Modules, fetch, and APIs

Michael Chang

Spring 2020

Announcements

assign2 due tomorrow

Project proposals in

- Will return feedback this weekend

assign1 graded

- Released on Paperless after lecture

Plan for today

Recap of classes

Modules, import, and export

- Splitting code into files
- Aside: debugging with modules

Reading external data

- fetch function
- Promises
- APIs and JSON

At the end, assign1 recap

- Common issues, style tips

JS aside: rest/spread operators

MDN reference

Functions can take variable number of args

```
const myFn = (a, b, ...rest) => { ... }
```

`rest` is an Array of the args after `b`

Can call function with variable args

```
const myFn2 = (a, b, c) => { ... }
let arr = [20, 30];
myFn2(10, ...arr);
```

In myFn2, `a=10, b=20, c=30`

Recommendation: don't use the `arguments` keyword

DOM aside: removing listeners

`elem.removeEventListener(type, fn)`

Remove previously added listener

Careful: fn must be **the same object** that was added

```
const add = () => {
  const handler = (event) => { ... };
  elem.addEventListener("click", handler);
};
const remove = () => {
  const handler = (event) => { ... };
  elem.removeEventListener("click", handler);
};
```

Won't work even if `handler` has the same code

Splitting JS into multiple files

One approach

- Create multiple files
- Add multiple `<script>` tags
- Each script has access to the global variables/functions of the others

This works fine, but...

- No file scope (everything at top level is global)
- Hard to keep track of what scripts use what vars
- Need to update the HTML file if you add a new script

New approach: use modules

- Not quite "best practice" yet, but getting there

JavaScript modules

Introduces in ES6

Has taken a little while to gain browser support

MDN reference

Script syntax

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

Note: no need for defer

Module exports

Module's variables not global

Not automatically accessible from other module

Need to be exported

`export`

```
export let exportedVar = ...;
export const exportedFn = () => { ... }
```

These are "named exports" (see next slide)

`export default`

```
export default /* function, class, etc. */;
```

This is the "default export"

Importing from module

import

```
import Binky from "./Binky.js";
```

Gets the default export from Binky.js, names it Binky

```
import { exportedFn } from "./Binky.js";
```

Gets a named export (name must match exactly)

```
import Binky, { exportedVar, exportedFn} from
"./Binky.js";
```

Combined syntax

Relative paths must start with "./"

Module issues

Debugging

- Since variables aren't global, can't access from console
- Solutions
  - Use the debugger to step/inspect
  - Use `console.log` + right-click "Store in global variable"
  - Assign to `window` object
  - (Of course, don't leave these in your final submissions)

Compatibility with non-modules

- Can't import a non-module
- Need to use global variables (e.g. through window) here

So far

Recap of classes

Modules, import, and export

- Splitting code into files
- Aside: debugging with modules

Reading external data

- fetch function
- Promises
- APIs and JSON

At the end, assign1 recap

- Common issues, style tips

External files

Recall: client/server

- Browser isn't reading directly from your hard drive
- Makes request to server, server returns files

Builtin handling of certain files

- HTML, images, CSS, JavaScript

What about other files?

- Data files, text
- More broadly: dynamic data sources ("Live feeds")

fetch API

`fetch(url[, options])`

Read contents from a URL (which could be relative)

Returns a `Promise` with the response

Detour: asynchronous programming

JavaScript is based on asynchronous, or event-driven, programming

We see this with event listeners and callbacks

Example (pseudocode)

```
main():
  when Add button clicked, call onAdd
  when Delete button clicked, call onDelete
  when checkbox changes, call onUpdate
```

Then main returns

Detour: asynchronous programming

Contrast this with synchronous program

Used in some languages/libraries

Example (pseudocode)

```
main():
  loop forever:
    wait for next event to happen
      if Add button clicked, call onAdd
      if Delete button clicked, call onDelete
      if checkbox changes, call onUpdate
      if Exit button clicked, return
```

main won't return until program exit

Detour: Promises

Promise: standard interface for handling asynchronous code

- Represents something that will happen later (or is happening in background)
- Once finished, the promise "settles"
- It can be in one of three states
  - pending: still waiting on result
  - fulfilled: has a result
  - rejected: error occurred

Detour: Promises

Cannot access result of Promise directly

Need to attach a callback

```
p.then(onFulfill[, onReject])
```

After p settles, call one of the callbacks according to its state

We'll see more on Promises later

fetch API

`fetch(url[, options])`

Read contents from a URL (which could be relative)

Returns a `Promise` with the response

`response.status`

Read the HTTP status code of the response

`response.text()`

`response.json()`

Interpret the response body

Returns a `Promise` with the data

fetch example

```
fetch("myfile.txt").then(response => {
  response.text().then(text => {
    console.log(text);
  });
});
```

We'll learn about a better syntax next time

Summary

Today

Intro to Promises and fetch

Next time

REST APIs, JSON, HTTP statuses