JAVASCRIPT
Overview

- Introduction to JavaScript
- The JavaScript Language
- Global DOM Objects (Browser)
- The DOM Tree model
- Unobtrusive JavaScript
Introduction to JavaScript
Client Side Scripting

Web Browser

http://example.com/hello.php
Hello world!

JavaScript Script

window.onload = function() {
  var bounds = $("start").offset();
  var end = $("end").offset();
  var i = 0;
  while (i < bounds.left)
    i = i + 1;

  window.location.href = "hello.php";
}

Web Server

GET
hello.php

<?php
include('header.php');
if (isset($_GET['page'])
  page = $_GET['p']
} else {

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Hello</title>
</head>

Server Computer

User's Computer

Execute script
Why use client-side programming?

Any server side programming language 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
Why use Server-side programming?

- server-side programming 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: a date)
  - react to events (ex: user clicks on a button)
  - get information about a user's computer (ex: browser type)
  - perform calculations on user's computer (ex: form validation)
What is Javascript?

- 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
- contained within a web page and integrates with its HTML/CSS content
Linking to a JavaScript file: `<script>`

- `<script src="filename" 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-driven programming

1. User interacts with page

   ![Click me!]

2. An "event" occurs

   ![EVENT!]

3. A piece of JS code runs in response

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

4. The page's appearance is updated/modified in some way as a result
A JavaScript statement: `alert`

```javascript
alert("IE6 detected.");
```

- a JS command that pops up a dialog box with a message
Event-driven programming

- you are used to programs that start with a main method (or implicit main like in PHP)
- JavaScript programs instead wait for user actions called events and respond to them
- event-driven programming: writing programs driven by user events
- Let's write a page with a clickable button that pops up a "Hello, World" window...
Buttons

- 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

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
Event handlers

- 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
- but popping up an alert window is disruptive and annoying
  - A better user experience would be to have the message appear on the page...

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

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

Look at this octopus:

```
<p>
  <img src="octopus.jpg" alt="an octopus" id="icon01" />
</p>
```

Cute, huh?

DOM Element Object

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tagName</td>
<td>&quot;IMG&quot;</td>
</tr>
<tr>
<td>src</td>
<td>&quot;octopus.jpg&quot;</td>
</tr>
<tr>
<td>alt</td>
<td>&quot;an octopus&quot;</td>
</tr>
<tr>
<td>id</td>
<td>&quot;icon01&quot;</td>
</tr>
</tbody>
</table>

JavaScript

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

- `document.getElementById` returns the DOM object for an element with a given id
- can change the text inside most elements by setting the `innerHTML` property
- can change the text in form controls by setting the `value` property
Change elem. style: `element.style`

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Property or style object</th>
</tr>
</thead>
<tbody>
<tr>
<td>color</td>
<td>color</td>
</tr>
<tr>
<td>padding</td>
<td>padding</td>
</tr>
<tr>
<td>background-color</td>
<td>backgroundColor</td>
</tr>
<tr>
<td>border-top-width</td>
<td>borderTopWidth</td>
</tr>
<tr>
<td>Font size</td>
<td>fontSize</td>
</tr>
<tr>
<td>Font family</td>
<td>fontFamily</td>
</tr>
</tbody>
</table>
function changeText() {
    // grab or initialize text here

    // font styles added by JS:
textbox.style.fontSize = "13pt";
textbox.style.fontFamily = "Comic Sans MS";
textbox.style.color = "red"; // or pink?
}
The JavaScript Language
Variables

var name = expression;

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

- 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

- 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

JS

```javascript
var enrollment = 99;
var medianGrade = 2.8;
var credits = 5 + 4 + (2 * 3);
```
Comments (same as Java)

- identical to Java's comment syntax
- recall: 4 comment syntaxes
  - HTML: <!-- comment -->
  - CSS/JS/PHP: /* comment */
  - Java/JS/PHP: // comment
  - PHP: # comment
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
Special values: null and undefined

```javascript
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
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
Boolean type

```javascript
var iLike190M = true;
var ieIsGood = "IE6" > 0; // false
if ("web development is great") { /* true */ }
if (0) { /* false */ }
```

- 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;`
for loop (same as Java)

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

```
var s1 = "hello";
var s2 = "";
for (var i = 0; i < s1.length; i++) {
    s2 += s1.charAt(i) + s1.charAt(i);
}
// s2 stores "hheellllloo"
```
while loops (same as Java)

while (condition) {
    statements;
}

do {
    statements;
} while (condition);

- break and continue keywords also behave as in Java
Popup boxes

```javascript
alert("message"); // message
confirm("message"); // returns true or false
prompt("message"); // returns user input string
```
Arrays

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

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

- 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
String type

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

- **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)
  - Strings can be specified with "" or "
  - concatenation with +:
    - 1 + 1 is 2, but "1" + 1 is "11"
More about String

- escape sequences as in Java: \\\

- converting 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
```

