The State of JavaScript

CIS 1962 (Spring 2026)
April 16th, 2026

Penn Engineering

Lesson Plan

5	JavaScript: Further Study Topics
15	JavaScript: What's Next?

Logistics

Next week: Presentation Day!

- Think of it as "Milestone 2" of your project. This means that by this point, you should have a working demo to show off!
- Your ~10 minute presentation should include:
  - A description of what your site is about
  - An overview of the technology stack used
  - A demo of the project
  - A reflection on your work on this project and future directions

Review Activity

https://edstem.org/us/courses/91614/lessons/164119/slides/965210

JavaScript: Further Study Topics

What more is there to learn about JavaScript?

Congrats, you made it!

You made it to the end of this mini-course!

Over the course of this semester, you've learned the basics of working with JavaScript to build web applications and have started building sophisticated full-stack web applications of your own!

What's next?

What's left of JavaScript?

We've covered JS up from basic syntax and infrastructure, to working with HTML/CSS/JS, to frameworks and full-stack applications.

If you're interested in pursuing JS further, you can look at:

- **Further Frameworks** (Angular, Vue, Preact)
- **WebAssembly** (JS + other languages like Rust, C++, Go)
- **Edge Computing** (Cloudflare workers, Vercel edge functions)
- **Other Backend Frameworks** (Fastify, NestJS, GraphQL)
- **Authorization/Authentication** (OAuth, Auth0, Firebase Auth)

More Frameworks

While React and Next.js are quite popular, there are many frameworks out there that companies use, either alongside React or in place of it.

- **Angular**: Google, Microsoft Office web apps
  - Strong TS support, dependency injection, built-in testing, routing, forms, and states
- **Vue**: Alibaba, Xiaomi, Baidu, Nintendo, Grammarly
  - Easy learning curve, great docs, elegant reactivity
- **Preact**: Uber, Lyft, Pinterest
  - React-like API, smaller bundle for performance

WebAssembly (Wasm)

Web Assembly (Wasm) is a low-level format that lets code from languages like C, C++, Rust, and Go run alongside JS.

The fast speed afforded by these languages lets you do a lot more compute-heavy code on the web, and make use of existing code and libraries from the languages you bring over.

The code is compiled into a `.wasm` file that runs using JS.

Examples of WebAssembly

- Conway's Game of Life
- lichess.org
- Figma
- AutoCAD Web
- Commander Keen Web Port
- Diablo Web Port

Edge Computing

Edge Computing is a distributed computing model where data served at a physically closer "edge" server rather than a centralized server. This results on lower latency, and relies on multiple distributed server locations.

This has allowed certain platforms like Cloudflare and Vercel to deliver static content (in Content Delivery Networks or CDNs) and actual code (Edge functions/runtimes) on edge servers.

Edge Computing Examples

Cloudflare Workers

- Full-stack deployment platform using edge computing
- Low latency due to having 300+ data centers for physically close access

Vercel Serverless Edge Functions

- Vercel handles server management and delivers content from the closest servers
- Can configure content tailored to different regions when specified in settings (default is USA East, can pay to use different regions)

Other Backend Technologies

oRPC: Helps link clients and servers with end-to-end type safety and OpenAPI standards

GraphQL/Apollo Server: Data querying language and the library to create a GraphQL server. Helps manage endpoints for getting and mutating data.

Fastify: Fast, efficient alternative to Express with Native TS support and great docs

NestJS: Comprehensive backend architecture that uses Express/Fastify under the hood, with support for WebSockets, MicroServices, GraphQL, and more

Auth and the Other Auth

OAuth: standard protocol for authentication, allows third party apps (like Google, Github and Facebook) to be integrated as authorization and authentication mechanisms

Auth0: A cloud-based authentication and authorization platform that uses protocols like OAuth into apps. Can handle complex auth flows like SSO (Single sign-on, like Penn SSO) and social logins.

JavaScript: What's Next?

What's in store for the future of JavaScript?

New JavaScript Versions

JavaScript comes out with a new edition each year, developed TC39 (Technical Comittee 39). The process goes as follows:

1. A proposal is made
2. A proposal goes through 5 stages, from 0-4, and only Stage 4 proposals make it into the language (before March)
3. Features are published in June

Browsers often implement features before the official release (features at Stage 3 or earlier)

Proposal Steps

0. Strawman/Strawperson: Informal idea

1. Proposal: Formal problem and rough solution, with a specification on GitHub

- Example: do expressions

2. Draft: Initial specification is created for semantics, syntax, and API

- Example: throw expressions, pipeline operator

3. Candidate: A mostly complete feature that just requires real-world feedback. Browsers can start implementing these features.

