# Intro to JavaScript

**Michael Chang**

**Spring 2020**

# Plan for today

**JavaScript background**

Context, brief history

**JS in the browser**

Including scripts, the console

**Detour: JS language features**

Expressions, variables, types, functions

**Back to JS in the browser**

The DOM: manipulating HTML elements

# JavaScript history

**In 1995, only static web pages**

- Various efforts to integrate dynamic content into browsers

**Netscape hired Brendan Eich**

- Created JavaScript in 10 days
- So named purely for marketing--no relation to Java

**Late '90s and early '00s browser war**

- Microsoft reverse engineered JS interpreter (JScript)
- Netscaope creates JS standard (ECMAScript)
- IE dominates market
- MS doesn't participate in standards process

# JavaScript (mostly) happy ending

**Late '00s**

- Firefox and Chrome gain market share
- 2008: Companies collaborate on ECMAScript 5
- IE slowly vanishes into the abyss

**2015: ECMAScript 6 (ES6 or ES2015)**

- Huge language update (classes, modules, async)
- Tons of cleanup of previous design choices
- Backwards compatible, broad browser support

# JavaScript (mostly) happy ending

**Current state**

- Overwhelmingly dominant (basically the only) browser language, few large-scale alternatives
- New ECMA standard every year, but only small changes
- Still some compatibility challenges with using very new features, but workarounds exist

**In my opinion, it's a good language now.**

# Our approach to JS

**Challenge: too many ways to do things**

- Backwards compatibility, significant change over time
- Many workarounds for older browsers
- Many bad habits still floating around

**We will focus on modern standards and best practices**

- Assume browser updates < 2 years ago
- Use many ES2015+ features
- Completely ignore some parts of the language

# Our approach to JS

**Not all the stuff we don't cover is bad**

**Best practices**

- Generally agreed on by the community
- Avoid older techniques with modern equivalents

**Recommendations**

- Approach we've found less confusing
- Other ways may be fine
- No JS experience? No problem, stick to these and you'll be all set

# JavaScript overview

**Interpreted language**

- But browsers are getting very good at running it quickly

**Native execution in browser**

- No (exposed) underlying "assembly" language

**Dynamically typed (like Python)**

- No declared type, but values have types
- Variables can change types

**Object-oriented**

- Everything is an object, including primitives and functions
- ES2015 added classes (syntactic sugar around awkward older syntax)

# JavaScript syntax

**C/C++/Java-like**

- Braces for blocks
- `//` and `/* ... */` comments
- The usual operators (`=`, `+`, `-`, `*`, `/`, `%`)
  - Except == and != are weird (come back to this)

**Semicolons are actually optional**

- Recommendation: Use semicolons

# JavaScript syntax example

```
let str = "Hello";
let x = 42;
if (x < 193) {
  str += ", world!";
  x = x * 2;
} else {
  str += ", CS193X!";
}
console.log(str);
console.log(x);
```

# JavaScript primitive types

**`Boolean`: true or false**

**`Number`: both integer and floating point**

- All numbers are double
- See `Math` for some useful functions on numbers

**`String`: immutable**

- Single or double quotes equivalent, be consistent
- `length` property (not method)

**`null`: "intentionally absent" value**

**`undefined`: no value**

- `return;` (or no return statement)
- Variable without assigned value

# JavaScript variables

## Several ways to define variables

`let x = 42;` -- define and initialize x

`let x;` -- define x, no initial value (undefined)

`const x = 42;` -- value of x can't change (must assign value)

## Also there's var

Scoping rules are unintuitive

Best practice: don't use var

## Sometimes you can just refer to a variable

Automatically becomes a global variable

Best practice: don't do this

# JavaScript conditionals

**All values are "truthy" or "falsy"**

E.g. `if (x) { ... }`

Falsy: 0, "", NaN, null, undefined

Truthy: everything else (incl. empty arrays, "0")

# JavaScript conditionals

**All values are "truthy" or "falsy"**

- E.g. `if (x) { ... }`
- Falsy: 0, "", NaN, null, undefined
- Truthy: everything else (incl. empty arrays, "0")

**Equality with == (and !=)**

- Implicitly converts operands to match type
- `false == 0`, `0 == ""`, `1 == "1"`, `false == "0"`
- Best practice: Don't use == and !=
  - Exception: `x == null` tests if `x` is `null` or `undefined`

# JavaScript conditionals

**All values are "truthy" or "falsy"**

E.g. `if (x) { ... }`

Falsy: 0, "", NaN, null, undefined

Truthy: everything else (incl. empty arrays, "0")

**Strict equality with === (and !==)**

Does what you want, `false` if types mismatch

For later: "shallow" equality for objects (e.g. only `true` if both operands refer to same exact object)

# JavaScript functions

## Declaration

```
const name = (arg1, arg2) => {
  /* ... */
  return ...;
}
```

Functions are just a kind of variable

- The implications of this won't be clear until later

## The traditional way: `function` keyword

Seen most often, some are moving to arrow

It's fine, but has some quirks

Recommendation: we won't show code that uses this, but you can if you want

# JS in the browser

## `<script>` element

- Can contain JavaScript code
  - Not recommended
- Use src attribute to include file
- Must have closing tag

```
<script src="myscript.js"></script>
```

# JS in the browser

## `<script>` element

- Can contain JavaScript code
  - Not recommended
- Use src attribute to include file
- Must have closing tag

## Issue: page read from top to bottom

- If <script> in head, will be executed before elements exist
- Solution: add "defer" attribute

```
<script src="myscript.js" defer></script>
```

# Document Object Model (DOM)

**JS can access the web page using the DOM**

- Each element is a `Node`
- Can walk the tree and add/change/remove elements

**Builtin variables**

- `window`: info/control the browser window
  - The "global object"; you can jam your global vars here
- `document`: methods for accessing the document
  - document.head, document.body

# Document Object Model (DOM)

**HTML attributes accessed as JS properties**

`src, href, id`

**document.querySelector(selector)**

Find element by CSS selector

Returns first matching elem

```
let elem = document.querySelector("#seal");
elem.src = "images/stanford.png";
```

# Changing style in JS

**Node's style prop is an object**

- Can set CSS properties on the element
- Hyphens changed to camelCase
- Caveat: only reads per-element (inline) styles, not styles from classes/ids/etc.
- Values are strings

```
elem.style.backgroundColor = "#0080ff";
/* Use computed value (from class/id/etc.) */
elem.style.display = "";
/* Have to define units */
elem.style.marginLeft = "100px";
```

# Changing style in JS

## Access CSS classes through `classList`

Aside: `class` is a JS keyword

`classList` is a `DOMTokenList`

Exact type doesn't matter, but see link for methods

```
if (elem.classList.contains("foo")) {
  elem.classList.add("bar");
} else {
  elem.classList.remove("baz");
}
```

# Summary

**Intro to JS in the browser**

- Syntax, types, functions
- Manipulating elements with the DOM

**Next time**

- Interactors (`<input>` and `<button>`)
- Events