CS193X: Web Programming Fundamentals

Spring 2017

Victoria Kirst
(vrk@stanford.edu)
Today's schedule

Today
- Finish up gift example
- Case study: Tic-Tac-Toe
- DOM revisited

Announcements
- Victoria's Office Hours today from 2:30 to 4pm
- Please use Piazza instead of email for communications
  - Post privately if you are sharing code
  - Send email only for very personal issues or if instructed by course staff
- HW2 updated with more info, test files for extension
**Forgot last time: List operations**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>list.push(element)</code></td>
<td>Add <em>element</em> to back</td>
</tr>
<tr>
<td><code>list.unshift(element)</code></td>
<td>Add <em>element</em> to front</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>list.pop()</code></td>
<td>Remove from back</td>
</tr>
<tr>
<td><code>list.shift()</code></td>
<td>Remove from front</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>list.indexOf(element)</code></td>
<td>Returns numeric index for <em>element</em> or -1 if none found</td>
</tr>
</tbody>
</table>
Forgot last time: splice

Add/remove element at index: **splice**

```
list.splice(startIndex, deleteCount, item1, item2, ...)
```

Remove one element at index 3:
```
list.splice(3, 1);
```

Add **element** at index 2:
```
list.splice(2, 0, element);
```
Back to events, etc…
Example: Present

Click for a present:

```
function openPresent() {
    const image = document.querySelector('img');
    image.src = 'https://media.giphy.com/media/27ppQU0xe7KlG/giphy.gif';
}
```

```
const image = document.querySelector('img');
image.addEventListener('click', openPresent);
```

See the [CodePen](https://codepen.io) - much more exciting!
Finding the element twice...

```javascript
function openPresent() {
    const image = document.querySelector('img');
    image.src = 'https://media.giphy.com/media/27ppQU0xe7KlG/giphy.gif';
}

const image = document.querySelector('img');
image.addEventListener('click', openPresent);
```

This redundancy is unfortunate.

Q: Is there a way to fix it?
Finding the element twice...

```javascript
function openPresent() {
    const image = document.querySelector('img');
    image.src = 'https://media.giphy.com/media/z7ppQU0xe7KlG/giphy.gif';
}

const image = document.querySelector('img');
image.addEventListener('click', openPresent);
```

This redundancy is unfortunate.

**Q: Is there a way to fix it?**

[CodePen](https://codepen.io/)
Event.currentTarget

An Event element is passed to the listener as a parameter:

```javascript
function openPresent(event) {
    const image = event.currentTarget;
    image.src = 'https://media.giphy.com/media/27ppQU0xe7KlG/giphy.gif';
    image.removeEventListener('click', openPresent);
}

const image = document.querySelector('img');
image.addEventListener('click', openPresent);
```

The event's currentTarget property is a reference to the object that we attached to the event, in this case the <img>'s Element to which we added the listener.
Psst.. Not to be confused with Event.target

(Note: Event has both:
- `event.target`: the element that was clicked / "dispatched the event" (might be a child of the target)
- `event.currentTarget`: the element that the original event handler was attached to)

(Programming note: I got these mixed up in lecture and used target when I meant currentTarget, so I'm correctly the slides retroactively. Whoops, sorry!)
Example: Present

Click for a present:

It would be nice to change the text after the present is "opened"...
Some properties of Element objects

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>id</strong></td>
<td>The value of the id attribute of the element, as a string</td>
</tr>
<tr>
<td><strong>innerHTML</strong></td>
<td>The raw HTML between the starting and ending tags of an element, as a string</td>
</tr>
<tr>
<td><strong>textContent</strong></td>
<td>The text content of a node and its descendants. (This property is inherited from Node)</td>
</tr>
<tr>
<td><strong>classList</strong></td>
<td>An object containing the classes applied to the element</td>
</tr>
</tbody>
</table>

Maybe we can adjust the **textContent**!  
[CodePen](https://codepen.io)
We can select the h1 element then set its `textContent` to change what is displayed in the h1. (CodePen)
Another approach: Changing the elements
Add elements via DOM

We can create elements dynamically and add them to the web page via `createElement` and `appendChild`:

```javascript
document.createElement('tag string')
 element.appendChild(element);
```

Technically you can also add elements to the webpage via `innerHTML`, but it poses a security risk.

