LECTURE 7: 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
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

```javascript
var icon = document.getElementById("icon01");
icon.src = "kitty.gif";
```
## 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>
</tbody>
</table>
| innerHTML  | content in element   | mainDiv.innerHTML is "\n <p>See our <a href="sale.html" id="saleslink">Sales</a> today!\n <img id="icon" src="images/borat.jpg" alt="Borat" />
</div>" |
| src        | URL target of an image | icon.src is "images/borat.jpg"         |
| href       | URL target of a link  | theLink.href is "sale.html"           |
Text fields: `<input>`

- `input` attributes: `disabled`, `maxlength`, `readonly`, `size`, `value`
- `size` attribute controls onscreen width of text field
- `maxlength` limits how many characters user is able to type into field
Text boxes: `<textarea>`

*a multi-line text input area (inline)*

```html
<textarea rows="4" cols="20">
Type your comments here.
</textarea>
```

- initial text is placed inside textarea tag (optional)
- required rows and cols attributes specify height/width in characters
- optional readonly attribute means text cannot be modified
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>
Adjusting styles with the DOM

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

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

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

<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)

<table>
<thead>
<tr>
<th>HTML</th>
<th>JavaScript</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;button onclick=&quot;okayClick();&quot;&gt;OK&lt;/button&gt;</td>
<td>function okayClick() { alert(&quot;booyah&quot;); }</td>
</tr>
</tbody>
</table>

- 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?

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

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
</table>
| ```javascript
function okayClick() {
  this.style.color = "red";
  this.className = "highlighted";
}
``` | JS |
| ```css
.highlighted { color: red; }
``` | CSS |

- 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

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

- 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!
Checkboxes: `<input>`

Yes/no choices that can be checked and unchecked (inline)

```html
<input type="checkbox" name="lettuce" /> Lettuce
<input type="checkbox" name="tomato" checked="checked" /> Tomato
<input type="checkbox" name="pickles" checked="checked" /> Pickles
```

- none, 1, or many checkboxes can be checked at same time
- when sent to server, any checked boxes will be sent with value on:
- use checked="checked" attribute in HTML to initially check the box
Radio buttons: `<input>`

Sets of mutually exclusive choices (inline)

```html
<input type="radio" name="cc" value="visa" checked="checked" /> Visa
<input type="radio" name="cc" value="mastercard" /> MasterCard
<input type="radio" name="cc" value="amex" /> American Express
```

- grouped by name attribute (only one can be checked at a time)
- must specify a value for each one or else it will be sent as value on
Text labels: `<label>`

```html
<label><input type="radio" name="cc" value="visa" checked="checked" /> Visa</label>
<label><input type="radio" name="cc" value="mastercard" /> MasterCard</label>
<label><input type="radio" name="cc" value="amex" /> American Express</label>
```

- associates nearby text with control, so you can click text to activate control
- can be used with checkboxes or radio buttons
- label element can be targeted by CSS style rules
Drop-down list: `<select>`, `<option>`

*menus of choices that collapse and expand (inline)*

```html
<select name="favoritecharacter">
  <option>Jerry</option>
  <option>George</option>
  <option selected="selected">Kramer</option>
  <option>Elaine</option>
</select>
```

- option element represents each choice
- select optional attributes: disabled, multiple, size
- optional selected attribute sets which one is initially chosen
Using `<select>` for lists

```html
<select name="favoritecharacter[]" size="3" multiple="multiple">
  <option>Jerry</option>
  <option>George</option>
  <option>Kramer</option>
  <option>Elaine</option>
  <option selected="selected">Newman</option>
</select>
```

- optional multiple attribute allows selecting multiple items with shift- or ctrl-click
  - must declare parameter's name with [] if you allow multiple selections
- option tags can be set to be initially selected
Option groups: `<optgroup>`

```html
<select name="favoritecharacter">
  <optgroup label="Major Characters">
    <option>Jerry</option>
    <option>George</option>
    <option>Kramer</option>
    <option>Elaine</option>
  </optgroup>
  <optgroup label="Minor Characters">
    <option>Newman</option>
    <option>Susan</option>
  </optgroup>
</select>
```

- What should we do if we don't like the bold appearance of the optgroups?
Grouping input: `<fieldset>`, `<legend>`

*groups of input fields with optional caption (block)*

```html
<fieldset>
  <legend>Credit cards:</legend>
  <input type="radio" name="cc" value="visa" checked="checked" /> Visa
  <input type="radio" name="cc" value="mastercard" /> MasterCard
  <input type="radio" name="cc" value="amex" /> American Express
</fieldset>
```

- Visa ○ MasterCard ○ American Express

- fieldset groups related input fields, adds a border; legend supplies a caption
### Styling form controls

<table>
<thead>
<tr>
<th>CSS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>element[attribute=&quot;value&quot;]</code> {</td>
<td></td>
</tr>
<tr>
<td>property : value;</td>
<td></td>
</tr>
<tr>
<td>property : value;</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>property : value;</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td>CSS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CSS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>input[type=&quot;text&quot;]</code> {</td>
<td></td>
</tr>
<tr>
<td>background-color: yellow;</td>
<td></td>
</tr>
<tr>
<td>font-weight: bold;</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td>CSS</td>
</tr>
</tbody>
</table>

**Borat**

- attribute selector: matches only elements that have a particular attribute value
- useful for controls because many share the same element (input)
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

<table>
<thead>
<tr>
<th>HTML</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;button onclick=&quot;addText();&quot;&gt;Click me!&lt;/button&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;span id=&quot;output&quot;&gt;Hello &lt;/span&gt;</code></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>JS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>function addText() {</td>
<td></td>
</tr>
<tr>
<td>var span = document.getElementById(&quot;output&quot;);</td>
<td></td>
</tr>
<tr>
<td>span.innerHTML += &quot; bro&quot;;</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
</tbody>
</table>

- 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)