COMPSCI 326 - Web Programming
JavaScript Modules, HTML, & the DOM

join on the Slack #q-and-a channel as well as Zoom
remember, you can ask questions of your teammates on your group Slack!
please **turn on your webcam** if you can
mute at all times when you aren't asking a question
(https://docs.google.com/document/d/1PROSgwoJqY1M8xV3r6qU6ESD2ERdwkHX4ujZnkO2JMM/edit?usp=sharing)

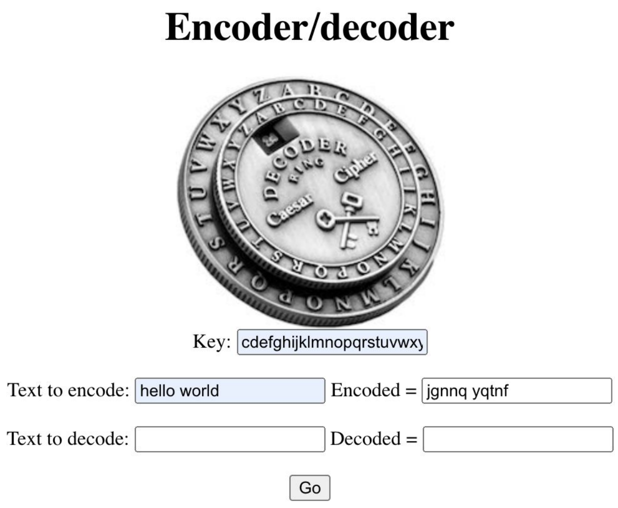
Background resources:

videos
JavaScript beginner tutorial 27 - forms

web sites
JavaScript modules - JavaScript | MDN
https://developer.mozilla.org/en-US/docs/Web/API/Document_object_model/Using_the_W3C_DOM_Level_1_Core
<script>: The Script element - HTML: Hypertext Markup Language
<form> - HTML: Hypertext Markup Language | MDN
<label> - HTML: Hypertext Markup Language | MDN
<input>: The Input (Form Input) element - HTML: Hypertext Markup Language
<input type="button"> - HTML: Hypertext Markup Language | MDN

Today: JavaScript Modules, HTML, and the DOM

Exercise today: make a web interface for the decoder functions - will be using HTML, scripts, and the DOM to do this



