HTML

- HTML consists of different *tags*
HTML consists of different *tags* e.g. `<html></html>`

- For example, every document starts with an `html` tag.
HTML

- HTML consists of different *tags*
  - For example, every document starts with an `html` tag.
  - There are so many different kinds of tags that do different things. (Video tags, music tags, form tags, code block tags, you name it...)

Some notable tags:
- `html`: Denotes an HTML document
- `head`: Contains metadata about the HTML document (e.g. title, Google search keywords, etc)
- `title`: Specifies the name of the page that should appear in the browser tab bar
- `body`: Contains the things that should be shown on the screen
- `div`: A versatile container element that can be used for anything (e.g. creating buttons, side panels, section separators, etc.)
- `img`: An image
- `a`: A clickable link to another page
HTML consists of different *tags*

- For example, every document starts with an `html` tag.
- There are so many different kinds of tags that do different things. (Video tags, music tags, form tags, code block tags, you name it...)
- We don't want you to memorize them. You can always Google them. We want you to know *how* to use them, and to be familiar with a small set of useful tags.

Some notable tags:

- `html`: Denotes an HTML document
- `head`: Contains metadata about the HTML document (e.g. title, Google search keywords, etc)
- `title`: Specifies the name of the page that should appear in the browser tab bar
- `body`: Contains the things that should be shown on the screen
- `div`: A versatile container element that can be used for anything (e.g. creating buttons, side panels, section separators, etc.)
- `img`: An image
- `a`: A clickable link to another page
• HTML consists of different *tags*
• Some tags have children, while others do not
HTML

- HTML consists of different *tags*
- Some tags have children, while others do not

The `p` (paragraph) tag accepts children (the text to put in the paragraph):

```html
<p>Hello world!</p>
```

The `input` tag (used for creating a text box) does not accept children:

```html
<input />
```

Same for the `br` (line break) tag:

```html
<br />
```
HTML

- HTML consists of different tags
- Some tags have children, while others do not
  - Tags that can accept children are conventionally written with an *open* and *close* tag: `<tag></tag>
  - Tags that do not accept children are conventionally written as *self-closing tags*: `<tag />`
  - This is just convention, and the browser usually accepts either form.

The `p` (paragraph) tag accepts children (the text to put in the paragraph):
<p>Hello world!</p>

The `input` tag (used for creating a text box) does not accept children:
<input />

Same for the `br` (line break) tag:
<br />
HTML

- HTML consists of different tags
- Some tags have children, while others do not
- Tags can also have attributes

\[
\text{\textless} \text{div}\text{\textgreater}</div>\]
HTML

- HTML consists of different *tags*
- Some tags have children, while others do not
- Tags can also have *attributes*
  - The `id` attribute is supposed to uniquely identify an element on the page

```html
<div id="signup-btn"></div>
```
HTML

- HTML consists of different *tags*
- Some tags have children, while others do not
- Tags can also have *attributes*
  - The *class* attribute labels an element as being a certain *kind* of thing. This is useful when you want to apply the same CSS style to all elements of the same class (e.g. all the buttons on the page), or if you want to attach a mouse listener to, e.g., all the buttons on the page.

```html
<div id="signup-btn" class="button">
</div>
```
HTML

- HTML consists of different tags
- Some tags have children, while others do not
- Tags can also have attributes
  - Each type of tag also has special attributes you can use to specify functionality. For example, the `img` (image) tag has a `src` attribute letting you specify what image file to load

```html
<div id="signup-btn" class="button">
  <img src="gold-star.png" />
  Sign me up!
</div>
```
HTML

- HTML consists of different **tags**
- Some tags have children, while others do not
- Tags can also have **attributes**
  - In fact, you can even define your own attributes! You can specify an attribute called anything you want. (These attributes can be targeted by CSS selectors and they can also be queried in JS.)
  - By convention, custom attributes should start with `data-` (although this isn't an enforced rule).

```html
<div id="signup-btn" class="button" data-myfield="Hello!">
  <img src="gold-star.png" />
  Sign me up!
</div>
```
HTML Files

