Interactive Web Programming

1st semester of 2021

Murilo Camargos
(**murilo.filho@fgv.br**)

Heavily based on **Victoria Kirst** slides

Today's schedule

Our last lecture!

- Next steps: General advice
- Important ideas we didn't cover
- Libraries and frameworks
- Final advice

The #1 question about web programming

How do I stay up to date??

There are so many changing technologies…

- How do I know which ones to use?
- How do I learn about new libraries?
- Won't everything I learn be obsolete in 2 months?!?!

(Note: not a real article)

Q: How do I stay up to date??

A: This is the wrong question to ask.

Staying "up to date" is not that important.

Tech doesn't fundamentally change very often or very fast.

(weak fashion analogy ends here)

New tech: Helpful, not necessary

Most new web technology makes your life easier **but is not necessary.**

Examples:

- `const` and `let`
- `async` / `await`
- CSS variables, etc

Everything* you want to do can already be done with the web technology available not just today, but 15 years ago.

*You know, within reason

Fundamentals don't change

Tech *doesn't* change that quickly

- Much of Facebook is still written in PHP
- Most of Google is written in Java and C++
- You will not (and should not) totally rewrite your codebase every year
- Tons of parallel problems, patterns, etc across tech

Personal anecdote:

- I learned web programming 10 years ago then didn't use it professionally for the last ~6-7 years
- It took me 1 week to "catch up" on new stuff… because they were all solutions to old problems

The real question to ask

Also: **Many new libraries are bad.**

- Literally anyone can post a library on npm - there is no
- Most libraries on npm are therefore garbage
- Even popular libraries can be poorly written.

So the real question to ask:

- **How do I distinguish good web technology from bad web technology?**

Choosing good tools

Either:

- You have enough knowledge to be able to decide whether a tool or technology is beneficial

Choosing good tools

Roll over image to zoom in

Wilson

Wilson Tour Slam Lite Tennis Racquet

19 customer reviews | 5 answered questions

List Price: $30.00

Price: $19.99 Prime

You Save: $10.01 (33%)

In Stock.

Want it Tuesday, June 6? Order within **11 hrs 34 mins** and choose On checkout. Details

Ships from and sold by Amazon.com. Gift-wrap available.

Color: **Blue/Black**

Size:

Grip Size: 4 3/8

- Volcanic frame technology for power and stability
- 110" head, 27.25" length
- 11.5 oz strung weight(326g)
- 4 3/8 inch grip size

Or:

- You don't have enough knowledge to tell the difference
- Therefore it doesn't really matter
- And you should choose the simplest / cheapest thing that other people say is good

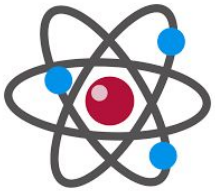
Choosing good tools

If you keep getting better at tennis, someday you'll look back at your first racquet and think

- "OMG how was I using this terrible racquet" or,
- "Lol I had a $300 racquet and had no idea how to use it", or
- "Huh, that cheap one was actually pretty good"

But the ability to choose good tools takes expertise and experience that you don't have as a beginner.

Choosing good tools



And sometimes there's just a bit of personal preference

(weak tennis analogy ends here)

General advice

Focus on becoming a good engineer.

- Learn how to build good software in any language, frontend, backend, web, iOS, Android, data pipelines, anything.

Work as a full-time software engineer for N years **with other (good) people.**

- Even after 1 year working full-time, your engineering skills will improve immensely

This is how you will develop and hone your own technical judgement.

General advice

Don't be afraid or intimidated by new technology.

When you confront a new web thing, like a library or framework, one of two things will happen:

1. You will be excited by it, and you will want to use it.
2. You will not be excited by it, and you can safely ignore it.

Simpler is always better.

- ALWAYS delete code if you can
- ALWAYS remove a library if you can
- ALWAYS remove a framework if you can

Helpful CS classes

Recommended CS classes:

- Databases
- As many systems classes as you can take
  - CS 107 and 110
  - Networking
  - Operating Systems
- Compilers
- Programming languages

With that context...