Encoder/decoder

Key: cdefghijklmnopqrstuvwxy

Text to encode: hello world Encoded = jgnnq yqtnf

Text to decode: Decoded =

Go

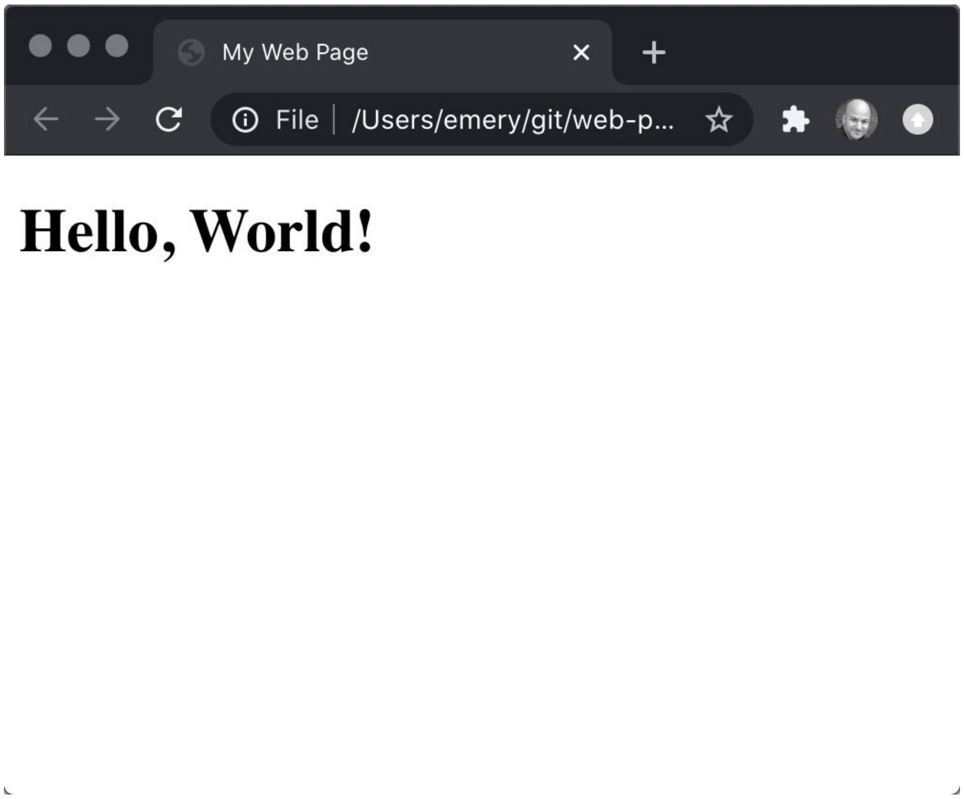
Modules

- JavaScript programs originally just snippets of code
    - "started off pretty small — most of its usage in the early days was to do isolated scripting tasks, providing a bit of interactivity to your web pages where needed, so large scripts were generally not needed."
    - Today, complete applications being run in browsers with JavaScript
    - JavaScript also used server-side (e.g., Node.js), desktop (e.g., Electron), mobile (e.g., React Native), database (e.g. Mongo), even embedded IoT (e.g. XS)!
- Don't want everything in one big .js file!
- Modules
    - Make it possible to have big JavaScript projects
    - Numerous implementations
        - CommonJS - uses `require()` and `module.exports`
            - Was initially adopted by Node.js as the module system for that environment, but needs special handling to work in browser
        - ES6 modules (newer) - uses `import` and `export` statements
            - "isomorphic" - same code works in the browser and in Node (as of version 12 with command-line flags, directly for versions >= 13)

        - doesn't work for IE
    - `import * as moduleName from "...";`
      `export const Foo;`
  - Good to be aware of both; we will move to ES6 modules in this class

HTML

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <title>My Web Page</title>
  <!-- this is a comment -->
  <!-- scripts go here -->
 </head>
 <body>
  <!-- your document goes here -->
  <h1>Hello, World!</h1>
 </body>
</html>
```



Hello, World!

- HTML documents
  - HEAD
    - where you put the <TITLE> and <SCRIPT> calls, among other things
    - <SCRIPT SRC=""></SCRIPT> -- loads JavaScript
    - can also do this (frowned upon)
      <SCRIPT> … *your JavaScript goes here…*
      function doIt() { …. } </SCRIPT>
  - BODY
    - the page itself
- initially, basic formatting
  - <B>, <EM>
  - <P>, <BR>
  - <CENTER>
  - ...and a lot more….
- plus basic input/output
  -
  - <INPUT> (checkboxes, radio buttons, text, textareas)
  - all in a <FORM>
    - some history

        In the early-to-mid 1990s, most Web sites were based on complete HTML pages. Each user action required that a complete new page be loaded from the server. This process was inefficient, as reflected by the user experience: all page content disappeared, then the new page appeared. Each time the browser reloaded a page because of a partial change, all of the content had to be re-sent, even though only some of the information had changed. This placed additional load on the server and made bandwidth a limiting factor on performance.

        - that is, web pages worked like this:
            - enter information
            - click a button
            - a message would get sent over the network to a server (a URL) containing the info
            - the server would respond with a new HTML page
            - rinse and repeat
        - slow! (every action ⇒ round-trip to the server, with data transfer)
        - not very interactive
    - Microsoft actually invented interactive web pages!
        - for Outlook; eventually took over as "Web 2.0"
        - programmatically modify and query the user interface (the "DOM")
        - have some actions *asynchronously* (concurrently) do slow tasks like get data from the network ⇒ web page stays interactive, I/O in the background
            - "AJAX"

Interacting with the DOM

- entire HTML page is represented as a big object: `document` -- a tree
    - this object does not exist in node!
- elements can have IDs, and they can belong to CLASSES
    - these are all user-defined
    - you can access and update values by querying, getting elements with a particular ID, or all elements in a class…
    - you can also just access things by what kind of thing they are, like an <H1> tag
        - `document.getElementsByTagName("H1")`
- `document.getElementById('username')`
- `document.getElementsByClassName('deleted inactive');`

- `document` is a regular object, so you can do everything you can do to a regular object (e.g., delete things), and it will update the HTML
    - you can create elements and add them in

```
// create a new Text node
 let newText = document.createTextNode("MOAR TEXT");
 // create a new Element to be this paragraph
 let newElement = document.createElement("P");
 // put the text in the paragraph
 newElement.appendChild(newText);
```

Exercise!

https://docs.google.com/document/d/1xPrxJ1PvqhJwJc70J4wVNAbhtC1gXU8CrsutQI4phuI/edit?usp=sharing