// Try not to use innerHTML like this:
 element.innerHTML = '<h1>Hooray!</h1>';
Remove elements via DOM

We can also call remove elements from the DOM by calling the `remove()` method on the DOM object:

```javascript
element.remove();
```

And actually setting the `innerHTML` of an element to an **empty string** is a **fine way** of removing all children from a parent node:

```javascript
// This is fine and poses no security risk.
element.innerHTML = '';
```
function openPresent(event) {
    const newHeader = document.createElement('h1');
    newHeader.textContent = 'Hooray!';
    const newImage = document.createElement('img');
    newImage.src = 'https://media.giphy.com/media/27ppQUOxe7KlG/giphy.gif';

    const container = document.querySelector('#container');
    container.innerHTML = '';
    container.appendChild(newHeader);
    container.appendChild(newImage);
}

const image = document.querySelector('img');
image.addEventListener('click', openPresent);
Hmm, the effect is slightly janky though: The text changes faster than the image loads.

**Q: How do we fix this issue?**
display: none;

There is yet another super helpful value for `display`:

- display: block;
- display: inline;
- display: inline-block;
- display: flex;
- display: none;

`display: none;` turns off rendering for the element and all its children. It's treated as if the element were not in the document at all...
There is yet another super helpful value for `display`:

```css
display: block;
display: inline;
display: inline-block;
display: flex;
display: none;
```

`display: none;` turns off rendering for the element and all its children. It's treated as if the element were not in the document at all...

...but the content (such as the images) is still loaded.
We can add both views to the HTML, with one view hidden by default…

(CodePen)
function openPresent(event) {
    const image = event.currentTarget;
    image.removeEventListener('click', openPresent);

    const giftOutside = document.querySelector('#gift-outside');
    const giftInside = document.querySelector('#gift-inside');
    giftOutside.classList.add('hidden');
    giftInside.classList.remove('hidden');
}

const image = document.querySelector('#gift-outside img');
image.addEventListener('click', openPresent);

Then we toggle the display state of the containers by adding/removing the hidden class.

(CodePen)
Several strategies for updating HTML elements in JS:

1. Change content of existing HTML elements in page:
   - Good for simple text updates

2. Add elements via `createElement` and `appendChild`:
   - Needed if you're adding a variable number of elements

3. Put all "views" in the HTML but set inactive ones to `hidden`, then update `display` state as necessary.
   - Good when you know ahead of time what element(s) you want to display
   - Can be used in conjunction with (1) and/or (2)
Case Study: A longer JS example
Example: Tic Tac Toe

Let's try to implement a game of Tic-Tac-Toe. (finished)
Tic Tac Toe plan

1. Every time we click on an empty space, change the empty space in an "X" by adding an image of an "x" into the empty <div>

2. After our turn, the computer puts an "O" in a random empty space

3. When there are 3 Xs or 3 Os in a row, declare a winner

CodePen starter code
Empty square -> X

First we need to make all div children of #grid clickable... how do we do that?

```html
<body>
  <h1>Tic-Tac-Toe</h1>
  <div id="grid">
    <div></div>
    <div></div>
    <div></div>
    <div></div>
    <div></div>
    <div></div>
    <div></div>
    <div></div>
    <div></div>
    <div></div>
  </div>
</body>
```
In `changeToX`, we need to add an `<img>` tag into the clicked element…

How do we do that?
Empty square -> X

```html
<body>
  <h1>Tic-Tac-Toe</h1>
  <div id="grid">
    <div></div>
    <div></div>
    <div></div>
    <div></div>
    <div></div>
    <div></div>
    <div></div>
    <div></div>
    <div></div>
    <div></div>
    <div></div>
  </div>

  <script>
    function changeToX(event) {
      const container = event.currentTarget;
      const image = document.createElement('img');
      image.src = X_IMAGE_URL;
      container.appendChild(image);
      container.removeEventListener('click', changeToX);
    }

    const boxes = document.querySelectorAll('#grid div');
    for (const box of boxes) {
      box.addEventListener('click', changeToX);
    }
  </script>
</body>
```

Step 1 Complete: CodePen
Tic Tac Toe plan

1. Every time we click on an empty space, change the empty space in an "X" by adding an image of an "x" into the empty space.