- An HTML file is a text file that looks something like this:

```html
<html>
<head>
  <meta charset="UTF-8">
  <title>To-Do List</title>
  <script src="todo.js" type="text/javascript"></script>
</head>
<body>
  Tasks for November 11th, 2019:
  <ul id="to-do-list">
    <li>Teach CS106AX and CS106X lectures.</li>
    <li>Dinner in San Francisco at L’Ardoise.</li>
  </ul>
  <hr/>
  <button id="clear-all-items" type="button">Clear</button>
</body>
</html>
```
HTML Files

- An HTML file is a text file
- The browser loads the text file and generates an object-based representation of the `<body>` portion of the file (called the DOM)
HTML Files

- An HTML file is a text file
- The browser loads the text file and generates an object-based representation of the `<body>` portion of the file (called the DOM)
  - DOM = Document Object Model
HTML Files

- An HTML file is a text file
- The browser loads the text file and generates an object-based representation of the `<body>` portion of the file (called the DOM)
  - DOM = Document Object Model
  - Every HTML tag is stored in a "node" object (sometimes called "elements")
An HTML file is a text file. The browser loads the text file and generates an object-based representation of the `<body>` portion of the file (called the DOM).

- **DOM = Document Object Model**
- Every HTML tag is stored in a "node" object (sometimes called "elements")
- Every node keeps an array of child nodes
HTML Files

- An HTML file is a text file
- The browser loads the text file and generates an object-based representation of the `<body>` portion of the file (called the DOM)
  - DOM = Document Object Model
  - Every HTML tag is stored in a "node" object (sometimes called "elements")
  - Every node keeps an array of child nodes
  - This is basically what you did in Adventure! The `ul`, `hr`, and `button` nodes are like rooms, and the `li` nodes are like objects that you store in the room's object array

```html
<body>
  <ul id="to-do-list">
    <li>Teach CS106AX and CS106X lectures.</li>
    <li>Dinner in San Francisco at L’Ardoise.</li>
  </ul>
  <hr/>
  <button id="clear-all-items" type="button">Clear</button>
</body>
```
An HTML file is a text file

The browser loads the text file and generates an object-based representation of the `<body>` portion of the file (called the DOM)

- DOM = Document Object Model
- Every HTML tag is stored in a "node" object (sometimes called "elements")
- Every node keeps an array of child nodes
- You can draw the DOM as a sort of upside-down tree structure:
**HTML Files**

- An HTML file is a text file
- The browser loads the text file and generates an object-based representation of the `<body>` portion of the file (called the DOM)
- Once the DOM is loaded, the browser generates a visual representation of the page and shows it on the screen
HTML Files

- An HTML file is a text file
- The browser loads the text file and generates an object-based representation of the `<body>` portion of the file (called the DOM)
- Once the DOM is loaded, the browser generates a visual representation of the page and shows it on the screen
- You can get information from the DOM in Javascript
  - Get a node by ID: `document.getElementById("some-id");`
  - Get an array of nodes by class:
    ```javascript
    document.getElementsByClassName("some-class");
    ```
  - Get an array of nodes by tag name: `document.getElementsByTagName("li");`
  - Get an attribute from a node: `node.getAttribute("src");`
HTML Files

- An HTML file is a text file
- The browser loads the text file and generates an object-based representation of the `<body>` portion of the file (called the DOM)
- Once the DOM is loaded, the browser generates a visual representation of the page and shows it on the screen
- You can get information from the DOM in Javascript
- You can also modify the DOM, causing the page to update in appearance
  - Create a new node (but don't add it to the screen yet -- kind of like creating a GImage/GCompound):
    ```javascript
    let newImage = document.createElement("img");
    ```
  - Create a new text node (kind of like creating a GLabel):
    ```javascript
    let textNode = document.createTextNode("text to display");
    ```
  - Add the new node to the end of the document (kind of like gw.add()):
    ```javascript
    document.appendChild(newImage);
    ```
  - Add the new node as a child of an existing node (kind of like gcompound.add()):
    ```javascript
    existingDiv.appendChild(newImage);
    ```
HTML Files

