CSE 154

LECTURE 18: THE DOCUMENT OBJECT MODEL (DOM);
UNOBTRUSIVE JAVASCRIPT
Event-driven programming

- JS programs have no `main`; they respond to user actions called **events**
- **event-driven programming**: writing programs driven by user events
Event handlers

- JavaScript functions can be set as **event handlers**
  - when you interact with the element, the function will execute
- **onclick** is just one of many event HTML attributes we'll use
Buttons: `<button>`

*the canonical clickable UI control (inline)*

```
<button onclick="myFunction();">Click me!</button>
```

- button's text appears inside tag; can also contain images
- To make a responsive button or other UI control:
  1. choose the control (e.g. button) and event (e.g. mouse click) of interest
  2. write a JavaScript function to run when the event occurs
  3. attach the function to the event on the control
### Accessing an element: document.getElementById

<table>
<thead>
<tr>
<th>JS</th>
<th>HTML</th>
</tr>
</thead>
<tbody>
<tr>
<td>var name = document.getElementById(&quot;id&quot;);</td>
<td>&lt;img id=&quot;icon01&quot; src=&quot;images/octopus.jpg&quot; alt=&quot;an animal&quot; /&gt; &lt;button onclick=&quot;changeImage();&quot;&gt;Click me!&lt;/button&gt;</td>
</tr>
<tr>
<td>function changeImage() { var octopusImage = document.getElementById(&quot;icon01&quot;); octopusImage.src = &quot;images/kitty.gif&quot;; }</td>
<td></td>
</tr>
</tbody>
</table>

- **document.getElementById** returns the DOM object for an element with a given `id`
Document Object Model (DOM)

*a set of JavaScript objects that represent each element on the page*

- each tag in a page corresponds to a JavaScript DOM object
- JS code can talk to these objects to examine elements' state
  - e.g. see whether a box is checked
- we can change state
  - e.g. insert some new text into a **div**
- we can change styles
  - e.g. make a paragraph red
DOM element objects

• access/modify the attributes of a DOM object with `objectName.attributeName`

• most DOM object attributes have the same names as the corresponding HTML attribute
  • `img` tag's `src` property
  • `a` tag's `href` property
## DOM object properties

```html
<div id="main" class="foo bar">
  <p>See our <a href="sale.html" id="saleslink">Sales</a> today!</p>
  <img id="icon" src="images/borat.jpg" alt="Borat" />
</div>
```

```javascript
var mainDiv = document.getElementById("main");
var icon = document.getElementById("icon");
var theLink = document.getElementById("saleslink");
```

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>tagName</td>
<td>element's HTML tag</td>
<td>mainDiv.tagName is &quot;DIV&quot;</td>
</tr>
<tr>
<td>className</td>
<td>CSS classes of element</td>
<td>mainDiv.className is &quot;foo bar&quot;</td>
</tr>
<tr>
<td>innerHTML</td>
<td>content in element</td>
<td>mainDiv.innerHTML is &quot;\n &lt;p&gt;See our &lt;a href...&quot;</td>
</tr>
<tr>
<td>src</td>
<td>URL target of an image</td>
<td>icon.src is &quot;images/borat.jpg&quot;</td>
</tr>
<tr>
<td>href</td>
<td>URL target of a link</td>
<td>theLink.href is &quot;sale.html&quot;</td>
</tr>
</tbody>
</table>
DOM properties for form controls

```html
<input id="sid" type="text" size="7" maxlength="7" />
<input id="frosh" type="checkbox" checked="checked" />
```

```javascript
var sid = document.getElementById("sid");
var frosh = document.getElementById("frosh");
```

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>the text/value chosen by the user</td>
<td>sid.value could be &quot;1234567&quot;</td>
</tr>
<tr>
<td>checked</td>
<td>whether a box is checked</td>
<td>frosh.checked is true</td>
</tr>
<tr>
<td>disabled</td>
<td>whether a control is disabled (boolean)</td>
<td>frosh.disabled is false</td>
</tr>
<tr>
<td>readOnly</td>
<td>whether a text box is read-only</td>
<td>sid.readOnly is false</td>
</tr>
</tbody>
</table>
More about form controls

```html
<select id="captain">
  <option value="kirk">James T. Kirk</option>
  <option value="picard">Jean-Luc Picard</option>
  <option value="cisco">Benjamin Cisco</option>
</select>

<label> <input id="trekkie" type="checkbox" /> I'm a Trekkie</label>
```

- when talking to a text box or `select`, you usually want its `value`
- when talking to a checkbox or radio button, you probably want to know if it's `checked` (true/false)
The `innerHTML` property

```html
<button onclick="addText();">Click me!</button>
<span id="output">Hello </span>
```

```javascript
function addText() {
  var span = document.getElementById("output");
  span.innerHTML += " bro";
}
```

- can change the text inside most elements by setting the `innerHTML` property
Abuse of innerHTML

```javascript
// bad style!
var paragraph = document.getElementById("welcome");
paragraph.innerHTML = "<p>text and <a href="page.html">link</a>";
```

- `innerHTML` can inject arbitrary HTML content into the page
- however, this is prone to bugs and errors and is considered poor style
- we forbid using `innerHTML` to inject HTML tags; inject plain text only
  - (later, we'll see a better way to inject content with HTML tags in it)
Adjusting styles with the DOM

```javascript
objectName.style.propertyName = "value";  // JS
```

```html
<button onclick="colorIt();">Click me!</button>
<span id="fancytext">Don't forget your homework!</span>  // HTML
```

```javascript
function colorIt() {
  var text = document.getElementById("fancytext");
  text.style.color = "#ff5500";
  text.style.fontSize = "40pt";
}
```  // JS

