8.1: Key JavaScript Concepts
8.2: JavaScript Syntax
8.3: Program Logic
8.4: Advanced JavaScript Syntax
- **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
PHP already allows us to create dynamic web pages. Why also use client-side scripting?

- client-side scripting (JavaScript) benefits:
  - **usability**: can modify a page without having to post back to the server (faster UI)
  - **efficiency**: can make small, quick changes to page without waiting for server
  - **event-driven**: can respond to user actions like clicks and key presses
- server-side programming (PHP) benefits:
  - **security**: has access to server's private data; client can't see source code
  - **compatibility**: not subject to browser compatibility issues
  - **power**: can write files, open connections to servers, connect to databases, ...
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**, not compiled
- more relaxed syntax and rules
  - fewer and "looser" data types
  - variables don't need to be declared
  - 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
JavaScript vs. PHP

- similarities:
  - both are interpreted, not compiled
  - both are relaxed about syntax, rules, and types
  - both are case-sensitive
  - both have built-in regular expressions for powerful text processing
- differences:
  - JS is more object-oriented: `noun.verb()`, less procedural: `verb(noun)`
  - JS focuses on UIs and interacting with a document; PHP on HTML output and files/forms
  - JS code runs on the client's browser; PHP code runs on the web server
Event-driven programming

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

*the canonical clickable UI control (inline)*

```html
<button>Click me!</button>
```

- 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
JavaScript functions

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

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

- 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
A JavaScript statement: `alert`

```javascript
alert("message");

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

- a JS command that pops up a dialog box with a message
Linking to a JavaScript file: `<script>`

- `<script src="filename" type="text/javascript"></script>`
- `<script src="example.js" type="text/javascript"></script>`

- `<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)
### Event handlers

To set a function as an event handler in an HTML element, you can use the `onclick` attribute:

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

For example:

```html
<button onclick="myFunction();">Click me!</button>
```

By default, JavaScript functions are executed in a new window, which can be disruptive and annoying. A better user experience would be to display the message on the page instead:

- 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

```html
<button onclick="myFunction();">Click me!</button>
```

```
Click me!
```

- But popping up an **alert** window is disruptive and annoying
  - A better user experience would be to have the message appear on the page...
8.2: JavaScript Syntax

- 8.1: Key JavaScript Concepts
- **8.2: JavaScript Syntax**
- 8.3: Program Logic
- 8.4: Advanced JavaScript Syntax
Variables and types

```javascript
var name = expression;

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

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

- 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
```javascript
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: \\

• to convert between numbers and Strings:

```javascript
var count = 10;
var s1 = "" + count;          // "10"
var s2 = count + " bananas, ah ah ah!"; // "10 bananas, ah 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`:

```javascript
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 */

- identical to Java's comment syntax
- recall: 4 comment syntaxes
  - HTML:  o <!-- comment -->
  - CSS/JS/PHP:  o /* comment */
  - Java/JS/PHP:  o // comment
  - PHP:  o # comment
for loop (same as Java)

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

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

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

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

- methods: `abs`, `ceil`, `cos`, `floor`, `log`, `max`, `min`, `pow`, `random`, `round`, `sin`, `sqrt`, `tan`
- properties: `E`, `PI`
Logical operators

• > < >= <= && || ! == != === !==
• most logical operators automatically convert types:
  ○ 5 < "7" is true
  ○ 42 == 42.0 is true
  ○ "5.0" == 5 is true
• === and !== are strict equality tests; checks both type and value
  ○ "5.0" === 5 is false
if/else statement (same as Java)

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

- identical structure to Java's `if/else` statement
- JavaScript allows almost anything as a `condition`
• 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);
while loops (same as Java)

while (condition) {
    statements;
}

do {
    statements;
} while (condition);

- `break` and `continue` keywords also behave as in Java
Arrays

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

```javascript
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();  // Stef, Jason

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`

```javascript
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"
```

- `split` breaks apart a string into an array using a delimiter
  - can also be used with **regular expressions** surrounded by `/`:
    ```javascript
    var a = s.split(/\s+/);
    ```

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

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

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

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

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

- `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?
alert("message"); // message
confirm("message"); // returns true or false
prompt("message"); // returns user input string
Document Object Model (DOM)

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

- most JS code manipulates elements on an HTML page
- we can 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

HTML

```html
<p>
Look at this octopus:
<img src="octopus.jpg" alt="an octopus" id="icon01" />
Cute, huh?
</p>
```

JavaScript

```javascript
var icon = document.getElementById("icon01");
icon.src = "kitty.gif";
```

- every element on the page has a corresponding DOM object
- access/modify the attributes of the DOM object with `objectName.attributeName`
**DOM object properties**

```html
<div id="main" class="foo bar">
  <p>Hello, <em>very</em> happy to see you!</p>
  <img id="icon" src="images/borat.jpg" alt="Borat" />
</div>
```

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

<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><code>mainDiv.tagName</code> is &quot;DIV&quot;</td>
</tr>
<tr>
<td>className</td>
<td>CSS classes of element</td>
<td><code>mainDiv.className</code> is &quot;foo bar&quot;</td>
</tr>
<tr>
<td>innerHTML</td>
<td>content in element</td>
<td><code>mainDiv.innerHTML</code> is &quot;\n &lt;p&gt;Hello, &lt;em&gt;ve...&quot;</td>
</tr>
<tr>
<td>src</td>
<td>URL target of an image</td>
<td><code>icon.src</code> is &quot;images/borat.jpg&quot;</td>
</tr>
</tbody>
</table>
DOM properties for form controls

