FED

Front-End Development

It's just...

HTML + CSS + JavaScript


ENDUPDATE WEEKENDUPDATE WEEKENDU


UNDER CONSTRUCTION

How do you get that HTML, CSS and Javascript to the user in a way that is...

→ Fast

→ Beautiful

→ Usable

→ Accessible

→ Maintainable

How do you get that HTML, CSS and Javascript to the user in a way that is...

→ Localisable

→ Automated

→ Tested

My day job

Build things for 700k+ users
and 100s of devs

My day job

Write code, review others' code

I love deleting code

My day job

Mentor and support other devs

My day job

Make decisions with:
* Designers
* Other developers
* QAs
* Product Owners / Managers

Appealing

→ Easy to learn, hard to master

→ Fast feedback

→ Tangible/visual

→ *Fun*

Unappealing

→ The user controls browser, device, dimensions*

→ Framework choices & pace-of-change*

→ Available technologies*

→ Changing platform*

→ Browser variations

* Depending on your PoV

Learn to use your browser's dev tools

Chrome's are probably the best
Very useful skill to have

HTML

Hypertext Markup Language

A set of elements for describing documents and relationships between them.

```
<!DOCTYPE html>
<html lang="en-NZ">
<head>
  <meta charset="utf-8">
  <title>Summer of Tech</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <h1>
    <a href="https://summeroftech.co.nz/">Summer of Tech</a>
  </h1>
  <p>Hi</p>
</body>
</html>
```

Learning HTML

Check out a reference like MDN (not W3Schools!)

Following the spec is pretty straightforward

`span` and `div` are generic elements to use when there's not a specific element to use

(so they tend to be used a lot by JavaScript libraries when creating custom controls)

HTML5

#marketing

New HTML, CSS & JavaScript features that make the web a better platform for writing applications.

CSS

CSS is easy

... and that makes it really hard

web (css) !== print

```
body {
  font-family: Arial, Helvetica, sans-serif;
  background-color: #fff;
  color: #333;
}

p {
  font-style: italic;
}

#thebest {
  text-decoration: underline;
  text-transform: uppercase;
}

.snazzy {
  color: pink !important;
  font-weight: bold;
}
```

CSS Specificity

Ugh

`!important > inline > id > class > element`

```
<p id="thebest" class="snazzy" style="padding-left: 20px;">
  What is even happening?
</p>
```

CSS pre + post processors

→ Sass

→ LESS

→ Rework

→ PostCSS

→ cssnext

Variables

```
$primary-color: #333;

body {
  color: $primary-color;
}
```

Imports

```
@import "components/button";
@import "components/modal";

.foo {
  color: fuschia;
}
```

Nesting

```
.calendar {
  border: 1px solid #555;

  .calendar-day {
    border: 1px solid #555;
    color: blue;
    font-size: 14px;
  }
}
```

Prefixes - Autoprefixer

```
.foo {
  background: linear-gradient(to bottom right, #333, #0c0);
}
```

=>

```
.foo {
  background: -webkit-linear-gradient(top left, #333, #0c0);
  background: linear-gradient(to bottom right, #333, #0c0);
}
```

And when the unprefixed form is universally supported autoprefixer will stop generating that rule

A few years ago

We thought CSS pre-processors were the hotness, but...

```scss
.nesting {
  .stuff {
    .is-easy {
      color: black;

      &.dangerous {
        color: blue;
      }
    }
  }
}
```

=>

```scss
.nesting .stuff .is-easy {
  color: black;
}

.nesting .stuff .is-easy.dangerous {
  color: blue;
}
```

When writing CSS for big projects, it's the stuff outside the braces that counts...

