Modular JavaScript

September 29

Today's Goal: Split your code up into multiple files.
And work time.

The Old Way

Multiple JS files get linked to a single HTML page by way of multiple `<script>` tags.

Variables and constants in the global scope of one file are actually added to the global scope of the whole document, making them available in the other files as well.

Order matters, and globally scoped memory is required (both of which are generally considered “bad programming practice”)

The New Way - Modules

As of ES6 (2015), JS now has a feature called modules, which allows your code to be broken up into smaller pieces without having to worry about loading order or globally scoped variables at all.

Use `<script type=”module”>` to load a JS file that uses modules.

`export`

`export` the pieces of code that need to be used in other files, either by using the `export` keyword before any declaration, or by using the `export` keyword plus some curly brackets `{}` to export any previously declared things.

Examples:

```
export const year = 2022;
let month = ‘September’;
export function getDayOfMonth() {
  return 29;
}
export { month };
```

`import`

`import` the pieces of code that has been `export`ed from other files, by specifying the names of the `export`s in curly brackets `{}` and the name of the file which they are exported from in either single `‘` or double quotes `“`

Example:

```
import {
  year,
  month,
  getDayOfMonth
} from ‘./otherFile.js’;
```

`import as` and `export as`

You can change the name of the module's items as you are `export`ing them or `import`ing them by using the `as` keyword.

Examples:

```
let thisMonth = 'September';
export { thisMonth as month };
```

...

```
import { month as theMonth } from './otherFile.js';
```

`import * as`

When there are a lot of `exports` from one file, it can be tedious to write and take up way too much space in the file if you had to name every single item that was `exported` from the file you want to `import`.

`import * as` saves the day by letting you `import` everything from one file, and house it all under one parent object that you name yourself as the importer.

Example:

```
import * as DateStuff from ‘./otherFile.js’;
```

`export default`

There is (an attempt at) a coding convention to modularize your code all the way down to only `export`ing a single item per JS file.

This idea of a singular `export` is supported by the `export default` keywords, and allows the `import`er not to have to worry about `{}` or `as` or `*` or anything at all.

Example:

```
const year = 2022;
export default year;
```

...

```
import year from ‘./otherFile.js’;
```