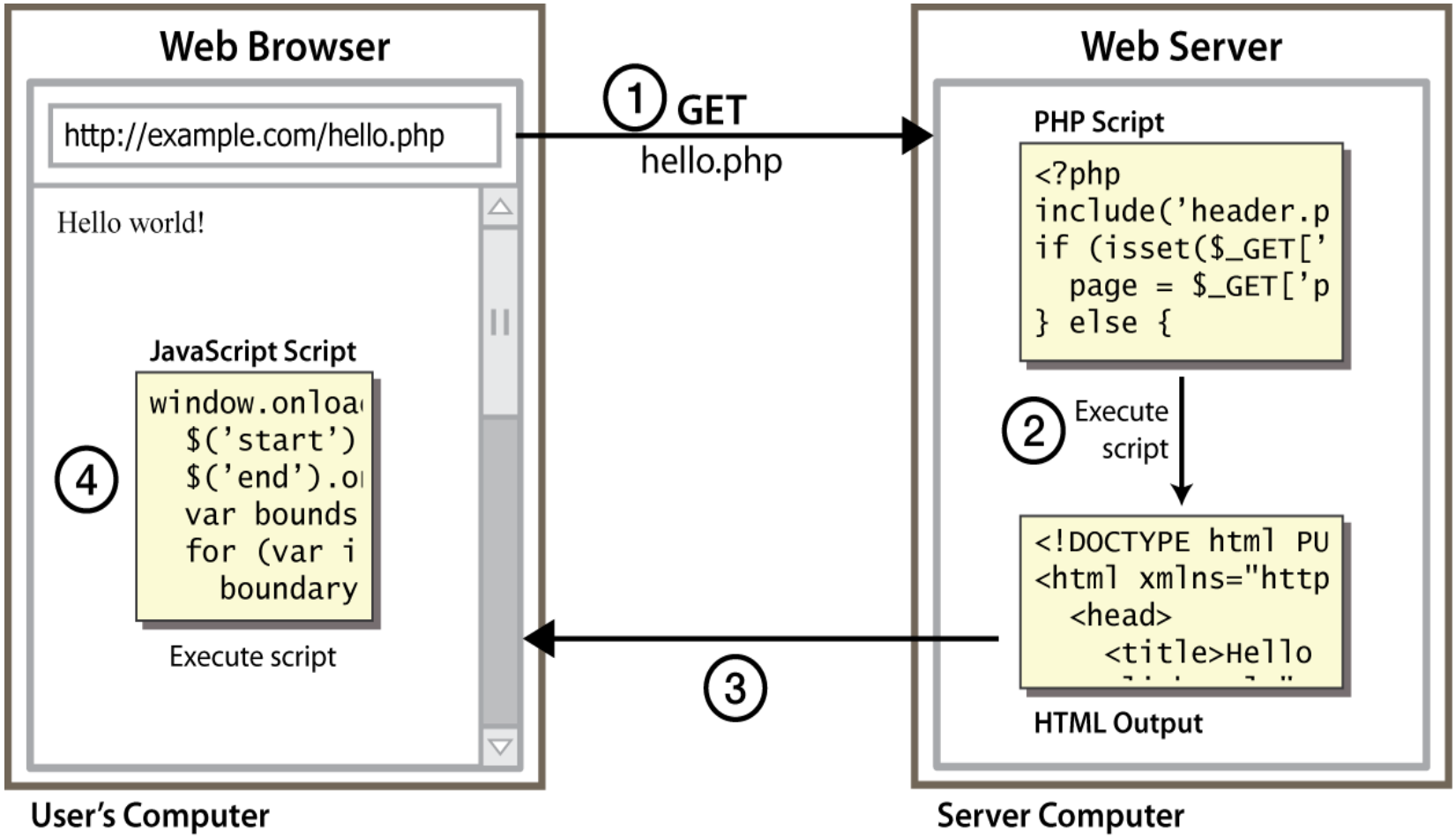
JAVASCRIPT

Overview

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

Introduction to JavaScript

Client Side Scripting



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

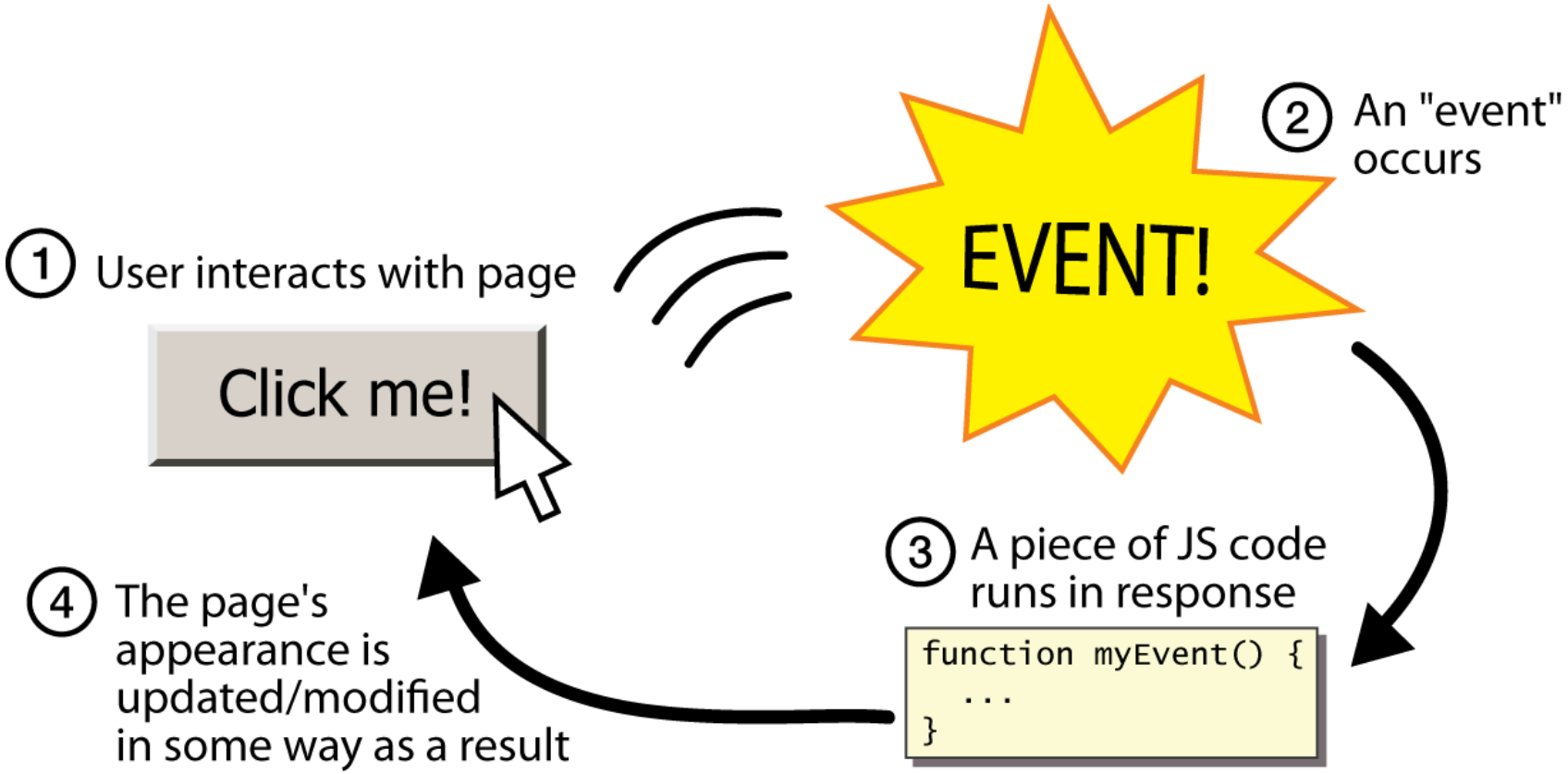
- interpreted, not compiled
- more relaxed syntax and rules
  - fewer and "looser" data types
  - variables don't need to be declared
  - errors often silent (few exceptions)
- key construct is the function rather than the class
- contained within a web page and integrates with its HTML/CSS content

Linking to a JavaScript file: `script`


```