---

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>style</td>
<td>lets you set any CSS style property for an element</td>
</tr>
</tbody>
</table>

- same properties as in CSS, but with camelCasedNames, not names-with-underscores
- examples: backgroundColor, borderLeftWidth, fontFamily
Unobtrusive JavaScript

- JavaScript event code seen previously was obtrusive, in the HTML; this is bad style
- now we'll see how to write unobtrusive JavaScript code
  - HTML with no JavaScript code inside the tags
  - uses the JS DOM to attach and execute all JavaScript event handlers
- allows separation of web site into 3 major categories:
  - content (HTML) - what is it?
  - presentation (CSS) - how does it look?
  - behavior (JavaScript) - how does it respond to user interaction?
Obtrusive event handlers (bad)

- this is bad style (HTML is cluttered with JS code)
- goal: remove all JavaScript code from the HTML body
Attaching an event handler in JavaScript code

<table>
<thead>
<tr>
<th>objectName.onevent = function;</th>
<th>JS</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;button id=&quot;ok&quot;&gt;OK&lt;/button&gt;</td>
<td>HTML</td>
</tr>
<tr>
<td>var okButton = document.getElementById(&quot;ok&quot;); okButton.onclick = okayClick;</td>
<td>JS</td>
</tr>
</tbody>
</table>

- it is legal to attach event handlers to elements' DOM objects in your JavaScript code
  - notice that you do not put parentheses after the function's name
- this is better style than attaching them in the HTML
When does my code run?

**HTML**

```html
<html>
  <head>
    <script src="myfile.js" type="text/javascript"></script>
  </head>
  <body> ... </body>
</html>
```

**JS**

```javascript
var x = 3;
function f(n) { return n + 1; }
function g(n) { return n - 1; }
x = f(x);
```

- your file's JS code runs the moment the browser loads the `script` tag
  - any variables are declared immediately
  - any functions are declared but not called, unless your global code explicitly calls them
- at this point in time, the browser has not yet read your page's `body`
  - none of the DOM objects for tags on the page have been created yet
A failed attempt at being unobtrusive

```html
<html>
  <head>
    <script src="myfile.js" type="text/javascript"></script>
  </head>
  <body>
    <div><button id="ok">OK</button></div>
  </body>
</html>
```

```javascript
var ok = document.getElementById("ok");
ok.onclick = okayClick; // error: null
```

- problem: global JS code runs the moment the script is loaded
- script in `head` is processed before page's `body` has loaded
  - no elements are available yet or can be accessed yet via the DOM
- we need a way to attach the handler after the page has loaded...
The window.onload event

```javascript
function functionName() {
    // code to initialize the page
    ...
}

// run this function once the page has finished loading
window.onload = functionName;
```

- There is a global event called `window.onload` event that occurs at the moment the page body is done being loaded.
- If you attach a function as a handler for `window.onload`, it will run at that time.
An unobtrusive event handler

```html
<button id="ok">OK</button> <!-- (1) -->
```

```js
// called when page loads; sets up event handlers
function pageLoad() {
  var ok = document.getElementById("ok"); // (3)
  ok.onclick = okayClick;
}

function okayClick() {
  alert("booyah"); // (4)
}

window.onload = pageLoad; // (2)
```
Anonymous functions

```javascript
function(parameters) {
    statements;
}
```

- JavaScript allows you to declare **anonymous functions**
- quickly creates a function without giving it a name
- can be stored as a variable, attached as an event handler, etc.
Anonymous function example

```javascript
window.onload = function() {
    var ok = document.getElementById("ok");
    ok.onclick = okayClick;
};

function okayClick() {
    alert("booyah");
}
```

• or the following is also legal (though harder to read and bad style):

```javascript
window.onload = function() {
    document.getElementById("ok").onclick = function() {
        alert("booyah");
    };
};
```
Unobtrusive styling

```javascript
function okayButtonClick() {
    this.style.color = "red";
    this.className = "highlighted";
}
```

CSS

```css
.highlighted { color: red; }
```

• well-written JavaScript code should contain as little CSS as possible
• use JS to set CSS classes/IDs on elements
• define the styles of those classes/IDs in your CSS file
The danger of global variables

- globals can be bad; other code and other JS files can see and modify them

- How many global symbols are introduced by the above code?

- 3 global symbols: count, incr, and reset
Enclosing code in a function

```javascript
function everything() {
  var count = 0;
  function incr(n) {
    count += n;
  }
  function reset() {
    count = 0;
  }
  incr(4);
  incr(2);
  console.log(count);
}

everything();
// call the function to run the code
```

- the above example moves all the code into a function; variables and functions declared inside another function are local to it, not global
- How many global symbols are introduced by the above code?
- 1 global symbol: everything (can we get it down to 0?)
The "module pattern"

```
(function() {
  statements;
})();
```

- wraps all of your file's code in an anonymous function that is declared and immediately called
- 0 global symbols will be introduced!
- the variables and functions defined by your code cannot be messed with externally
Module pattern example

```javascript
(function() {
    var count = 0;
    function incr(n) {
        count += n;
    }
    function reset() {
        count = 0;
    }
    incr(4);
    incr(2);
    console.log(count);
})();
```

• How many global symbols are introduced by the above code?
  • 0 global symbols
JavaScript "strict" mode

"use strict";

your code...

• writing "use strict"; at the very top of your JS file turns on strict syntax checking:
  • shows an error if you try to assign to an undeclared variable
  • stops you from overwriting key JS system libraries
  • forbids some unsafe or error-prone language features
• You should always turn on strict mode for your code in this class!