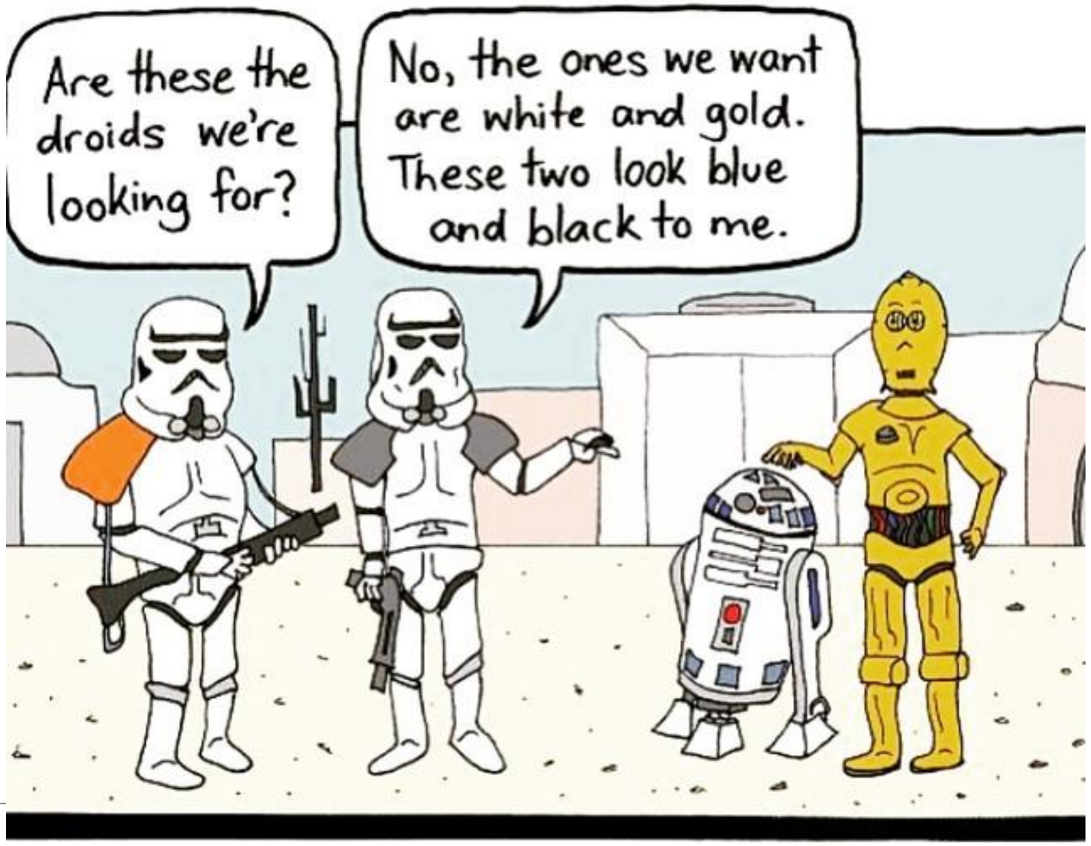
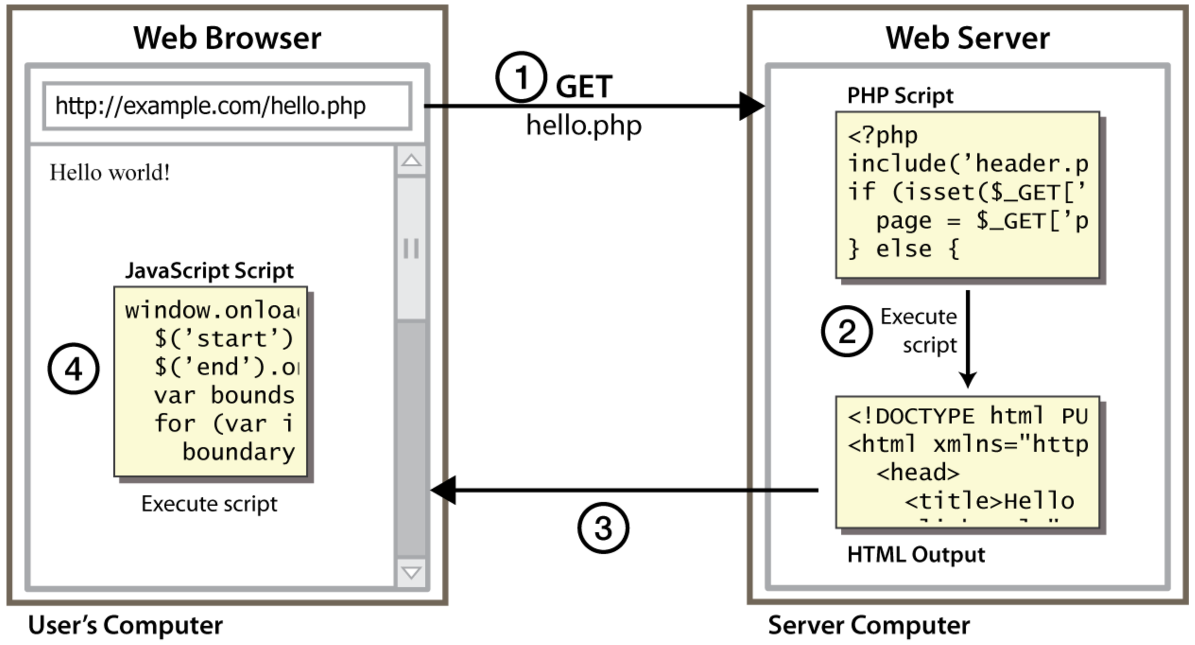
CSc 337