- Example: import text

4. Finished: Complete proposal, added to ECMAScript

JavaScript: The Current State

Upcoming Version: ECMAScript 2026 (ES17)

- Temporal API
- Math.sumPrecise()
- using and await using keyword

Past Features:

- Set Operations (2025)
- Iterator Helpers (2025)

Problems with Dates

Parsing Ambiguity

```
const d1 = new Date("2024-01-01");
const d2 = new Date("2024-01-01T00:00");
console.log(d1.toString()); // UTC
console.log(d2.toString()); // Local time
```

Mutability

```
const original = new Date();
const mutated = original;
mutated.setHours(mutated.getHours() + 1);

console.log("Original:", original); // changed!
console.log("Mutated:", mutated);
```

Month Indexing

```
const jan = new Date(2024, 0, 1); // January
const feb = new Date(2024, 1, 1); // February
```

Date Arithmetic

```
const dateMath = new Date(2024, 0, 31);
dateMath.setMonth(dateMath.getMonth() + 1);

console.log(dateMath.toString()); // rolls over to
March 2nd unexpectedly
```

Temporal API

Temporal API (Proposal): The new standard for working with dates and times

```
// Explicit, immutable, timezone-aware
const zdt = Temporal.ZonedDateTime.from(
     "2024-01-01T10:00:00[America/New_York]"
);

// Safe date math (no mutation)
const later = zdt.add({ hours: 2 });

// Clear separation of concepts
const dateOnly = Temporal.PlainDate.from("2024-01-01");

console.log(zdt.toString()); // original unchanged
console.log(later.toString()); // new value
console.log(dateOnly.toString());
```

Math.sumPrecise()

Floating point precision has been an issue in JS:

```
const nums = [0.1, 0.2, 0.3];
const sum = nums.reduce((a, b) => a + b, 0);
console.log(sum) // 0.6000000000000001
```

Math.sumPrecise() has a slower, but more accurate summing algorithm:

```
const nums = [0.1, 0.2, 0.3];
const sum = nums.reduce((a, b) => a + b, 0);

const precise = Math.sumPrecise(nums);
console.log(precise) // 0.6
```

using and await using keyword

`using` and `await using` are ways to have **explicit resource management** in JS.

They are used like const, but can run disposal logic upon leaving scope, which can be defined in a `[Symbol.dispose]()` method.

```
const file = {
     name: "data.txt",
     read() {
          console.log("reading", this.name);
     },
     [Symbol.dispose]() {
          console.log("closing", this.name);
     }
};

using f = file;
f.read();
```

Set Operations (ES 2025)

`union()`: All elements from both sets

`intersection()`: Only common elements

`difference()`: Elements in A but NOT in B

`symmetricDifference()`: Elements in A or B but NOT both

`isSubsetOf()`: A is fully contained in B

`isSupersetOf()`: A contains all elements of B

`isDisjointFrom()`: No shared elements at all

Generator Functions

Generator functions (ES6) are functions that can pause execution (yield) and resume later. They are defined with function*.

They can be used to create infinite sequences and streams, and allow consumption on demand (memory efficiency)

```
function* counter() {
     let i = 0;
     while (true) {
          yield i++;
     }
}
```

```
const c = counter();

console.log(c.next().value); // 0
console.log(c.next().value); // 1
console.log(c.next().value); // 2
```

Iterator Helpers (ES2024/5)

Generator functions and other iterators had the issue of needing to be converted to arrays before transformation, like using the `map` method.

Iterator helpers solved this by introducing array-like methods:

- `map`
- `filter`
- `take (first n items)`
- `drop (skip first n)`
- `reduce`
- `toArray`

```
function* count() {
     let i = 0;
     while (true) yield i++;
}

const result = count()
     .filter(x => x % 2 === 0)
     .take(5)
     .toArray();

console.log(result); // [0, 2, 4, 6, 8]
```

JavaScript Trends

- Dominance of TypeScript in development workflows
- The rise of WebAssembly alongside JS
- Standardization of generative AI in development
- Baked-in security features to reduce XSS and CSRF
- Low-cost serverless computing
- Framework agnostic developers

The End, and Thank You

Thank you all for a wonderful mini-course experience!

See you at the final project presentations :)