Unobtrusive JavaScript

Lesson outline

- unobtrusive event handlers: why, how, when
- dynamically adjusting styles
- timers: run an event handler after a set amount of time

Unobtrusive JavaScript

- event handling code inside HTML elements is *obtrusive*
  - `<button onclick = "doSomething()">`
  - bad style
- how to write *unobtrusive* code
  - HTML with minimal JavaScript inside
  - uses the DOM to attach and execute all JavaScript functions
- 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?

Attach event handler to DOM element

```
// where element is a DOM element object
element.onevent = function; // syntax structure

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

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

- good style to attach event handlers to DOM objects in your JavaScript code
  - notice that you do **not** put parentheses after the function's name
- Where should we put the above JS code?

JavaScript in a separate file

- JS code can be placed in HTML file's body or head (like CSS)
  - bad style (should separate content, presentation, and behavior)
- script code should be stored in a separate .js file
  - script tag to link the .js files

```
<script src="filename" script>
```

- browser treats multiple script files as a single file
  - share global context.
  - Loading order matters
  - browser executes code when it hits the <script> tag.

code runs the moment the browser loads the script tag

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

- variables are declared immediately
- functions are declared but not called
  - unless your 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>
```

```javascript
// global code (in myfile.js)
document.getElementById("ok").onclick = okClick; // null
function okClick(){alert(" You clicked ok!! " ) }
```

- problem: global JS code runs the moment the script is loaded
- script in head is processed before page's body has loaded
  - button DOM element does not exist yet
- we need a way to attach the handler after the page has loaded...

Exercise

- Implement and try running code on previous slide
  - Implement the okay function
  - Check in the console window for any error messages
  - Ctrl-Shift-I in Chrome or Firefox to get console window

The `window.onload` event

- We need to attach our event handlers after the page is done loading
  - global **event** called `window.onload` event that occurs at that moment
- in `window.onload` handler we attach all the other handlers to run when events occur

```
// this will run once the page has finished loading
function functionName() {
        element.event = functionName;
        element.event = functionName;
        ...
}

window.onload = functionName; // global code
```

An unobtrusive event handler

```
<button id="ok">OK</button>   <!-- look Me, no JavaScript! -->
```

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

function okayClick() {
    alert(" You clicked ok!!");
}

window.onload = pageLoad;  // onload is a browser event
```

Exercise

- Update the previous example to use the onload event and be unobtrusive

Anonymous event handlers

- JavaScript allows you to declare anonymous functions
- Can be stored as a variable, attached as an event handler, etc.
- Keeping unnecessary names out of namespace for performance and safety (and memory management)

```
window.onload = attachHandlers;

function attachHandlers() {
 const okButton =document.getElementById("ok");
 okButton.onclick = okayClick;
};

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

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

Common unobtrusive JS errors

- many students mistakenly write () when attaching the handler

```
window.onload = pageLoad~~()~~; //what will happen?
window.onload = pageLoad;
okButton.onclick = okayClick~~()~~;
okButton.onclick = okayClick;
```

  - IMPORTANT FUNDAMENTAL CONCEPT !!!
    - Function reference versus evaluation

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

```
window.~~onLoad~~ = pageLoad;    //what will happen?
window.onload = pageLoad;
```

Lesson outline

- unobtrusive event handlers: why, how, when
- dynamically adjusting styles
- timers: run an event handler after a set amount of time

Adjusting styles with the DOM

- To change a DOM element style, we use **`style`** property which allows you to set any CSS style property
- It contains same properties as in CSS, but with **camelCasedNames**
  - examples: **`backgroundColor`**, **`borderLeftWidth`**, **`fontFamily`**

```
element.style.fontSize="14pt";
```

Common DOM styling errors

```
Many students forget to write .style when setting styles
    const clickMe = document.getElementById("clickme");
    ~~clickMe.color = "red";~~
    clickMe.style.color = "red";

style properties are capitalized as camelCasedNames
    ~~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";

The below example computes "200px" + 100 + "px" which would evaluate to "200px100px"
    const top = document.getElementById("main").style.top;
    ~~top  = top + 100 + "px";~~
    top = parseInt(top) + 100 + "px";
```

Unobtrusive styling

```
function okayClick() {
 const okButton = document.getElementById("ok");
 okButton~~.style.color = "red";~~
 okButton.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

Exercise

- Use unobtrusive styling to update the previous example to turn the button green and the OK text white after it is clicked

```
.fancy {
background-color: #4CAF50;
color: white;
}
```

Lesson outline

- unobtrusive event handlers: why, how, when
- dynamically adjusting styles
- timers: run an event handler after a set amount of time

JS Timers

```
setTimeout(function, delayMS); // call function after delay in ms
setInterval(function, delayMS); // call function repeatedly every delayMS ms
```

- Both **`setTimeout`** and **`setInterval`** return an ID representing timer
  - pass to `clearTimeout(timerID)` and `clearInterval(timerID)` to stop timer.

Note: If **`function`** has parameters:

```
setTimeout(function, delayMS, param1, param2 ..etc);
```

```
setTimeout(hideBanner, 5000);

function hideBanner() { // called when the timer goes off
        document.getElementById("banner").style.display = "none";
}
```

Common Timer Errors

```
function multiply(a, b) {
      alert(a * b);
}

~~setTimeout(hideBanner(), 5000);~~ // what will happen?
setTimeout(hideBanner, 5000);

~~setTimeout(multiply(num1 , num2), 5000);~~
setTimeout(multiply, 5000, num1, num2);
```

Exercise

Update the previous exercise so that the button will revert to it’s original style after 3 seconds.

Main Point

- Unobtrusive JavaScript promotes separation of web page content into 3 different concerns: content (HTML), presentation (CSS), and behavior(JS) (ala MVC, knower, known, process of knowing)

- JavaScript code runs when the page loads it. Event handlers cannot be assigned until the target elements are loaded. In intelligent systems certain events must happen in an order. Creative intelligence proceeds in an orderly sequential manner.