LECTURE 6: JAVASCRIPT


Are these the droids we're looking for?
No, the ones we want are white and gold. These two look blue and black to me.

Client-side scripting



- **client-side script**: code runs in browser *after* page is sent back from server
often this code manipulates the page or responds to user actions

What is JavaScript?

- a lightweight programming language ("scripting language")
- used to make web pages interactive
  - insert dynamic text into HTML (ex: user name)
  - react to events (ex: page load user click)
  - get information about a user's computer (ex: browser type)
  - perform calculations on user's computer (ex: form validation)
- a web standard (but not supported identically by all browsers)
- NOT related to Java other than by name and some syntactic similarities

JavaScript vs. Java

- **interpreted** like Python, not compiled like Java
- more relaxed syntax and rules
  - "looser" data types like Python
  - variables don't need to be declared like Python
  - errors often silent (few exceptions)
- key construct is the **function** rather than the class
  - "first-class" functions are used in many situations
- contained within a web page and integrates with its HTML/CSS content

Java +

= JavaScript

Linking to a JavaScript file: script

```html
<script src="filename" type="text/javascript"></script>        HTML
```

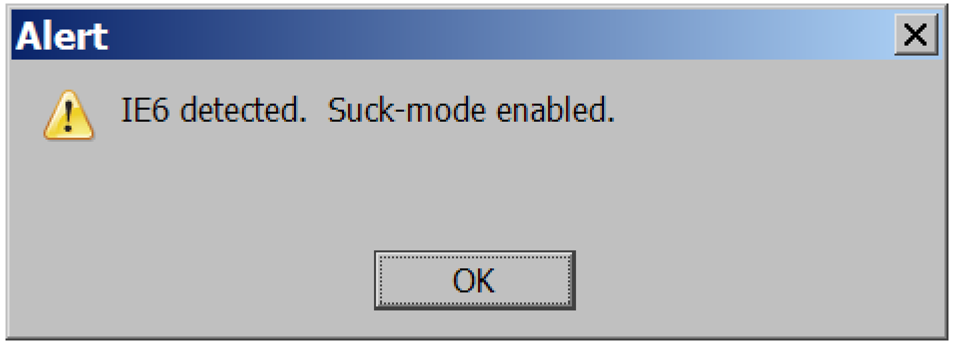
```html
<script src="example.js" type="text/javascript"></script>      HTML
```

- `script` tag should be placed in HTML page's head
- script code is stored in a separate `.js` file
- JS code can be placed directly in the HTML file's body or head (like CSS)
  - but this is bad style (should separate content, presentation, and behavior)