-- Harry Roberts, http://csswizardry.com

Naming Conventions

→ SMACSS

→ BEM

SMACSS

```
body {}                          /* Base styles */

.layout-sidebar {}               /* .layout-{name} */

.modal {}                        /* .{moduleName} */

.modal--header {}                /* .{moduleName}--{subComponent} */

.button {}                       /* .{moduleName} */

.button-is-disabled {}           /* .{moduleName}-is-{stateName} */

.button-default {}               /* .{moduleName}-{subModule} */

.button-primary {}               /* .{moduleName}-{subModule} */
```

In HTML

```
<div class="modal">
  <!-- .modal--header is a subcomponent -->
  <div class="modal--header">Modal</div>
  <div class="modal--body">
    ...
  </div>
  <div class="modal--footer">
    <!-- .button-primary is a submodule -->
    <button type="button" class="button button-primary">Save</button>
    <button type="button" class="button button-cancel">Cancel</button>
  </div>
</div>
```

TL;DR

By following strong conventions and minimising specificity, naming conventions keep your CSS more maintainable.

CSSLint is also a thing.

Browsers

→ Still some differences

→ ~~IE~~ Edge has been catching up

→ User agent strings are easy to fake

→ Feature detection (e.g. Modernizr)

It used to be *much much* worse.

Atwood's Law

Any application that can be written in JavaScript, will eventually be written in JavaScript.

-- Jeff Atwood, http://blog.codinghorror.com/

JavaScript

By many measures the world's most popular programming language.

→ Created in 10 days

→ Released in 1995

→ Renaissance started around 2006-2007

→ Rise of Node.js from 2010-2011

→ In the browser allows you to change HTML and

JavaScript looks like Java and C#, but it's very different to those languages

#marketing

More like Scheme than Java

No block scope (until ES6's `const` and `let`)

Dynamic types and type coercion

`this` is weird

Understanding JavaScript Helps More

JavaScript is flexible so it's easy to write Ruby-style, Java-style, whatever-style code.

Don't do that. Or if you really want to consider a language that compiles to JavaScript.

Compile to JavaScript Languages

TypeScript / Scala.js / ClojureScript / Dart / etc

JavaScript

ES3

```
function square (n) {
  return n * n;
}

var input = [1, 2, 3, 4, 5];
var result = [];
var i;

for (i = 0; i < input.length; i++) {
  result.push(square(input[i]));
}
```

ES5

```
var input = [1, 2, 3, 4, 5];

var result = input.map(function (n) {
  return n * n;
});
```

ES6

```
const input = [1, 2, 3, 4, 5];

const result = input.map(n => n * n);
```

TypeScript

```
var input:number[] = [1, 2, 3, 4, 5];

var result = input.map((n: number) => {
  return n * n;
});
```

Document Object Model (DOM)

```
// find an element in the document
const el = document.getElementById('content');

// listen to events on that element
el.addEventListener('click', function (e) {
  e.preventDefault();
  // modify that element somehow (e.g. changing styling)
  el.classList.add('link-is-clicked');
});
```

Accessibility

Involves adding `role` and `aria` attributes to elements to describe them

Accessibility

```
<div>
    An alert <button>X</button>
</div>
```

Accessibility

```
<div role="alert">
    An alert <button aria-label="Close">X</button>
</div>
```

Let's Talk About Async

or

How the heck does Node.js scale when JavaScript is single threaded!?!?!

It's used *in production* by PayPal, eBay, LinkedIn, Walmart, Yahoo!, Uber, Trello, New York Times, News Corp and many others.

Reading a file

In Java:

```
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;

public static void main(String[] args) throws IOException {
    String data = new String(Files.readAllBytes(Paths.get("data.txt")));
    // do stuff with data here
}
```

Reading a file

In Node.js:

```
const fs = require('fs');

fs.readFile('data.txt', function(err, data) {
    if(err) {
      // handle error here
      return;
    }
    // do stuff with data here
});

console.log('Do more stuff');
```

Reading a file

In Node.js with promises:

```
const readFile = require('bluebird').promisify(require('fs').readFile);

readFile('data.txt').then(function(data) {
  // do stuff with data here
}).catch(function(err) {
  // handle error here
});

console.log('Do more stuff');
```

JavaScript Libraries & Frameworks

→ jQuery for DOM normalisation

→ Underscore for utility fns

→ React

→ Backbone

→ Angular / Angular 2

→ Ember

Massive ecosystem, with lots of options.