2. After our turn, the computer puts an "O" in a random empty space.

3. When there are 3 Xs or 3 Os in a row, declare a winner.
Aside: Random in JS

Inconveniently, JavaScript only has one* random generator: `Math.random()`

- `Math.random()` returns a random floating point number between [0, 1) (0 inclusive, 1 exclusive)

To get a random number from 0 inclusive to max exclusive: `Math.floor(Math.random() * max)`;

(Intuition: It's like a random percentage of max… so if max is 5, then [0, 0.2) maps to 0, [0.2, 0.4) maps to 1, [0.4, 0.6) maps to 2, [0.6, 0.8) maps to 3, [0.8, 1) maps to 4)

*aside from crypto libraries
How do we figure out an empty space?
Empty space: DOM approach

Maybe something like:
- For each `#grid div`
  - See if it has an `img` child

Note that `querySelector` can also be used on an element, not just document:

```javascript
const sectionElement = document.querySelector('section');
// All h1s that are children of sectionElement:
const headers = sectionElement.querySelector('h1');
```
function computerChoose0() {
    const allBoxes = document.querySelectorAll('#grid div');
    const freeBoxes = [];
    for (const box of allBoxes) {
        let imageChild = box.querySelector('img');
        if (!imageChild) {
            freeBoxes.push(box);
        }
    }
    const index = Math.floor(Math.random() * freeBoxes.length);
    const freeSpace = freeBoxes[index];
    const image = document.createElement('img');
    image.src = O_IMAGE_URL;
    freeSpace.removeEventListener('click', changeToX);
    freeSpace.appendChild(image);
}
Don't query UI for state

We're querying the UI state to understand the game state.

This is not a great software engineering technique:
- Couples your "view" and your "model"
- Can lead to hard-to-find bugs:
  - What if we later decide to display X's and O's using background-image instead of an `<img>` tag?
- Code is also a little hard to read
  - What do "img" tags have to do with a free space?

Better to keep track of state separately from UI!
Better(?) approach: Global Variable

We can instead store the game state in a global variable:

```javascript
const freeBoxes = [];
const boxes = document.querySelectorAll('#grid div');
for (const box of boxes) {
    box.addEventListener('click', changeToX);
    freeBoxes.push(box);
}
```

`freeBoxes` is our array that contains the available boxes
Better(?) approach: Global Variable

```javascript
function changeToX(event) {
    const container = event.currentTarget;
    const image = document.createElement('img');
    image.src = X_IMAGE_URL;
    container.appendChild(image);
    container.removeEventListener('click', changeToX);

    // Also remove |container| from |freeBoxes|
    const indexToRemove = freeBoxes.indexOf(container);
    freeBoxes.splice(indexToRemove, 1);
    computerChoose0();
}
```

Then we update the freeBoxes state when we add an X...
Better(?) approach: Global Variable

```javascript
function computerChooseO() {
  const allBoxes = document.querySelectorAll('#grid div');
  const index = Math.floor(Math.random() * freeBoxes.length);
  const freeSpace = freeBoxes[index];
  // Remove the chosen box from freeBoxes.
  freeBoxes.splice(index, 1);
  const image = document.createElement('img');
  image.src = O_IMAGE_URL;
  freeSpace.removeEventListener('click', changeToX);
  freeSpace.appendChild(image);
}

...And when the computer add an O. (CodePen)
Is that really better?!

What's wrong with that solution?
- Aren't we still coupling UI with state a little bit?
  - We are storing references to UI elements in `freeBoxes` to track which ones are free...
- Aren't global variables bad?!
  - We aren't supposed to create global variables in other programming contexts...
Is that really better?!

What's wrong with that solution?
- Aren't we still coupling UI with state a little bit?
  - We are storing references to UI elements in freeBoxes to track which ones are free...
- Aren't global variables bad?!
  - We aren't supposed to create global variables in other programming contexts…

(We'll deal with these problems next week)
(Basically we want classes)
Tic Tac Toe plan

1. Every time we click on an empty space, change the empty space in an "X" by adding an image of an "x" into the empty `<div>`

2. After our turn, the computer puts an "O" in a random empty space

3. When there are 3 Xs or 3 Os in a row, declare a winner
Distinguishing boxes

The same event handler is called for each element.

How do we distinguish between elements?
**Terrible idea:** 9 event handlers