A JavaScript statement: alert

```
alert("message");
```
JS

```
alert("IE6 detected.  Suck-mode enabled.");
```
JS


Alert

IE6 detected. Suck-mode enabled.

output

- a JS command that pops up a dialog box with a message

Variables and types

```
var name = expression;                                    JS
```

```
var age = 32;
var weight = 127.4;
var clientName = "Connie Client";                         JS
```

- variables are declared with the `var` keyword (case sensitive)
- types are not specified, but JS does have types ("loosely typed")
  - `Number`, `Boolean`, `String`, `Array`, `Object`, `Function`, `Null`, `Undefined`
  - can find out a variable's type by calling `typeof`

Number type

```
var enrollment = 99;
var medianGrade = 2.8;
var credits = 5 + 4 + (2 * 3);                                    JS
```

- integers and real numbers are the same type (no `int` vs. `double`)
- same operators: `+ - * / % ++ -- = += -= *= /= %=`
- similar precedence to Java
- many operators auto-convert types: `"2" * 3` is `6`

String type

```
var s = "Connie Client";
var fName = s.substring(0, s.indexOf(" "));   // "Connie"
var len = s.length;                           // 13
var s2 = 'Melvin Merchant';                   // can use "" or ' '
```

- methods: charAt, charCodeAt, fromCharCode, indexOf, lastIndexOf, replace, split, substring, toLowerCase, toUpperCase
  - `charAt` returns a one-letter `String` (there is no `char` type)
- `length` property (not a method as in Java)
- concatenation with `+` : `1` + 1 is `2`, but `"1"` + 1 is `"11"`

More about String

- escape sequences behave as in Java: `\' \" \& \n \t \\`
- to convert between numbers and `String`s:

```
var count = 10;
var s1 = "" + count;                      // "10"
var s2 = count + " bananas, ah ah!";      // "10 bananas, ah ah!"
var n1 = parseInt("42 is the answer");    // 42
var n2 = parseFloat("booyah");            // NaN
```

- to access characters of a `String`, use [*index*] or `charAt`:

```
var firstLetter = s[0];
var firstLetter = s.charAt(0);
var lastLetter = s.charAt(s.length - 1);
```

Comments (*same as Java*)

```
// single-line comment
/* multi-line comment */                                        JS
```

- identical to Java's comment syntax
- recall: 3 comment syntaxes
  - HTML:`<!--` *comment* `-->`
  - CSS/JS:`/*` *comment* `*/`
  - Java/JS:`//` *comment*

for loop (same as Java)

```
for (initialization; condition; update) {
  statements;
}                                                JS
```

```
var sum = 0;
for (var i = 0; i < 100; i++) {
  sum = sum + i;
}                                                JS
```

```
var s1 = "hello";
var s2 = "";
for (var i = 0; i < s.length; i++) {
  s2 += s1[i] + s1[i];
}
// s2 stores "hheelllloo"                        JS
```

Math object

```
var rand1to10 = Math.floor(Math.random() * 10 + 1);

var three = Math.floor(Math.PI);                                          JS
```

- methods: abs, ceil, cos, floor, log, max, min, pow, random, round, sin, sqrt, tan

- properties: E, PI

Logical operators

- Relational: `> < >= <=`
- Logical: `&& || !`
- Equality: `== != === !==`
    - most logical operators automatically convert types. These are all `true`:
        - `5 < "7"`
        - `42 == 42.0`
        - `"5.0" == 5`
    - The `===` and `!==` are strict equality tests; checks both type and value:
        - `"5.0" === 5` is `false`

Boolean type

```
var iLikeJS = true;
var ieIsGood = "IE6" > 0;    // false
if ("web dev is great") {  /* true */ }
if (0) {  /* false */ }                                          JS
```

- any value can be used as a `Boolean`
  - "falsey" values: `0`, `0.0`, `NaN`, `""`, `null`, and `undefined`
  - "truthy" values: anything else
- converting a value into a `Boolean` explicitly:
  - `var boolValue = Boolean(otherValue);`
  - `var boolValue = !!(otherValue);`