What next?

This is a fundamentals course, meaning we covered the critical stuff, but we just scratched the surface.

We'll do a quick tour of the following:

- Topics you really-really-really ought to know
- Topics you might find handy
- Opinions on libraries
- Final suggestions

Topics you really-really-really ought to know

Testing

Missed topic: Robustness

The code we wrote in this class is **extremely fragile**:

- No tests
  - Especially dangerous on backend… we can accidentally delete the entire database with one line of code.
- No type checking
- No backups for databases
- Doesn't work on older browsers
- Etc

Spot the difference

What's the difference between the following code snippets?

```
// A
const query = { _id: ObjectID(id) };
userData.deleteOne(query);

// B
const query = { };
userData.deleteOne(query);
```

Spot the difference

What's the difference between the following code snippets?

```
// A: Deletes the specified document (or
// does nothing if not found).
const query = { _id: ObjectID(id) };
userData.deleteOne(query);

// B: Deletes the first document.
const query = { };
userData.deleteOne(query);
```

MUST: Server-side Testing

If you write production server code, **you must write tests.**

Q: What are tests?

- A test is a type of software that verifies the code you wrote works
- Tests help you:
  - Verify everything works before you launch your product
  - Catch regressions as you modify code

MUST: Server-side Testing

You should probably write tests for all your code, but server is especially important

Check out:

- MochaJS: A popular JavaScript test framework that works on frontend and backend (NodeJS) code
- Jest: Facebook's JS test framework
- Chai: Helper library to write easier-to-read tests
  - Used with Mocha, Jest, etc

Warning: Setting up tests for the first time always sucks.

Module bundlers

Missed topic: Bundling

Our frontend JavaScript includes look ridiculous:

```
<script src="js/player-bullet.js" defer></script>
<script src="js/player-ship.js" defer></script>
<script src="js/space-game.js" defer></script>
<script src="js/text-screen.js" defer></script>
<script src="js/app.js" defer></script>
<script src="js/main.js" defer></script>
```

- Have to define JavaScript includes in HTML
- Have to remember the include order
  - Can't specify dependencies, e.g. `PlayerBullet` must be included before `PlayerShip`, but is independent of TextScreen

SHOULD: JavaScript modules

We want a module system for our frontend JavaScript.

- Recall: NodeJS has a module system using `require()`

Tooling option: **Module bundlers**

- Browserify
- WebPack

Not ready yet: A native JavaScript option

- ES6 modules and `import`

Browserify

Lets you use `require()` in frontend JavaScript, exactly like how it would work in NodeJS.

- You can write your own modules and require them
- You can download NodeJS modules and require them

Browserify works by **transpiling** the code written using `require()` statements into code that can be executed in the browser.

Before: Raw JS

hello-lib.js

```
function printHello() {
  console.log('hello world');
}
```

main.js

```
printHello();
```

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title></title>
    <link rel="stylesheet" href="css/style.css">
    <script src="js/hello-lib.js" defer></script>
    <script src="js/main.js" defer></script>
  </head>
```

After: browserified

hello-lib.js

```
function printHello() {
  console.log('hello world');
}

module.exports = printHello;
```

main.js

```
const printHello = require('./hello-lib.js');

