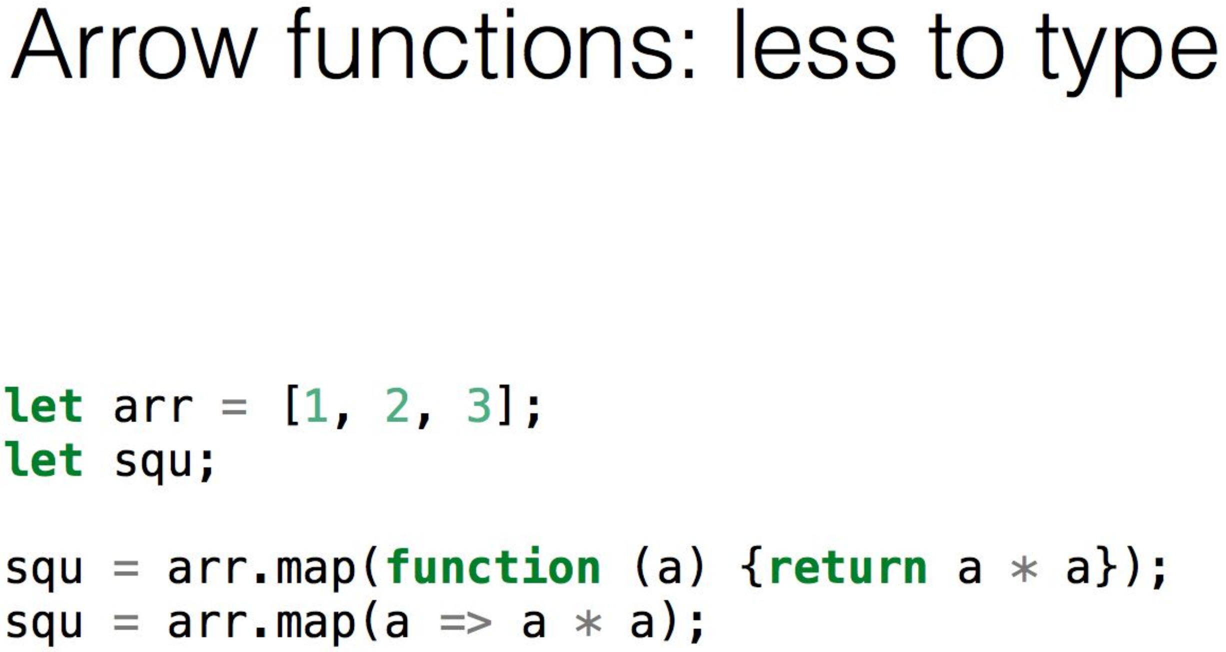
Spread operator (...): function arguments

```
Math.max(...[7, 4, 11]); // 11

let arr1 = ['a', 'b'];
let arr2 = ['c', 'd'];
arr1.push(...arr2);
    // arr1 is now ['a', 'b', 'c', 'd']

// Also works in constructors!
new Date(...[1912, 11, 24]) // Christmas Eve 1912
```

Turn an array into function/method arguments:

- The inverse of rest parameters
- Mostly replaces `Function.prototype.apply()`

ECMAScript 6



Arrow functions: less to type

```
let arr = [1, 2, 3];
let squ;

squ = arr.map(function (a) {return a * a});
squ = arr.map(a => a * a);
```

TypeScript

As JavaScript applications have became larger and more common, pressure has increased for better developer tooling:

- Automated refactors
- Go-to-definition, find references
- Type safety, null safety

Today many projects use TypeScript. It gets transformed into JavaScript and is a strict superset of ES6. If you use TypeScript, you get to use all the features of ES6.

For lab tomorrow

- Your project group will meet with me to present your project idea and project slug. If you can’t make it to section with your project group, they can present for you. **You must get your project team and project idea approved**.
- You’ll join the `scalableinternetservices` GitHub org and push your team’s starter project to a blank repository.
- ⚠️ Making progress during lab is factored into your final grade! Milestones set at the beginning of each lab and should be completed by the end of lab and presented to the teaching staff.