```html
<body>
  <h1>Tic-Tac-Toe</h1>
  <div id="grid">
    <div id="one"></div>
    <div id="two"></div>
    <div id="three"></div>
    <div id="four"></div>
    <div id="five"></div>
    <div id="six"></div>
    <div id="seven"></div>
    <div id="eight"></div>
    <div id="nine"></div>
  </div>
</body>
```

```javascript
function changeToXRow1Column1(event) {
  //...
}
function changeToXRow1Column2(event) {
  //...
}
function changeToXRow1Column3(event) {
  //...
}
const first = document.querySelector('#one');
first.addEventListener('click', changeToXRow1Column1);
const second = document.querySelector('#two');
second.addEventListener('click', changeToXRow1Column2);
```
Uniquely identifying items

But this idea of uniquely identifying squares is a good one!
Add another state variable, `takenBoxes`, that maps box number to who owns the box.

```javascript
const freeBoxes = [];
// Map of box number -> 'x' or 'o'
const takenBoxes = {};
const boxes = document.querySelectorAll('#grid div');
for (const box of boxes) {
  box.addEventListener('click', changeToX);
  freeBoxes.push(box);
}
```
function changeToX(event) {
    assignSpace(event.currentTarget, 'x');
}

function computerChooseO() {
    const allBoxes = document.querySelectorAll('#grid div');
    const index = Math.floor(Math.random() * freeBoxes.length);
    const freeSpace = freeBoxes[index];
    assignSpace(freeSpace, 'o');
}

function assignSpace(space, owner) {
    const image = document.createElement('img');
    image.src = owner === 'x' ? X_IMAGE_URL : O_IMAGE_URL;
    space.appendChild(image);

    takenBoxes[space.id] = owner;
    const indexToRemove = freeBoxes.indexOf(space);
    freeBoxes.splice(indexToRemove, 1);
    space.removeEventListener('click', changeToX);
}

Update takenBoxes with the owner each time a space is assigned.
Find winner by checking rows, columns and diagonal spaces

function getWinner() {
    // Check rows
    let rowResult = checkboxes('one', 'two', 'three') ||
                    checkboxes('four', 'five', 'six') ||
                    checkboxes('seven', 'eight', 'nine');

    // Check columns
    let colResult = checkboxes('one', 'four', 'seven') ||
                    checkboxes('two', 'five', 'eight') ||
                    checkboxes('three', 'six', 'nine');

    // Check diagonal
    let diagonalResult = checkboxes('one', 'five', 'nine') ||
                         checkboxes('three', 'five', 'seven');
    return rowResult || colResult || diagonalResult;
}

function checkboxes(one, two, three) {
    if (takenBoxes[one] !== undefined &&
        takenBoxes[one] === takenBoxes[two] &&
        takenBoxes[two] === takenBoxes[three]) {
        return takenBoxes[one];
    }
    return null;
}
Create a results div and add results to the div (CodePen)
Attach "data" to divs?

Wouldn't it be nicer if we could operate on numbers instead of string ids?

But we can't have numeric IDs…

Is there some way to attach additional "data" to an element?
Data attributes

You can assign special `data-* attributes` to HTML elements to give associate additional data with the element.

`data-your-name="Your Value"`

```html
<article
    id="electriccars"
    data-columns="3"
    data-index-number="12314"
    data-parent="cars">
  ...
</article>
```
Data attributes in JavaScript

You can access your custom-defined data attributes via the dataset object on the DOM object:

```javascript
var article = document.getElementById('electriccars');

article.dataset.columns // "3"
article.dataset.indexNumber // "12314"
article.dataset.parent // "cars"
```

- Dash-separated words turn to camel case, e.g. `data-index-number` in HTML is `dataset.indexNumber` in JS

- **Aside:** Data attributes are returned as strings, but you can cast them to Number via `parseInt`
Data attributes in CSS

You can also style data attributes in CSS:

```
[data-variable-name] or
[data-variable-name='value'] or
element[data-variable-name] etc
```