printHello();
```

After: browserified

hello-lib.js

```
function printHello() {
  console.log('hello world');
}

module.exports = printHello;
```

main.js

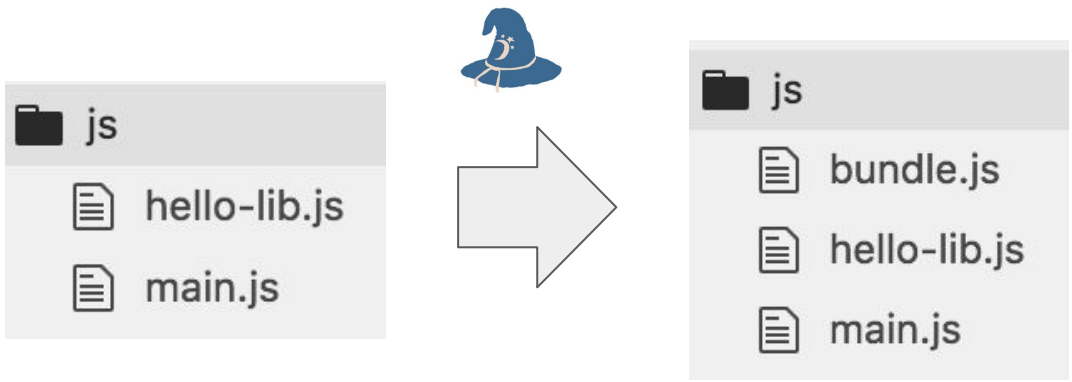
```
const printHello = require('./hello-lib.js');

printHello();
```

This code **no longer runs** natively in the browser, since browser don't support require()ing npm modules.

Instead, you must run the `browserify` command:

- This will "**transpile**" the code into JavaScript the the browser can run.
- It will be "**bundled**" into a single script.js file.

Browserify




```
$ sudo npm install -g browserify
$ browserify js/* -o js/bundle.js
```

- This will create a file called `bundle.js`, which contains the code for `main.js` and the `hello-lib.js` file that it requires.
- You need to include **`bundle.js`** in your HTML file

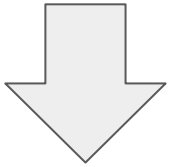
After: browserified

hello-lib.js

```
function printHello() {
  console.log('hello world');
}

module.exports = printHello;
```

main.js

```
const printHello = require('./hello-lib.js');

printHello();
```



```
browserify js/* -o js/bundle.js
```

bundle.js

```
(function e(t,n,r){function s(o,u){if(!r
require=="function"&&require;if(!u&&a)re
module '"+o+"'");throw f.code="MODULE_NO
l=n[o]={exports:{}};t[o][0].call(l.expor
s(n?n:e)},l,l.exports,e,t,n,r)}return nl
o=0;o<r.length;o++)s(r[o]);return s})({1
function printHello() {
  console.log('hello world');
```

After: browserified

bundle.js

```
(function e(t,n,r){function s(o,u){if(!r
require=="function"&&require;if(!u&&a)re
module '"+o+"'");throw f.code="MODULE_NO
l=n[o]={exports:{}};t[o][0].call(l.expor
s(n?n:e)},l,l.exports,e,t,n,r)}return n
o=0;o<r.length;o++)s(r[o]);return s})({1
function printHello() {
  console.log('hello world');
```

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title></title>
    <link rel="stylesheet" href="css/style.css">
    <script src="js/bundle.js" defer></script>
  </head>
  <body>
  </body>
</html>
```

Browserify recap

Lets you use `require()` in frontend JavaScript

- You can write your own modules and require them
- You can download NodeJS modules and require them

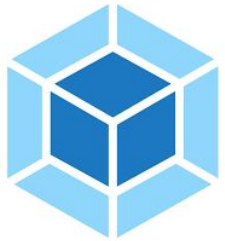
You must **transpile** your JavaScript code in order to run it

- Use the `browserify` command to generate `bundle.js`
- Include the single `bundle.js` file in your HTML

This idea of **transpiling JavaScript** is very common for modern JavaScript tools and libraries!

See also: WebPack and import

WebPack: A more sophisticated JS module bundler



- Newer than Browserify
- More complicated than Browserify
- Can do more than Browserify

Not ready yet: A native JavaScript option

- ES6 modules and `import`
- **Keep an eye out for this! (CanIUse)**

Older browser support

Older browsers?

In CS193X, we used JavaScript features that worked on the latest version of each major browser.

But sometimes you need to support older browsers.

