Plan for today

Recap: HTML, servers

JavaScript background
  Context, brief history

JS language features
  Expressions, variables, types, functions

JS in the browser
  Including scripts, the console
Hypertext Markup Language

Not a programming language

Overall structure:

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>...</title>
</head>
<body>
  ...
</body>
</html>
head

Information about the page, not displayed
<meta>: page metadata
   Author, search engine keywords, character set
<title>: title shown in tab/taskbar
<script> and <link>: refer to other files
Page contents

A few elements

- `<h1>`, `<h2>`, ..., `<h6>`: headings
- `<p>`: paragraph
- `<a href="...">`: link

Will cover more tags as we use them

Will post reference on website with some common tags
JavaScript history

In 1995, only static web pages
Various efforts to integrate dynamic content into browsers

Netscape
Created JavaScript in 10 days
Named purely for marketing--no relation to Java

Late '90s and early '00s browser war
Microsoft reverse engineered JS interpreter (JScript)
Netscape created JS standard (ECMAScript)
IE dominated market
MS didn'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)
Tons of cleanup of previous design choices
Backwards compatible, broad browser support
Current state

Dominant browser language, no alternatives that run directly in browser
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 < 2 years old (all browsers auto-update now)
- 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
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")

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: stick to arrow functions
JS in the browser

<script> element
- Can contain JavaScript code inside the node
  - Best practice: don't do that
- Use src attribute to include file
- Must have closing tag

[type="module"]
- Allows importing other modules
- Avoids some quirks with global variables, errors accessing HTML

Recommendation: we'll use this for all our scripts

<script type="module" src="myscript.js"></script>
Summary

Intro to JS in the browser
  Syntax, types, functions

Before next time
  Set up environment (assign0)
  Come to office hours, check out Ed

Next time
  More JS syntax (arrays, objects, classes)
  Interacting with the web page (DOM)