CSc 337

LECTURE 7: UNOBTRUSIVE JAVASCRIPT

Unobtrusive JavaScript

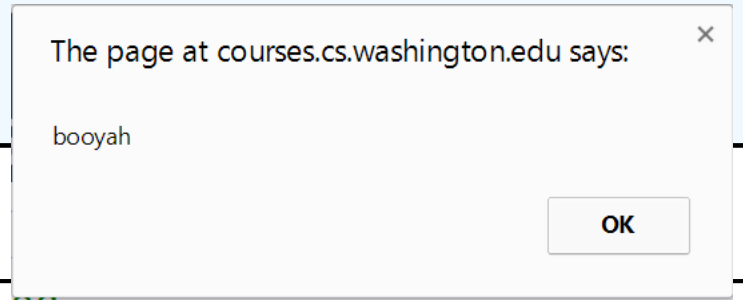
- JavaScript event code seen previously was *obtrusive*, in the HTML; this is bad style
- now we'll see how to write *unobtrusive* JavaScript code
  - HTML with no JavaScript code inside the tags
  - uses the JS DOM to attach and execute all JavaScript event handlers
- allows separation of web site into 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>
```
HTML

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

OK



output

- 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;                                    JS
```

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

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

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

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

- 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>
```

HTML

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

JS

- 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) -->
```

HTML

```
// 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)
```

JS

output

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;
```

  - our **JSLint** checker will catch this mistake

- 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;
}
```
JS

- 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");
}
```

JS

output

- 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";
}
```
JS

```
.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

```
var count = 0;
function incr(n) {
  count += n;
}
function reset() {
  count = 0;
}

incr(4);
incr(2);
console.log(count);
```
JS

- 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;
})();
```
JS

- 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);
})();
```

JS

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

JavaScript "strict" mode

```
"use strict";

your code...
```

```
6  "use strict";
7
8  function calculate() {
9      abc = 42;
10
11 Uncaught ReferenceError: abc is not defined
12
13     // go get the subtotal and tip amounts from the page
14     var subtotalBox = document.getElementById("subtotal");
```

- 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!

Exercise: random colors

Create a page with a button on it. Whenever that button is pressed, turn the background of the page a different random color.

Hints:

- `Math.random()` will give you a random number between 0 and 1
- Multiply that by the range of random numbers you want: `Math.random() * 10`
- Turn that into an integer using `Math.floor()`: `Math.floor(Math.random() * 10)`
- You can create a random color by generating 3 random numbers between 0 and 256 and setting your css as: `background-color: rgb(23, 234, 4)`

Checkboxes: <input>

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

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

HTML

output

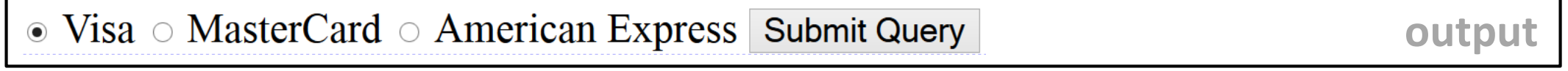
- none, 1, or many checkboxes can be checked at same time
- when sent to server, any checked boxes will be sent with value on:
  - http://webster.cs.washington.edu/params.php?tomato=on&pickles=on
- use checked="checked" attribute in HTML to initially check the box

Radio buttons: <input>

sets of mutually exclusive choices (inline)

```
<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
```

HTML



output

- 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>

```
<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>
```

HTML

output

- 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)

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

HTML

- option element represents each choice
- select optional attributes: disabled, multiple, size
- optional selected attribute sets which one is initially chosen

Using <select> for lists

```
<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>
```

HTML

output

- 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>

```
<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>
```

HTML

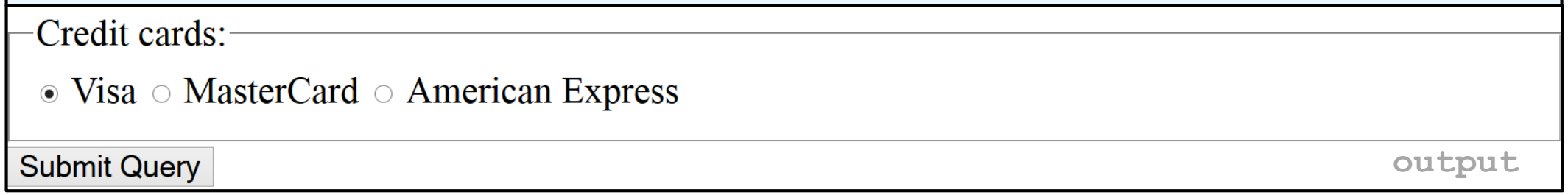
- What should we do if we don't like the bold appearance of the optgroups?

Grouping input: <fieldset>, <legend>

groups of input fields with optional caption (block)

```
<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>
```

HTML



- fieldset groups related input fields, adds a border; legend supplies a caption

Styling form controls

```
element[attribute="value"] {
  property : value;
  property : value;
  ...
  property : value;
}
```
CSS

```
input[type="text"] {
  background-color: yellow;
  font-weight: bold;
}
```
CSS

output

- 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

```
<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>
```

HTML

output

- 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)

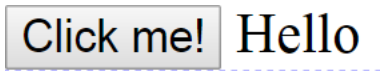
The innerHTML property

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

HTML

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

JS



output

- 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>";
```

JS

- `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)

Exercise: tip calculator

Create a page that allows the user to input a price and a percentage they would like to tip. Your page should show the user the total cost (original price plus tip) when the user clicks a button.

Turn the tip red if the percentage is less than 15.

Hint: use `console.log()` to output variables and help you find bugs in your page