What do you do?

- Don't use the new stuff until it's ready? But when will that be?
- Write multiple versions of your code? But that's time-consuming and annoying
- Write polyfill fallback code? Also super annoying

BabelJS

Solution:**BabelJS**

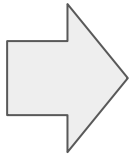
- Babel is a JavaScript compiler for the latest features of EcmaScript, including ES6+
  - If the browser supports ES6 natively, babel does nothing
  - If the browser does not support ES6 natively, babel provides a polyfill

Use BabelJS so that you can:

- Write code with the latest features in JavaScript
- Support older browsers without having to rewrite anything

Compiling with Babel

```
const x = [1, 2, 3];
foo([...x]);
```



```
var x = [1, 2, 3];
foo([].concat(x));
```

ES6 code

JavaScript that works on older browser

Babel [can be used] with Browserify, WebPack, etc:

```
$ browserify script.js -t babelify -o
bundle.js
```

Use Babel!

Type checking

Missed topic: Type checking

JavaScript is loosely typed, meaning you do not declare the data types of variables.

- Sometimes loose typing a great thing, e.g. when you are starting a project from scratch, prototyping, etc.

- But loose typing gets to be a pain as your code base grows.

Type checking

There are ways to essentially add **type checking** to JS:

- TypeScript: A different programming that is a superset of JavaScript. Write TypeScript code and **transpile** it to raw JavaScript.
- Flow: A static type checker for JavaScript. Write annotated JavaScript code and **transpile** it to raw JavaScript.
- Closure Compiler: An early bundler, code minimizer, and static type checker for JavaScript. Type definitions are done in comments and doesn't require transpiling.

TypeScript (2012)

- TypeScript is a **programming language** by Microsoft
- It is a superset of JavaScript that includes static typing.
- Browsers can only execute JavaScript, so you must **transpile** TypeScript to JavaScript

TypeScript

```
function Greeter(greeting: string) {
    this.greeting = greeting;
}
```

JavaScript

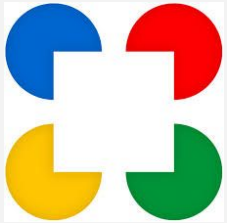
```
function Greeter(greeting) {
    this.greeting = greeting;
}
```

Flow (2014)

- Flow is a static type checker by Facebook.
- It is not a full programming language, but it involves a adding a combination of non-standard annotations and comments to your JavaScript.
- Browsers can only execute JavaScript, so you must **transpile** Flow-annotated code to JavaScript

```
// @flow
function square(n: number): number {
  return n * n;
}

square("2"); // Error!
```

Closure compiler (2009)



- Closure Compiler is a command-line tool by Google
- Transforms valid JavaScript into more efficient valid JavaScript.
- Type information (closure annotations) is specified in comments

```
/** @define {boolean} */
var ENABLE_DEBUG = true;

/** @define {boolean} */
goog.userAgent.ASSUME_IE = false;
```

Accessibility

Missed topic: Accessible tech

Technology should be accessible to **everyone**, regardless of their abilities or disabilities.

- Accessibility: design of products, devices, services, or environments for people who experience disabilities

The web is designed to be accessible, if you use it correctly. For example:

- Using `<h1>Heading</h1>` instead of `<div class="heading">Heading</div>` will help a screenreader create an audio outline for the page, since a visually impaired person may not be able to skim

Making tech accessible

Resources for accessibility:

- MDN accessibility
  - ARIA: Accessible Rich Internet Applications
- Google accessibility
- Teach Access / Tutorial
- Udacity course
- Accessibility dev tools extension

What next?

This is a fundamentals course, meaning we covered the critical stuff, but we just scratched the surface.