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

```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><code>sid.value</code> could be &quot;1234567&quot;</td>
</tr>
<tr>
<td>checked</td>
<td>whether a box is checked</td>
<td><code>frosh.checked</code> is true</td>
</tr>
<tr>
<td>disabled</td>
<td>whether a control is disabled (boolean)</td>
<td><code>frosh.disabled</code> is false</td>
</tr>
<tr>
<td>readOnly</td>
<td>whether a text box is read-only</td>
<td><code>sid.readOnly</code> is false</td>
</tr>
</tbody>
</table>
Accessing elements: `document.getElementById(id)`;

- `document.getElementById(id)` returns the DOM object for an element with a given `id`
- can change the text in most form controls by setting the `value` property
Modifying text inside an element

```javascript
var paragraph = document.getElementById("welcome");
paragraph.innerHTML = "Welcome to our site!"; // change text on page
```

DOM element objects have the following properties:

- `innerHTML`: text and/or HTML tags inside a node
- `textContent`: text (no HTML tags) inside a node
  - simpler than `innerHTML`, but not supported in IE6
- `value`: the value inside a form control
• can change the text inside most elements by setting the `innerHTML` property
• `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
  o (later, we'll see a better way to inject content with HTML tags in it)
Adjusting styles with the DOM

```html
<button id="clickme">Color Me</button>
```

```javascript
window.onload = function() {
    document.getElementById("clickme").onclick = changeColor;
};
function changeColor() {
    var clickMe = document.getElementById("clickme");
    clickMe.style.color = "red";
}
```

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>style</strong></td>
<td>lets you set any CSS style property for an element</td>
</tr>
</tbody>
</table>

- contains same properties as in CSS, but with camelCasedNames
  - examples: `backgroundColor`, `borderLeftWidth`, `fontFamily`
Common DOM styling errors

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

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

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

```javascript
clickMe.style.fontSize = "14pt";
clickMe.style.fontSize = "14pt";
```

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

```javascript
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
JavaScript in HTML body (example)

- JS code can be embedded within your HTML page's **head** or **body**
- runs as the page is loading
- this is considered *bad style* and shouldn't be done in this course
  - mixes HTML content and JS scripts (bad)
  - can cause your page not to validate
• prints specified text into the HTML page
• this is very bad style; this is how newbs program JavaScript:
  ○ putting JS code in the HTML file's **body**
  ○ having that code use `document.write`
  ○ (this is awful style and a poor substitute for server-side PHP programming, which we'll learn later)
The `typeof` function

- given these declarations:
  - function foo() { alert("Hello"); }
  - var a = ["Huey", "Dewey", "Louie"];

- The following statements are true:
  - typeof(3.14) === "number"
  - typeof("hello") === "string"
  - typeof(true) === "boolean"
  - typeof(foo) === "function"
  - typeof(a) === "object"
  - typeof(null) === "object"
  - typeof(undefined) === "undefined"
The arguments array

function example() {
    for (var i = 0; i < arguments.length; i++) {
        alert(arguments[i]);
    }
}

example("how", "are", "you"); // alerts 3 times

- every function contains an array named `arguments` representing the parameters passed
- can loop over them, print/alert them, etc.
- allows you to write functions that accept varying numbers of parameters
The "for each" loop

```javascript
for (var name in arrayOrObject) {
    do something with arrayOrObject[name];
}
```

- loops over every index of the array, or every property name of the object
- using this is actually discouraged, for reasons we'll see later
Arrays as maps

```javascript
var map = [];
map[42] = "the answer";
map[3.14] = "pi";
map["champ"] = "suns";
```

- the indexes of a JS array need not be integers!
- this allows you to store *mappings* between an index of any type ("keys") and value
- similar to Java's `Map` collection or a hash table data structure
Date object

```javascript
var today = new Date(); // today
var midterm = new Date(2007, 4, 4); // May 4, 2007
```

- methods
  - `getDate`, `getDay`, `getMonth`, `getFullYear`, `getHours`, `getMinutes`, `getSeconds`, `getMilliseconds`, `getTime`, `getTimezoneOffset`, `parse`, `setDate`, `setMonth`, `setFullYear`, `setHours`, `setMinutes`, `setSeconds`, `setMilliseconds`, `setTime`, `toString`

- quirks
  - `getFullYear` returns a 2-digit year; use `getFullYear` instead
  - `getDay` returns day of week from 0 (Sun) through 6 (Sat)
  - `getDate` returns day of month from 1 to (# of days in month)
  - `Date` stores month from 0-11 (not from 1-12)
The `eval (evil?)` function

```javascript
eval("JavaScript code");
```

```javascript
eval("var x = 7; x++; alert(x / 2); "); // alerts 4
```

- `eval` treats a String as JavaScript code and runs that code
- this is occasionally useful, but usually a very bad idea
  - if the string's contents come from user input, the user can cause arbitrary code execution
  - can lead to security problems and bugs
Debugging JS code

- Firebug/Chrome JS debugger can set breakpoints, step through code, examine values (Script tab)
- interactive console for typing in arbitrary JS expressions (Console tab)
JSLint: an analyzer that checks your JS code, much like a compiler, and points out common errors
  - CSE3026 version (recommended)
  - original version, by Douglas Crockford of Yahoo!
when your JS code doesn't work, paste it into JSLint first to find many common problems
"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!