Special values: `null` and `undefined`

```
var ned = null;
var benson = 9;
var caroline;

// at this point in the code,
//   ned is null
//   benson's 9
//   caroline is undefined
```
JS

- `undefined` : has not been declared, does not exist
- `null` : exists, but was specifically assigned an empty or `null` value
- Why does JavaScript have both of these?

if/else statement (same as Java)

```
if (condition) {
  statements;
} else if (condition) {
  statements;
} else {
  statements;
}                                                     JS
```

- identical structure to Java's `if/else` statement
- JavaScript allows almost anything as a *condition*

while loops (same as Java)

```
while (condition) {
   statements;
}                                                                JS
```

```
do {
  statements;
} while (condition);                                             JS
```

- `break` and `continue` keywords also behave as in Java but do not use them in this class!

Arrays

```
var name = [];                          // empty array
var name = [value, value, ..., value];  // pre-filled
name[index] = value;                    // store element     PHP
```

```
var ducks = ["Huey", "Dewey", "Louie"];

var stooges = [];          // stooges.length is 0
stooges[0] = "Larry";      // stooges.length is 1
stooges[1] = "Moe";        // stooges.length is 2
stooges[4] = "Curly";      // stooges.length is 5
stooges[4] = "Shemp";      // stooges.length is 5                PHP
```

- two ways to initialize an array
- `length` property (grows as needed when elements are added)

Array methods

```
var a = ["Stef", "Jason"];    // Stef, Jason
a.push("Brian");              // Stef, Jason, Brian
a.unshift("Kelly");           // Kelly, Stef, Jason, Brian
a.pop();                      // Kelly, Stef, Jason
a.shift();                    // Stef, Jason
a.sort();                     // Jason, Stef
                                                          JS
```

- array serves as many data structures: list, queue, stack, ...
- methods: concat, join, pop, push, reverse, shift, slice, sort, splice, toString, unshift
  - push and pop add / remove from back
  - unshift and shift add / remove from front
  - shift and pop return the element that is removed

Splitting strings: split and join

```
var s = "the quick brown fox";
var a = s.split(" ");             // ["the", "quick", "brown", "fox"]
a.reverse();                      // ["fox", "brown", "quick", "the"]
s = a.join("!");                  // "fox!brown!quick!the"          JS
```

- split breaks apart a string into an array using a delimiter
  - can also be used with regular expressions surrounded by /:

    ```
    var a = s.split(/[ \t]+/);
    ```

- join merges an array into a single string, placing a delimiter between them

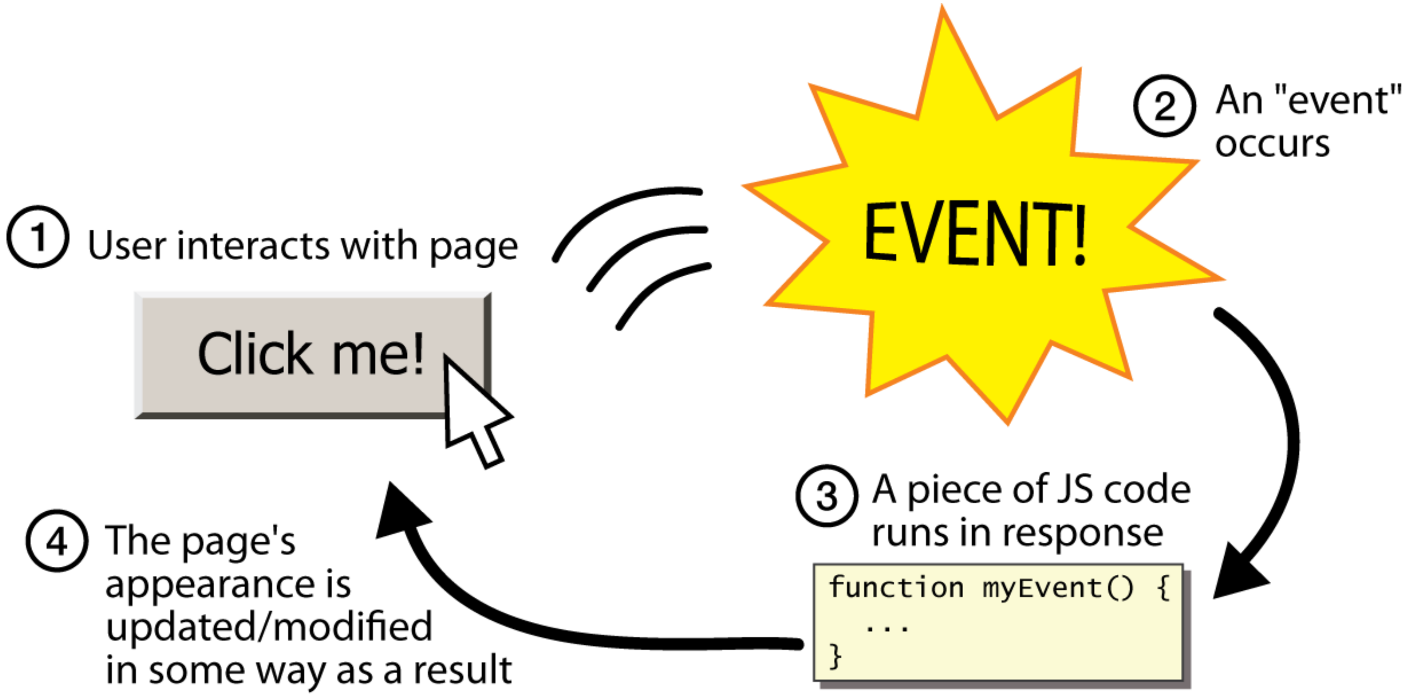
Defining functions

```
function name() {
  statement ;
  statement ;
  ...
  statement ;
}                                                        JS
```

```
function myFunction() {
  alert("Hello!");
  alert("How are you?");
}                                                        JS
```

- the above could be the contents of `example.js` linked to our HTML page
- statements placed into functions can be evaluated in response to user events

Event-driven programming


1 User interacts with page
Click me!
EVENT!
2 An "event" occurs
3 A piece of JS code runs in response
4 The page's appearance is updated/modified in some way as a result

```
function myEvent() {
   ...
}
```

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

Event handlers

```
<element attributes onclick="function();">...                HTML
```

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

Click me!   HTML

- 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>          HTML
```

output

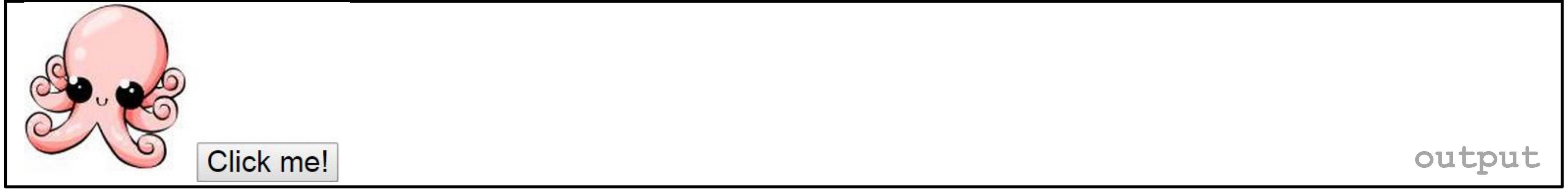
- button's text appears inside tag; can also contain images
- To make a responsive button or other UI control:
  1. choose the control (e.g. button) and event (e.g. mouse click) of interest
  2. write a JavaScript function to run when the event occurs
  3. attach the function to the event on the control

Accessing an element: document.getElementById

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

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

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

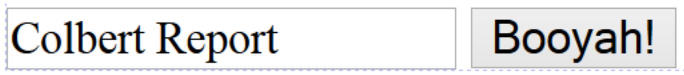


- `document.getElementById` returns the DOM object for an element with a given `id`

<input>