We'll do a quick tour of the following:

- ~~Topics you really-really-really ought to know~~
- **Topics you might find handy**
- Libraries and frameworks
- Final suggestions

Topics you might find handy

Misc web topics

A few other topics that might be useful for you:

<canvas>

- Allows you to draw graphics in a <canvas> tag
- Uses more traditional, lower level graphics commands
- 3d support with WebGL
- Simple demo; complex demo
- Canonical examples: Games; complex visualizations

WebSockets / Socket.io

- Used for server -> client messages
- Canonical examples: Chat client; gaming; anything that has live updating

Misc web topics

CSS grid layout

- The final missing piece for CSS layout!
- Not quite ready yet, but should be within the next year

Progressive web apps

- An alternative to server-side rendering, single-page-app, and isometric web apps:
  - Design an "app shell" that loads first
  - Use Service Workers to cache content
- Complex, but huge potential benefits

Publishing tools

Publishing static web pages

Domain name registration:

- Reserves a custom URL: myawesomesite.com
- But doesn't usually include web hosting; all you own is the name.

Web hosting:

- Provides a location on the internet to upload files
- Usually with some crummy URL, like http://bucket.s3-website-us-west-2.amazonaws.com/

Domain name registration and web hosting are sometimes provided by the same company, but not always.

Publishing static web pages

You can register your own domain name through many companies:

- Google Domains: Only domain name registration
- Amazon S3: Only web hosting
- Dreamhost: Domain name **and** web hosting options
- GoDaddy: Domain name **and** web hosting options

Domain name registration is usually ~$12/year
Web hosting is usually ~$10/month

- Amazon S3 is **significantly** cheaper (virtually free for low-traffic websites) but more complicated to set up

Publishing server-side code

If you want to host both a frontend and a backend, you need a web host that allows you to configure a server.

There are an immense number of options, with different levels of configuration. Here are some:

- Heroku: Super easy to use, but offers less control. Also a lot more expensive.
- AWS: Cheap, lots of options, but more complicated
- Google Cloud: Basically the Target brand of AWS: Cheaper than AWS; as complex as AWS; fewer products than AWS

What next?

CS193X is a fundamentals course, meaning we covered the critical stuff, but we just scratched the surface.

We'll do a quick tour of the following:

- ~~Topics you really-really-really ought to know~~
- ~~Topics you might find handy~~
- **Libraries and frameworks**
- Final suggestions

Libraries and frameworks

Web libraries and frameworks

A JavaScript library:

- Code that is written by someone who is not you
- Code that you import and call from your code
- Great examples of our course: ExpressJS

A web framework:

- A way of writing and deploying web applications
- Usually involves a combination of command-line tools and libraries
- Bigger than a library
- We didn't use a framework in this class

Some web frameworks

Libraries:

- jQuery

Frameworks:

- AngularJS
- Backbone.js
- Bootstrap
- Ember.js
- ReactJS
- Vue.js
- Flask (backend)
- Ruby on Rails (backend)
- Django (backend)

Using a framework

In this class, we wrote frontends using raw, modern JavaScript.

Q: Should I use a framework or write apps using raw JavaScript?

Using a framework

In this class, we wrote frontends using raw, modern JavaScript.

Q: Should I use a framework or write apps using raw JavaScript?
A: Depends on what it is.

- Small apps don't need a framework.

Now that you know how to write apps without a framework, I suggest you learn how to use a framework.

Suggestion: Learn a framework!

In this class, we learned how to write frontends without a framework.

- Sometimes that's the right choice
- Sometimes a framework is the right choice

Suggestion: Your next step after this class should be to learn a web framework.

Q: How do I learn how to use a framework?

A: Pick one and try.

Just try it out

General advice:

- Go to the official website
- Use the official website's tutorials
  - Like, actually follow along; don't just skim the docs
- Then **build a small app of your own** on the framework
  - The only way to "learn" a framework is to build something using it, beyond just following a tutorial
  - Suggestion: Choose something you could build in 24 hours using the tech you already know

Most well-known frameworks have tutorials, excellent documentation, strong developer communities, etc.

Q: Which framework do I pick??