- accessing the letters of a String:

```javascript
var firstLetter = s[0]; // fails in IE
var firstLetter = s.charAt(0); // does work in IE
var lastLetter = s.charAt(s.length - 1);
```
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 (seen later)
- join merges an array into a single string, placing a delimiter between them
JavaScript Object Hierarchy
The Browser Object Hierarchy
# The Browser Objects (Global DOM)

<table>
<thead>
<tr>
<th>name</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>document</td>
<td>current HTML page and its content</td>
</tr>
<tr>
<td>history</td>
<td>list of pages the user has visited</td>
</tr>
<tr>
<td>location</td>
<td>URL of the current HTML page</td>
</tr>
<tr>
<td>navigator</td>
<td>info about the web browser you are using</td>
</tr>
<tr>
<td>screen</td>
<td>info about the screen area occupied by the browser</td>
</tr>
<tr>
<td>window</td>
<td>the browser window</td>
</tr>
</tbody>
</table>
The window object

- the entire browser window (DOM top-level object)
- technically, all global code and variables become part of the window object properties:
  - document, history, location, name

- methods:
  - alert, confirm, prompt (popup boxes)
  - setInterval, setTimeout, clearInterval, clearTimeout (timers)
  - open, close (popping up new browser windows)
  - blur, focus, moveBy, moveTo, print, resizeBy, resizeTo, scrollBy, scrollTo
The document object (details soon)

- **the current web page and the elements inside it**
- **properties:**
  - anchors, body, cookie, domain, forms, images, links, referrer, title, URL
- **methods:**
  - `getElementById`
  - `getElementsByTagName`
  - `open`, `write`, `writeln`
The location object

- the URL of the current web page
- properties:
  - host, hostname, href, pathname, port, protocol, search
- methods:
  - assign, reload, replace
The navigator object

- *information about the web browser application*

- *properties:*
  - `appName`, `appVersion`, `browserLanguage`,
    `cookieEnabled`, `platform`, `userAgent`

- Some web programmers examine the navigator object to see what browser is being used, and write browser-specific scripts and hacks:

```javascript
if (navigator.appName === "Microsoft Internet Explorer") {
  ...  
}
```
The screen object

- information about the client's display screen
- properties:
  - availHeight, availWidth, colorDepth, height, pixelDepth, width
The history object

- the list of sites the browser has visited in this window

- properties:
  - length

- methods:
  - back, forward, go

- sometimes the browser won't let scripts view history properties, for security
The DOM tree
The DOM tree
Types of DOM nodes

- element nodes (HTML tag)
  - can have children and/or attributes
- text nodes (text in a block element)
- attribute nodes (attribute/value pair)
  - text/attributes are children in an element node
  - cannot have children or attributes
  - not usually shown when drawing the DOM tree

```html
<p>This is a paragraph of text with a <a href="/path/page.html">link in it</a>.</p>
```
Types of DOM nodes

This is a paragraph of text with a link in it.

HTML
## Traversing the DOM tree

<table>
<thead>
<tr>
<th>name(s)</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>firstChild, lastChild</td>
<td>start/end of this node's list of children</td>
</tr>
<tr>
<td>childNodes</td>
<td>array of all this node's children</td>
</tr>
<tr>
<td>nextSibling, previousSibling</td>
<td>neighboring nodes with the same parent</td>
</tr>
<tr>
<td>parentNode</td>
<td>the element that contains this node</td>
</tr>
</tbody>
</table>

**complete list of DOM node properties**  
**browser incompatibility information**
DOM tree traversal example

```html
<p id="foo">This is a paragraph of text with a <a href="/path/to/another/page.html">link</a></p>
```
Q: How many children does the div above have?
A: 3
- an element node representing the <p>
- two text nodes representing "\n\t" (before/after the paragraph)

Q: How many children does the paragraph have?
Q: The a tag?
Selecting groups of DOM objects

methods in document and other DOM objects for accessing descendants:

<table>
<thead>
<tr>
<th>name</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getElementsByTagName</td>
<td>returns array of descendants with the given <em>tag</em>, such as &quot;div&quot;</td>
</tr>
<tr>
<td>getElementsByName</td>
<td>returns array of descendants with the given <em>name attribute</em> (mostly useful for accessing form controls)</td>
</tr>
</tbody>
</table>
Getting all elem. of a certain type

```javascript
var allParas = document.getElementsByTagName("p");
for (var i = 0; i < allParas.length; i++) {
    allParas[i].style.backgroundColor = "yellow";
}
```

```html
<body>
    <p>This is the first paragraph</p>
    <p>This is the second paragraph</p>
    <p>You get the idea...</p>
</body>
```
Combining with `getElementById`

```javascript
var addrParas =
    document.getElementById("address").getElementsByTagName("p");
for (var i = 0; i < addrParas.length; i++) {
    addrParas[i].style.backgroundColor = "yellow";
}
```

```html
<p>This won't be returned!</p>
<div id="address">
    <p>1234 Street</p>
    <p>Atlanta, GA</p>
</div>
```
Creating new nodes