- An HTML file is a text file
- The browser loads the text file and generates an object-based representation of the `<body>` portion of the file (called the DOM)
- Once the DOM is loaded, the browser generates a visual representation of the page and shows it on the screen
- You can get information from the DOM in Javascript
- You can also modify the DOM, causing the page to update in appearance
- Note that once the DOM is loaded, you aren't working with the text file anymore!
  - No text parsing or complicated string manipulation in this assignment!
Big Picture

- **HTML file**
  - Browser reads HTML file

- **DOM**
  - Browser renders DOM to visual representation
  - JS can read nodes from DOM
  - JS can change DOM, leading to update in visual presentation on screen

- **Screen**
Assignment
Live Demo

(If you are reading these slides afterwards, see pages 1-2 of the handout)
Logistics

- Due Friday, Nov 22nd
- Taking one late "day" gets you until the Monday after Thanksgiving break
  - I don't recommend doing this unless you really need to! This assignment isn't that long and you'll feel better if you don't need to worry about work over break
- Partners are allowed
Components of the Assignment

- No writing HTML! (You should understand what's there, but you don't need to add anything.)

- No writing CSS! (In fact, you probably don't even need to look at what's there.)

- Only need to change `match-the-flags.js`
  - You'll need to work with arrays, event (click) handlers, and super simple animation (making an image disappear after 1 second)
  - However, NO graphics library work! All graphics will be generated through the DOM :)


<html>
<head>
    <meta charset="UTF-8">
    <title>Match The Flag</title>
    <link rel="stylesheet" type="text/css" href="match-the-flags.css"/>
    <script type="text/javascript" src="match-the-flags-constants.js"></script>
    <script type="text/javascript" src="match-the-flags.js"></script>
</head>
<body>
    <div id="board"></div>
</body>
</html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Match The Flag</title>
  <link rel="stylesheet" type="text/css" href="match-the-flags.css"/>
  <script type="text/javascript" src="match-the-flags-constants.js"></script>
  <script type="text/javascript" src="match-the-flags.js"></script>
</head>
<body>
  <div id="board"></div>
</body>
</html>
HTML file

<html>
<head>
    <meta charset="UTF-8"/>
    <title>Match The Flag</title>
    <link rel="stylesheet" type="text/css" href="match-the-flags.css"/>
    <script type="text/javascript" src="match-the-flags-constants.js"></script>
    <script type="text/javascript" src="match-the-flags.js"></script>
</head>
<body>
    <div id="board"></div>
</body>
</html>
There is only one div of interest
HTML file

<body>
  <div id="board"></div>
</body>

- There is only one div of interest
  - You can get the corresponding DOM node in JS:
    ```js
    let board = document.getElementById("board");
    ```
There is only one `div` of interest
  ○ You can get the corresponding DOM node in JS:
    ```javascript
    let board = document.getElementById("board");
    ```
● You are going to create 16 `img` elements and add them to that `div`
There is only one `div` of interest
  ○ You can get the corresponding DOM node in JS:
    ```javascript
    let board = document.getElementById("board");
    ```

You are going to create 16 `img` elements and add them to that `div`

The CSS is already in place to position the images inside of `board` such that the first four images will be in the first row, the next four will be in the next row, etc.
DOMContentLoaded?