A: Doesn't really matter right now.

(If you've never used a framework, using *literally any of them* will be educational.)

Roll over image to zoom in

Wilson

Wilson Tour Slam Lite Tennis Racquet

19 customer reviews | 5 answered questions

List Price: $30.00

Price: $19.99 Prime

You Save: $10.01 (33%)

In Stock.

Want it Tuesday, June 6? Order within **11 hrs 34 mins** and choose On checkout. Details

Ships from and sold by Amazon.com. Gift-wrap available.

Color: **Blue/Black**

Size:

Grip Size: 4 3/8

- Volcanic frame technology for power and stability
- 110" head, 27.25" length
- 11.5 oz strung weight(326g)
- 4 3/8 inch grip size

Victoria's take.

Also my own.

jQuery: **Don't use**

jQuery was built in 11 years ago when the web was in a much worse state

But now most of jQuery's features have native JS equivalents

- `document.querySelector`
- `classList`
- ES6 classes
- CSS animations
- etc.

jQuery: **Don't use**

jQuery also provides cross-browser compatibility, but you should prefer babel for that.

Suggestion:

- Only use jQuery if you're forced to, i.e. if you're working in a code base that already uses jQuery and you can't change it.

Bootstrap: **Don't use**

Bootstrap is a *really heavyweight*, not-very-flexible set of default CSS styles and JavaScript components

Bootstrap is nice for what the name implies: bootstrapping a pretty, generic-looking website

However, Bootstrap is often used as a crutch by people who don't want to learn CSS.

Bootstrap: **Don't use**

Suggestion:

- Use Bootstrap if you want your page to look literally like this
- Otherwise, avoid Bootstrap:
  - It is really hard to do anything that's not this
  - It is *really* hard to debug
- Learn and use raw CSS:
  - Use CSS flexbox
  - Use CSS grid when it's ready
- Hire a designer to make your website look nice

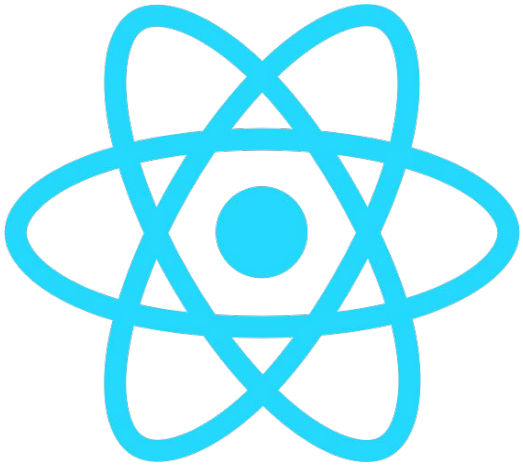
ReactJS: **Good** with some issues

ReactJS is a fairly lightweight frontend framework.

Uses JSX, which mixes JavaScript and HTML-looking syntax:

```
const element = <h1>Hello, world!</h1>;
```

```
class ShoppingList extends React.Component {
  render() {
    return (
      <div className="shopping-list">
        <h1>Shopping List for {this.props.name}</h1>
        <ul>
          <li>Instagram</li>
          <li>WhatsApp</li>
          <li>Oculus</li>
        </ul>
      </div>
    );
  }
}
```



ReactJS: **Good** with some issues

Overall take:

- ReactJS is very good!
- But there are some major open issues
  - E.g.: How to deal with global state (Redux is a very popular library to use in conjunction with ReactJS, but it counteracts React's state model)

Suggestion:

- Learn ReactJS and make your own judgement
- Use create-react-app
- If you decide to use Redux, watch the A+ video series and don't try to read the indecipherable documentation

Recap

MUST-dos:

- Learn server-side testing, if you are ever going to launch a server

SHOULD-dos:

- Use browserify or WebPack for JS bundling
- Use babeljs with browserify or WebPack for older browser support

SHOULD-try:

- Pick a web framework and learn it

Recap

DON'T-dos:

- **Don't use jQuery**
- **Don't use Bootstrap**
- Don't unnecessarily complicate your tech stack
- Don't be afraid of new libraries/tools/frameworks.
    - If they are good, they make your life easier, not harder!