React Example

```
import React from 'react';

function Button(props) {
  return (
    <button type={props.type}>{props.children}</button>
  );
}

Button.propTypes = {
  type: React.propTypes.string;
};

Button.defaultProps = {
  type: 'button'
};
```

Javascript Package Managers

NPM

→ Nested dependencies (no conflicts)

→ Favors dynamic resolution

→ Great with node; use with care for browser bundles

Javascript Package Managers

Bower

→ Flattened dependencies (you resolve conflicts)

→ Favors static resolution

→ Good for bundling front-end code for browsers

The secret to building large apps is never build large apps. Break your applications into small pieces. Then, assemble those testable, bite-sized pieces into your big application.

-- Justin Meyer, JavaScriptMVC

HTTP

Hypertext Transfer Protocol

1991 - v 0.9
1996 - v 1.0
1999 - v 1.1
2015 - v 2.0

Request / Response

Clients make *requests* to servers

Servers provide *responses* to clients

Request Methods

GET / POST / DELETE / PUT / PATCH / HEAD / OPTIONS

Learn & use status codes in APIs

```
200 OK
301 Moved Permanently
304 Not Modified
400 Bad Request
404 Not Found
418 I'm a teapot (RFC 2324)
500 Internal Server Error
```

A GET Request

```
GET / HTTP/1.1
Host: www.summeroftech.co.nz
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
User-Agent: user-agent:Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/50.0.2661.102 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-NZ,en;q=0.8,en-US;q=0.6
```

And Its Reponse

```
HTTP/1.1 200 OK
Content-Encoding: gzip
Content-Length:5285
Content-Type: text/html; charset=UTF-8
Date: Tue, 17 May 2016 04:14:53 GMT
Host-Header: 192fc2e7e50945beb8231a492d6a8024
Link: <https://summeroftech.co.nz/wp-json/>; rel="https://api.w.org/", <https://wp.me/P6yDxo-yD>; rel=shortlink
Server: nginx
Vary: Accept-Encoding
X-Pingback: https://www.summeroftech.co.nz/xmlrpc.php
X-Proxy-Cache: MISS

<!doctype html>
...
```

XHR / AJAX, JSON & APIs

```
const button = document.getElementById('sendButton');

sendButton.addEventListener('click', function (e) {
  const request = new XMLHttpRequest();
  request.open('GET', '/my/url');

  request.onload = function() {
    if (this.status >= 200 && this.status < 400) {
      // Success!
      var data = JSON.parse(this.response);
    } else {
      // We reached our target server, but it returned an error

    }
  };

  request.onerror = function() {
    // There was a connection error of some sort
  };

  request.send();
});
```

HTTP Servers

Apache / IIS / Nginx / Node / Tomcat / etc

Modern Front-Ends

Backends decoupled from front-ends
Deploy independently
Communicate via defined contracts (APIs)
Back End for Front-Enders (proxies for transformation or rendering)

FE Ops

Transpiling, combining, linting, testing, measuring, etc. all that code needs tools.

Node Tooling

Node = evented I/O (reading/writing to/from file system, std in/out, etc.).

Build little CLI tools is really easy ...

Node runs fast! Starting up Node is way cheaper than starting up a JVM.

Tools like Grunt, Gulp and Yeoman make task automation and project scaffolding easier.

Web vs. Native

Today's hot drama (still!?)

Web means the best reach, best device support, but you have to work harder.

Loads of "native" apps use web views for parts of their app.

It's not a binary choice and web technology makes it easy to try something.