- The starter code has this code written:

```javascript
function BootstrapMatchTheFlag() {
}

/* Execute the above function when the DOM tree is fully loaded. */
document.addEventListener("DOMContentLoaded", BootstrapMatchTheFlag);
```

- Treat BootstrapMatchTheFlag as your main function
  - Of course, you can/should still decompose your code into other functions. These functions can be top-level functions

- The DOMContentLoaded event is fired (i.e. BootstrapMatchTheFlag is called) once the HTML file is completely done loading
  - Although it's possible to access the DOM before DOMContentLoaded is fired, you might be missing parts of the page that haven't loaded yet
Milestone 1

- The starter code contains an `images` directory with images of country flags
  - `images/finland.png` is an image of Finland's flag
Milestone 1

- The starter code contains an images directory with images of country flags
- It also contains a COUNTRIES constant:
  ```javascript
  const COUNTRIES = [
    "Belize", "Brazil", "China", "Colombia",
    "Egypt", "Finland", "Greece", "India"
  ];
  ```
Milestone 1

- The starter code contains an images directory with images of country flags
- It also contains a COUNTRIES constant:
  ```javascript
  const COUNTRIES = [
    "Belize", "Brazil", "China", "Colombia",
    "Egypt", "Finland", "Greece", "India"
  ];
  
  You should add code to BootstrapMatchTheFlag to create an array with 16 image path strings, two per country (e.g. "images/finland.png")
Milestone 1

- The starter code contains an images directory with images of country flags
- It also contains a COUNTRIES constant:
  ```javascript
  const COUNTRIES = [
    "Belize", "Brazil", "China", "Colombia",
    "Egypt", "Finland", "Greece", "India"
  ];
  ```
- You should add code to `BootstrapMatchTheFlag` to create an array with 16 image path strings, two per country (e.g. "images/finland.png")
  - Note that COUNTRIES contains proper cased strings, and your image paths should be all lowercase
Milestone 1: Creating image path strings

- The starter code contains an `images` directory with images of country flags.
- It also contains a `COUNTRIES` constant:
  ```javascript
  const COUNTRIES = [
    "Belize", "Brazil", "China", "Colombia",
    "Egypt", "Finland", "Greece", "India"
  ];
  ```
- You should add code to `BootstrapMatchTheFlag` to create an array with 16 image path strings, two per country (e.g. "images/finland.png")
- Then, call the provided `shuffle` function on your array to randomize the order of the paths.
Milestone 2: Showing image flags

- For each filename in your array, create an `img` node and add it to the `board div`
  - You know how to create an `img` node (`document.createElement`), but how to you tell that node what image to display?
Milestone 2: Showing image flags

- For each filename in your array, create an `img` node and add it to the `board div`
  - `node.setAttribute("src", filename);`
Milestone 2: Showing image flags

- For each filename in your array, create an `img` node and add it to the `board` `div`
Milestone 3: "Covering" image flags

- We want to "cover" each tile by showing images/cover.png instead of the country flag image
  - Easy! You could do this by changing your milestone 2 code to show this image instead of the country flag
  - However, if you do that, it will be hard to un-cover a tile when you implement the click handling logic later. When you go to un-cover a tile, what country flag should you use?
  - To make this easier, we recommend adding a custom attribute data-country-image to each img node, containing the original country flag image filename
Milestone 3: "Covering" image flags

- You can confirm that this is done correctly by looking for the `data-country-image` attributes in the Chrome inspector:

```html
<!DOCTYPE html>
<html>
<head>
</head>
<body class="esc-Initialized">

```
Milestone 4: Click to reveal image flag

- Add a click-handling function to each `img` node that toggles between the gray image and the country flag
  - `node.addEventListener("click", clickHandler);
    function clickHandler(e) {
      // e.target is the img node that was clicked
    }
  - If the clicked `img` node is currently displaying the gray image, replace the `src` attribute to show the country flag instead
  - If it is currently showing the country flag, show the gray image instead

- You shouldn't be creating any new nodes! You only need to modify the existing `img` node (available as `e.target`).
Milestone 5: Implement gameplay logic

- Modify your click handler from Milestone 4
  - You need to keep track of which tiles are currently revealed
  - When clicking a tile for the first time, you should reveal the tile (your Milestone 4 code already does this)
  - When you click a second tile, reveal that tile, and check whether it is showing the same flag as the first tile
Milestone 5: Implement gameplay logic

- Modify your click handler from Milestone 4
  - You need to keep track of which tiles are currently revealed
  - When clicking a tile for the first time, you should reveal the tile (your Milestone 4 code already does this)
  - When you click a second tile, reveal that tile, and check whether it is showing the same flag as the first tile
    - If the tiles are the same, set a 1-second timer that "removes" both tiles by setting `src` to a transparent image (see `MATCHED_IMAGE` constant)
Milestone 5: Implement gameplay logic

- Modify your click handler from Milestone 4
  - You need to keep track of which tiles are currently revealed
  - When clicking a tile for the first time, you should reveal the tile (your Milestone 4 code already does this)
  - When you click a second tile, reveal that tile, and check whether it is showing the same flag as the first tile
    - If the tiles are the same, set a 1-second timer that "removes" both tiles by setting `src` to a transparent image (see `MATCHED_IMAGE` constant)
  - Why wouldn't we want to simply remove the `img` node?
Milestone 5: Implement gameplay logic

● Modify your click handler from Milestone 4
  ○ You need to keep track of which tiles are currently revealed
  ○ When clicking a tile for the first time, you should reveal the tile (your Milestone 4 code already does this)
  ○ When you click a second tile, reveal that tile, and check whether it is showing the same flag as the first tile
    ■ If the tiles are the same, set a 1-second timer that "removes" both tiles by setting `src` to a transparent image (see MATCHED_IMAGE constant)
  ● Why wouldn't we want to simply remove the `img` node?
    ○ That will mess up the layout of the images
Milestone 5: Implement gameplay logic

- Modify your click handler from Milestone 4
  - You need to keep track of which tiles are currently revealed
  - When clicking a tile for the first time, you should reveal the tile (your Milestone 4 code already does this)
  - When you click a second tile, reveal that tile, and check whether it is showing the same flag as the first tile
    - If the tiles are the same, set a 1-second timer that "removes" both tiles by setting `src` to a transparent image (see `MATCHED_IMAGE` constant)
  - Why wouldn't we want to simply remove the `img` node?
    - That will mess up the layout of the images
  - How do you set a 1-second timer?
Milestone 5: Implement gameplay logic

- Modify your click handler from Milestone 4
  - You need to keep track of which tiles are currently revealed
  - When clicking a tile for the first time, you should reveal the tile (your Milestone 4 code already does this)
  - When you click a second tile, reveal that tile, and check whether it is showing the same flag as the first tile
    - If the tiles are the same, set a 1-second timer that "removes" both tiles by setting `src` to a transparent image (see MATCHED_IMAGE constant)
      - Why wouldn't we want to simply remove the `img` node?
        - That will mess up the layout of the images
      - How do you set a 1-second timer?
        - `setTimeout`
Milestone 5: Implement gameplay logic

- Modify your click handler from Milestone 4
  - You need to keep track of which tiles are currently revealed
  - When clicking a tile for the first time, you should reveal the tile (your Milestone 4 code already does this)
  - When you click a second tile, reveal that tile, and check whether it is showing the same flag as the first tile
    - If the tiles are the same, set a 1-second timer that "removes" both tiles by setting `src` to a transparent image (see `MATCHED_IMAGE` constant)
    - If the tiles are different, set a 1-second timer that covers the tiles back up
Milestone 5: Implement gameplay logic

- Modify your click handler from Milestone 4
- Edge cases
  - If you click a tile, then click the same tile again for the second click, you shouldn't do anything. (Don't cover the tile back up, and definitely don't treat the second click as a correct guess)
  - If you click a tile that has been "removed" (i.e. set to the transparent image), don't do anything -- just ignore the click.
  - If you click while the timer is running (i.e. while two country flags are being shown, before they get covered back up or "removed"), don't do anything. (There should never be more than 2 country flags shown at a time, no matter how many times the user clicks.)
- This is going to be the hardest milestone
Questions?