On the horizon

Keep an eye out for:

- Public / private fields in ES6 classes
- ES6 Modules / `import`
- Custom elements
    - More broadly: Web components

These are not ready yet, but they will be soon.

Watch the discussions around web app architecture:

- Isometric / universal websites
- Progressive web apps
- Progressive loading

One last rant

Everyone's 2nd favorite question for the web:

BASARAT Follow

That TypeScript Guy http://www.ba

Apr 27, 2016 · 3 min read

Forget Angular & Ember, React Has Already Won the Client-Side War

TypeScript won

> *I love all the people (great develo*
> *mentioned in this post 🌹. That s*

Is golang the future?

Is Golang dead?

Is jQuery Still Relevant?

R.I.P. Ruby on Rails. Thanks for everything.

Published on January 13, 2016

Is Java Dead? No! Here's Why…

▲ Ask HN: Is Python dying?

Is Django already a dying technology?

BASARAT Follow

That TypeScript Guy http://www.ba

Apr 27, 2016 · 3 min read

TypeScript won

> *I love all the people (great develo*
> *mentioned in this post 🌹. That s*

Forget Angular & Ember, React Has Already Won the Client-Side War

Is golang the future?

Is Golang dead?

Q: Which library/tool/language/platform is going to **win**?????

Is jQuery Still Relevant?

R.I.P. Ruby on Rails. Thanks for everything.

Published on January 13, 2016

Is Java Dead? No! Here's Why…

▲ Ask HN: Is Python dying?

Is Django already a dying technology?

A: Wrong question.

CS is not a competitive sport.

Not everything is a dominance hierarchy.
Not everything is a dominance hierarchy.
Not everything is a dominance hierarchy.
Not everything is a dominance hierarchy.
Not everything is a dominance hierarchy.

- JavaScript libraries are not at war.
- Multiple things can be good.
- Learning **any good library** is valuable, even if it's not in its absolute height of popularity.
  - A great way to improve your software engineering skills: Studying other people's designs

Better questions

- Does this library solve the problems that I care about?
- Is this library production-ready?
  - Does it have prominent clients?
  - Does it work at scale?
  - Has it worked out most of its bugs?
- Is this library under active development?
  - Does it *need* work?
- How easy is it to find documentation/StackOverflow results for this library?
  - Does it *need* documentation/help pages?

Final advice

Staying up to date

With all the caveats aside:

Q: "How do you stay up to date on web stuff?"

Staying up to date

With all the caveats aside:

Q: "How do you stay up to date on web stuff?"

A: Read the internet! But tread carefully:

Garbage piles

Do not trust:

- Comment sections of Reddit
- Comment sections of Hacker News
- Comment sections of any website
- Medium articles by randos

In my experience, these are far too often full of posturing, gross misinformation, terrible opinions based on little-to-no facts, etc.

Hit-and-miss

Usually works, but sometimes poor style / not best practice

- StackOverflow answers
- W3C schools
- Programming YouTube videos

Better opinions than most, but sometimes still trash

- Quora answers

Good web resources

Reliable websites

- Google Web Fundamentals
- Mozilla hacks

Prominent JavaScript accounts/people on Twitter, e.g.

- NodeJS, Sarah Drasner, Suz Hinton, Sebastian Markbåge, Henry Zhu, Dan Abramov, David Walsh

Official documentation:

- HTML WHATWG spec / HTML W3C spec
- EcmaScript status / spec

Write code

The only way to get better at web programming is to write lots and lots of code.

- Become a software engineer
- Work with software engineers who are better than you
- Write simple side projects to learn new tech
  - **Suggestion:** Choose a project you know you could finish in 1 day - 1 week

You can do it!

Thank you!