The Future

→ Single Page Applications*

→ Bigger, more ambitious apps

→ Smaller, more fine-grained modules

→ Mainstreaming of functional programming

→ Isomorphic apps

→ Cloud infrastructure / platform-as-a-service

* we're sort of there now, but there is still some resistance

Functional programming

Immutability (we do it with Strings already)

Referential transparency

Testability

Predictability

Community

→ Meetups (AkJS, Auckland CSS)

→ Conferences

→ Github

→ npm

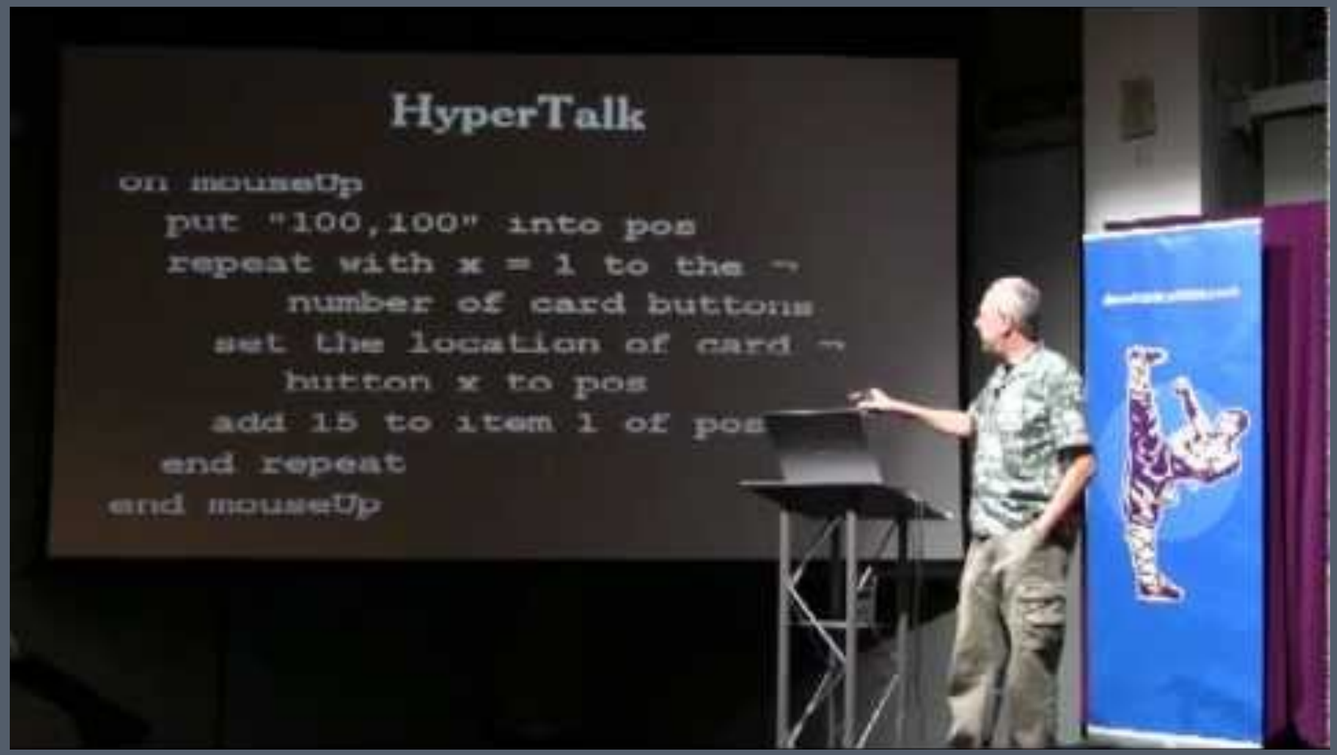
→ Social Media

→ Podcasts

Resources

Watch

Crockford on JavaScript