<table>
<thead>
<tr>
<th>name</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>document.createElement(&quot;tag&quot;)</code></td>
<td>creates and returns a new empty DOM node representing an element of that type</td>
</tr>
<tr>
<td><code>document.createTextNode(&quot;text&quot;)</code></td>
<td>creates and returns a text node containing given text</td>
</tr>
</tbody>
</table>

// create a new `<h2>` node
```javascript
var newHeading = document.createElement("h2");
newHeading.innerHTML = "This is a heading";
newHeading.style.color = "green";
```

- merely creating a node does not add it to the page
- you must add the new node as a child of an existing element on the page...
## Modifying the DOM tree

<table>
<thead>
<tr>
<th>name</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>appendChild(node)</code></td>
<td>places given node at end of this node's child list</td>
</tr>
<tr>
<td><code>insertBefore(new, old)</code></td>
<td>places the given new node in this node's child list just before old child</td>
</tr>
<tr>
<td><code>removeChild(node)</code></td>
<td>removes given node from this node's child list</td>
</tr>
<tr>
<td><code>replaceChild(new, old)</code></td>
<td>replaces given child with new node</td>
</tr>
</tbody>
</table>

```javascript
var p = document.createElement("p");
p.innerHTML = "A paragraph!";
document.getElementById("main").appendChild(p);
```
Removing a node from the page

```javascript
function slideClick() {
    var bullets = document.getElementsByTagName("li");
    for (var i = 0; i < bullets.length; i++) {
        if (bullets[i].innerHTML.indexOf("children") >= 0) {
            bullets[i].removeChild();
        }
    }
}
```

- each DOM object has a `removeChild` method to remove its children from the page
DOM versus innerHTML hacking

Why not just code the previous example this way?

```javascript
function slideClick() {
    document.getElementById("thisslide").innerHTML +=
    "<p>A paragraph!</p>";
}
```

- Imagine that the new node is more complex:
  - ugly: bad style (e.g. JS code embedded within HTML)
  - error-prone: must carefully distinguish " and '
  - can only add at beginning or end, not in middle of child list

```javascript
function slideClick() {
    this.innerHTML += "<p style='color: red;" +
    "margin-left: 50px;' " +
    "onclick='myOnClick();'>" +
    "A paragraph!</p>";
}
```
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 minimal JavaScript inside
  - uses the DOM to attach and execute all JavaScript functions
Unobtrusive JavaScript

- allows separation of web site functionality into:
  - content (HTML) - what is it?
  - presentation (CSS) - how does it look?
  - behavior (JavaScript) - how does it respond to user interaction?
Obtrusive event handlers (bad)

- this is bad style (HTML is cluttered with JS code)
- goal: remove all JavaScript code from the HTML body
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

Where should we put the above code?
When does my code run?

- 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
When does my code run?

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

```html
<head>
<script src="myfile.js" type="text/javascript"></script>
</head>
<body> ... </body>
```
A failed attempt to be unobtrusive

- 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
The `window.onload` event

- we want to attach our event handlers right after the page is done loading
- there is a global event called `window.onload` event that occurs at that moment *(after the page is loaded)*
- in `window.onload` handler we attach all the other handlers to run when events occur

```javascript
// 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

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

```js
// called when page loads; sets up event handlers
function pageLoad() {
    document.getElementById("ok").onclick = okayClick;
}
function okayClick() {
    alert("booyah");
}
window.onload = pageLoad; // global code
```
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.
window.onload = function() {
    var okButton = document.getElementById("ok");
    okButton.onclick = okayClick;
};

function okayClick() {
    alert("booyah");
}
The keyword `this`

```javascript
this.fieldName // access field
this.fieldName = value; // modify field
this.methodName(parameters); // call method
```

- All JavaScript code actually runs inside of an object.
- By default, code runs inside the global `window` object:
  - All global variables and functions you declare become part of `window`.
- The `this` keyword refers to the current object.
The keyword `this`

```javascript
function pageLoad() {
    document.getElementById("ok").onclick = okayClick;
    // bound to okButton here
}
function okayClick() { // okayClick knows what DOM object
    this.innerHTML = "booyah"; // it was called on
}
window.onload = pageLoad;
```

- event handlers attached unobtrusively are **bound** to the element
- inside the handler, that element becomes this (rather than the window)
<canvas> element is used to draw graphics, on the fly, via scripting (usually JavaScript).

→ only a container for graphics. You must use a script to actually draw the graphics.
Canvas Resources to Study

1. http://www.w3schools.com/canvas/
2. http://www.w3schools.com/tags/ref_canvas.asp