```html
<!-- 'q' happens to be the name of Google's required parameter -->
<input type="text" name="q" value="Colbert Report" />
<input type="submit" value="Booyah!" />
```

HTML

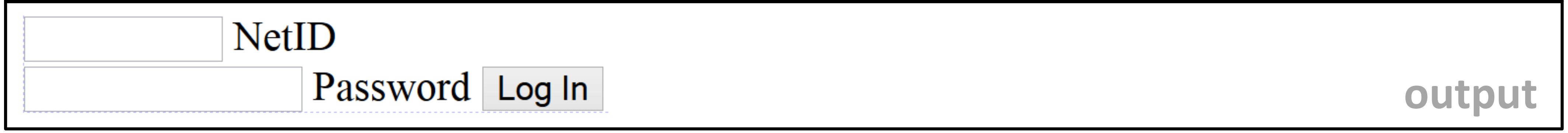


output

- input element is used to create many UI controls
    - an inline element that MUST be self-closed
- name attribute specifies name of query parameter to pass to server
- type can be button, checkbox, file, hidden, password, radio, reset, submit, text, ...
- value attribute specifies control's initial text

Text fields: <input>

```html
<input type="text" size="10" maxlength="8" /> NetID <br />
<input type="password" size="16" /> Password
<input type="submit" value="Log In" />                        HTML
```



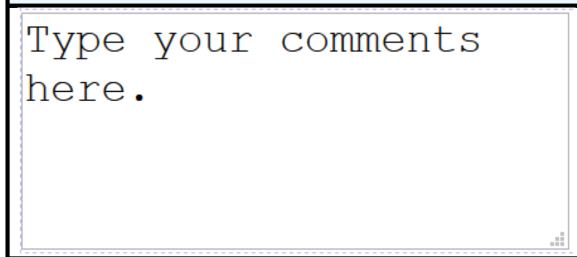
- input attributes: disabled, maxlength, readonly, size, value
- size attribute controls onscreen width of text field
- maxlength limits how many characters user is able to type into field

Text boxes: <textarea>

a multi-line text input area (inline)

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

HTML



```
Type your comments
here.
```

output

- 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

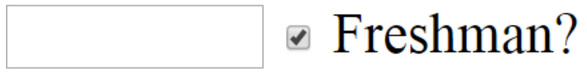
DOM properties for form controls

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

HTML

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

JS



output

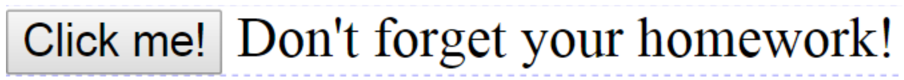
Property	Description	Example
value	the text/value chosen by the user	sid.value could be "1234567"
checked	whether a box is checked	frosh.checked is true
disabled	whether a control is disabled (boolean)	frosh.disabled is false
readOnly	whether a text box is read-only	sid.readOnly is false

Adjusting styles with the DOM

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

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

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



```
Click me! Don't forget your homework!
```

output

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`

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";                                        JS
```

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

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

- 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";                                    JS
```

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

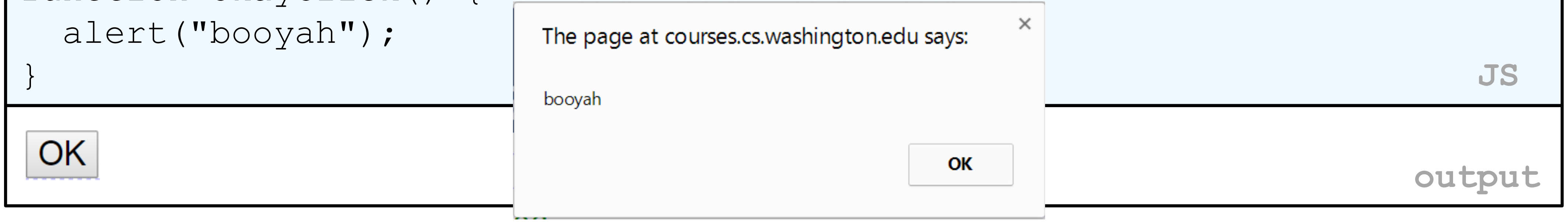
Unobtrusive JavaScript

- 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)

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

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



The page at courses.cs.washington.edu says:

booyah

- 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

```javascript
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>
  <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

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

HTML

```javascript
// 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

```
OK
```

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"); }
```