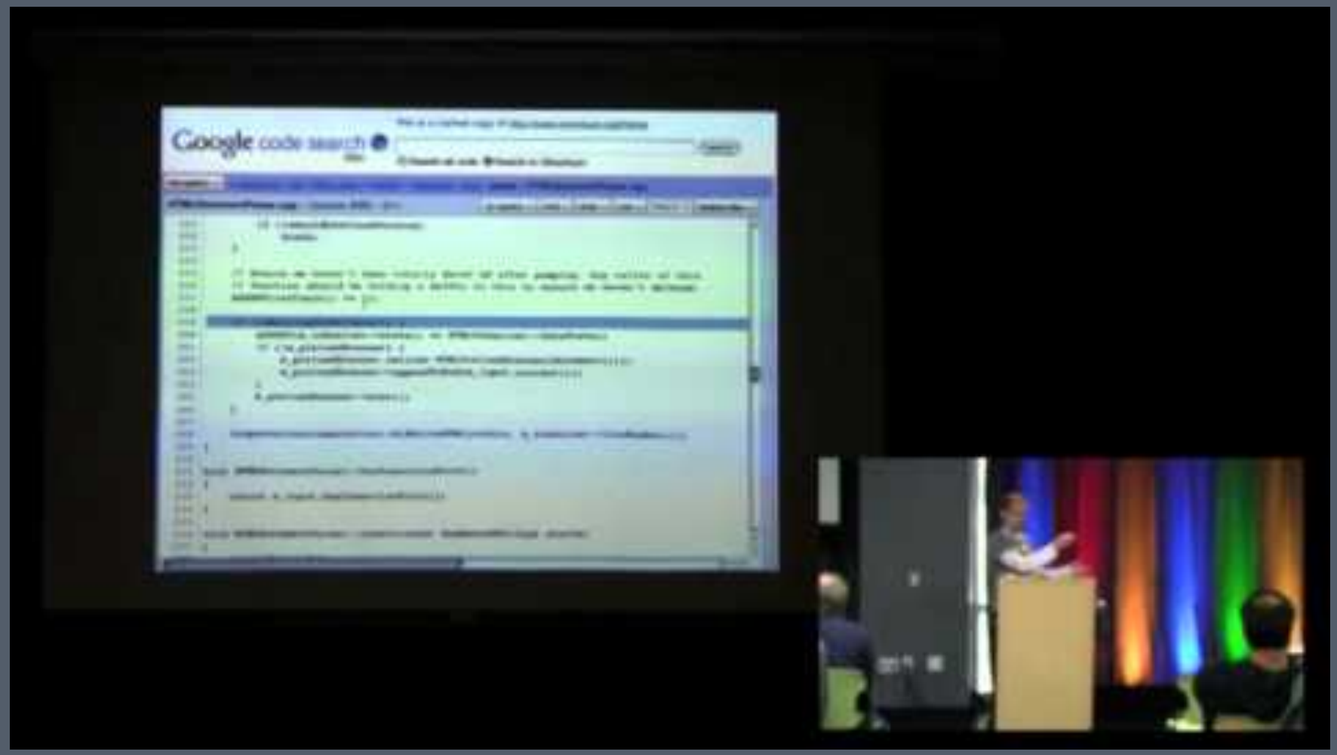


Watch

What the Heck is the Event Loop Anyway?

Watch

How Web Browsers Work



Read

How Browsers Work
More meaningful CSS
CSS Specificity Rules

The Inception Rule (why nesting is bad)

Read

A Baseline for Front-End (JS) Developers

Listen

→ JavaScript Jabber - http://devchat.tv/js-jabber/

Bookmark These

→ http://devdocs.io

→ http://caniuse.com

Play With These

→ CodePen - http://codepen.io

→ ES6Fiddle - http://www.es6fiddle.net

→ Babel - https://babeljs.io

→ ESLint - http://eslint.org

→ http://nodeschool.io

Tweet

@ivanandsickle

QnA