```css
article[data-columns='3'] {
  width: 400px;
}
article[data-columns='4'] {
  width: 600px;
}
```
<body>
  <h1>Tic-Tac-Toe</h1>
  <div id="grid">
    <div data-index="0"></div>
    <div data-index="1"></div>
    <div data-index="2"></div>
    <div data-index="3"></div>
    <div data-index="4"></div>
    <div data-index="5"></div>
    <div data-index="6"></div>
    <div data-index="7"></div>
    <div data-index="8"></div>
  </div>
  <div id="results"></div>
</body>

const index = parseInt(space.dataset.index);
takenBoxes[index] = owner;
Understanding the DOM
DOM Nodes

If the DOM is a tree composed of Nodes…

Q: Does that mean a Node in the DOM has child pointers like the trees we learned about in 106B?
DOM Nodes

If the DOM is a tree composed of Nodes…

Q: Does that mean a Node in the DOM has child pointers like the trees we learned about in 106B?  
A: Yes!
### Node properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>textContent</code></td>
<td>The text content of a node and its descendants. (This property is writeable)</td>
</tr>
<tr>
<td><code>childNodes</code></td>
<td>An array of this node's children (empty if a leaf)</td>
</tr>
<tr>
<td><code>parentNode</code></td>
<td>A reference to this node's parent Node</td>
</tr>
</tbody>
</table>

```html
<body>
  <h1>My favorites</h1>
  <section>
    <p>Strawberries</p>
    <p>Chocolate</p>
  </section>
</body>
```

What's the `parentNode` of `<section>`?
parentNode

```javascript
section = document.querySelector('section');

section.parentNode
```

```html
<body>
  <h1>My favorites</h1>
  <section>
    <p>Strawberries</p>
    <p>Chocolate</p>
  </section>
</body>
```

The `parentNode` of `<section>` is `<body>`.

What are the `childNodes` of `<section>`?
Why does `section` have 5 children, not 2?!
In addition to `Element` nodes, the DOM also contains `Text` nodes. All text present in the HTML, including `whitespace`, is contained in a text node:

```html
<body>
  <h1>My favorites</h1>
  <section>
    <p>Strawberries</p>
    <p>Chocolate</p>
  </section>
</body>
```
All text present in the HTML, including whitespace, is contained in a `<Text>` node:

```
<body>
  <h1>My favorites</h1>
  <section>
    <p>Strawberries</p>
    <p>Chocolate</p>
  </section>
</body>
```
DOM and Text nodes

The DOM is composed of Nodes, and there are several subtypes of Node.

- **Element**: HTML (or SVG) elements in the DOM
- **Text**: Text content in the DOM, including whitespace
  - **Text** nodes cannot contain children (are always leafs)
- **Comment**: HTML comments
- *(more)*

The type of a node is stored in the **nodeType** property
Q: How would we print out all nodes in the DOM?
Q: How would we print out all nodes in the DOM?
A: Recursively walk the DOM tree:

```javascript
function walkTree(root, level) {
    if (root.nodeType == Node.TEXT_NODE) {
        console.log(level + 'text:' + root.textContent);
    } else {
        console.log(level + root.nodeName);
    }
    for (const child of root.childNodes) {
        walkTree(child, level + "   ");
    }
}
walkTree(document.querySelector('html'), "");
```
What's the point?

- If we have `document.querySelector` that lets us get elements in the DOM…
- And if we can change the HTML as necessary to add classes/ids/elements/etc to select the right things…

Q: When would we ever want to traverse the DOM?
What's the point?

- If we have `document.querySelector` that lets us get elements in the DOM…
- And if we can change the HTML as necessary to add classes/ids/elements/etc to select the right things…

Q: When would we ever want to traverse the DOM?

A: Pretty much only in browser extensions or the Web Console (i.e. manipulating someone else's page)
Browser extensions

- Add-on that extends the functionality of the browser
- A piece of JavaScript that is injected into the webpage before or after it has loaded
Hacks and Mischief
More next time!
Example: Egghead

Dan Abromov's Redux videos

$199.99 PER YEAR
Example: Quora signin wall

Example: Adblock block

http://www.ondemandkorea.com/kpop-star-season-6-seoul-qualifier.html