Introduction to Web Programming

Lecture 16: The Document Object Model (DOM); Unobtrusive JavaScript

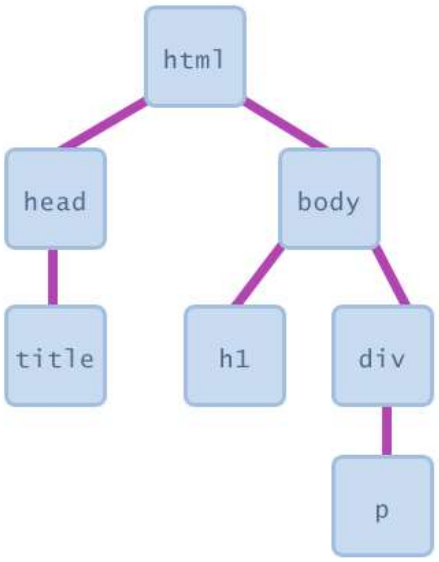
9.2: DOM Element Objects

- 9.1: Global DOM Objects
- **9.2: DOM Element Objects**
- 9.3: The DOM Tree

Document Object Model (DOM)

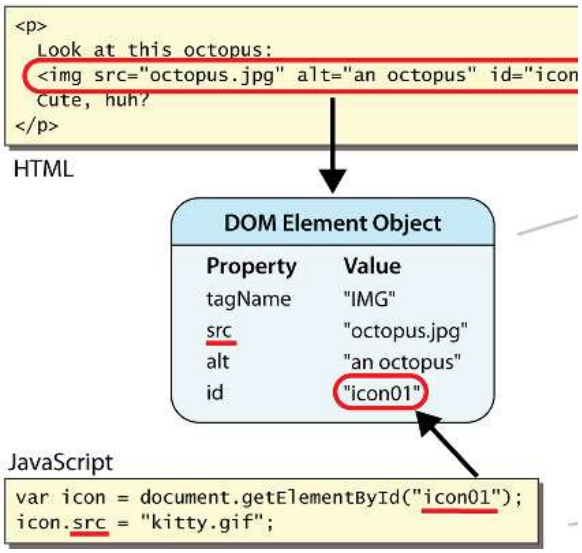
a set of JavaScript objects that represent each element on the page

- 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



Accessing an element: document.getElementById

```
var name = document.getElementById("id");
```

```
<img id="icon01" src="images/octopus.jpg" alt="an animal" />
<button onclick="changeImage();">Click me!</button>
```

```
function changeImage() {
  var octopusImage = document.getElementById("icon01");
  octopusImage.src = "images/kitty.gif";
}
```

Click me!

- document.getElementById returns the DOM object for an element with a given id

Selecting (groups of) DOM objects

- methods in document and other DOM objects:

name	description
getElementById	returns the DOM object for an element with a given id
getElementsByTagName	returns array of descendents with the given tag, such as "div"
getElementsByName	returns array of descendents with the given name attribute (mostly useful for accessing form controls)
querySelector	returns the first element that would be matched by the given CSS selector string
querySelectorAll	returns an array of all elements that would be matched by the given CSS selector string

DOM object properties

```
<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>
```

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

Property	Description	Example
tagName	element's HTML tag	mainDiv.tagName is "DIV"
className	CSS classes of element	mainDiv.className is "foo bar"
innerHTML	content in element	mainDiv.innerHTML is "\n <p>See our <a hr...
src	URL target of an image	icon.src is "images/borat.jpg"
href	URL target of a link	theLink.href is "sale.html"

DOM properties for form controls

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

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

Freshman?

Property	Description	Example
value	the text/value chosen by the user	sid.value could be "1234567"
checked	whether a box is checked	frosh.checked is true
disabled	whether a control is disabled (boolean)	frosh.disabled is false
readOnly	whether a text box is read-only	sid.readOnly is false

More about form controls

```
<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>
```

James T. Kirk ▼ I'm a Trekkie

- 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

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

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

Click me! Hello

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

Abuse of innerHTML

```
// 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

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

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

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

Click me! Don't forget your homework!

Property	Description
style	lets you set any CSS style property for an element

- same properties as in CSS, but with camelCasedNames, not names-with-underscores
  - examples: backgroundColor, borderLeftWidth, fontFamily
  - one exception: float → cssFloat (can you guess why?)

Common DOM styling errors

- many students forget to write `.style` when setting styles

```
var clickMe = document.getElementById("clickme");
clickMe.color = "red";
clickMe.style.color = "red";
```

- style properties are capitalized `likeThis`, not `like-this`

```
clickMe.style.font-size = "14pt";
clickMe.style.fontSize = "14pt";
```

- style properties *must* be set as strings, often with units at the end

```
clickMe.style.width = 200;
clickMe.style.width = "200px";
clickMe.style.padding = "0.5em";
```

  - write exactly the value you would have written in the CSS, but in quotes

9.1.1: Unobtrusive JavaScript

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)

```
<button onclick="okayClick();">OK</button>
```

```
// called when OK button is clicked
function okayClick() {
  alert("booyah");
}
```

OK

- 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

```
objectName.onevent = function;
```

```
<button id="ok">OK</button>
```

```
var okButton = document.getElementById("ok");
okButton.onclick = okayClick;
```

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

```
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>
  <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

```
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

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

// 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)
```

OK

Common unobtrusive JS errors

- event names are all lowercase, not capitalized like most variables

```
window.onLoad = pageLoad;
window.onload = pageLoad;
```

- you shouldn't write () when attaching the handler
  (if you do, it calls the function immediately, rather than setting it up to be called later)

```
ok.onclick = okayClick();
ok.onclick = okayClick;
```

- related: can't directly call functions like `alert`; must enclose in your own function

```
ok.onclick = alert("booyah");
ok.onclick = okayClick;

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

Anonymous functions

```
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

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

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

OK

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

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

Unobtrusive styling

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

```
.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

```
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

```
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

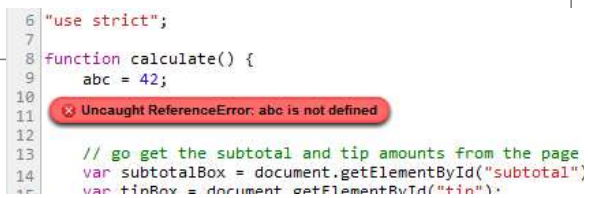
Module pattern example

```
(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!