<script src="filename" type="text/javascript"></script>
```
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**)

Event-driven programming



A JavaScript statement: `alert`


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

- 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>Click me!</button>
```
HTML

- 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 ;
}
```
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 handlers


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

HTML

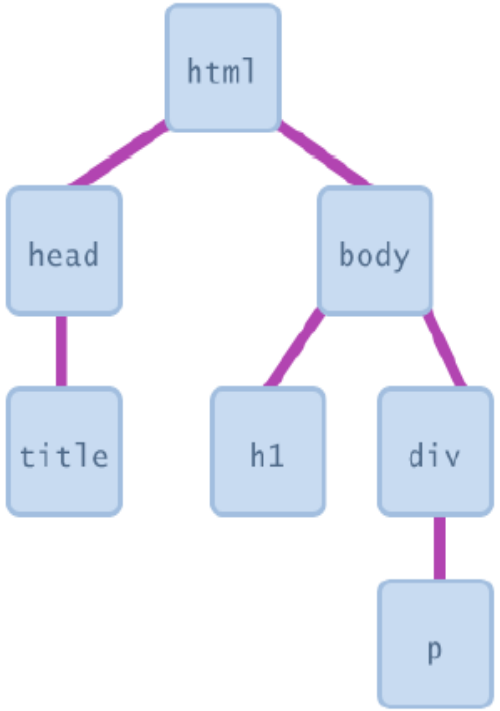
```
<button onclick="myFunction();">Click me!</button>
```

HTML

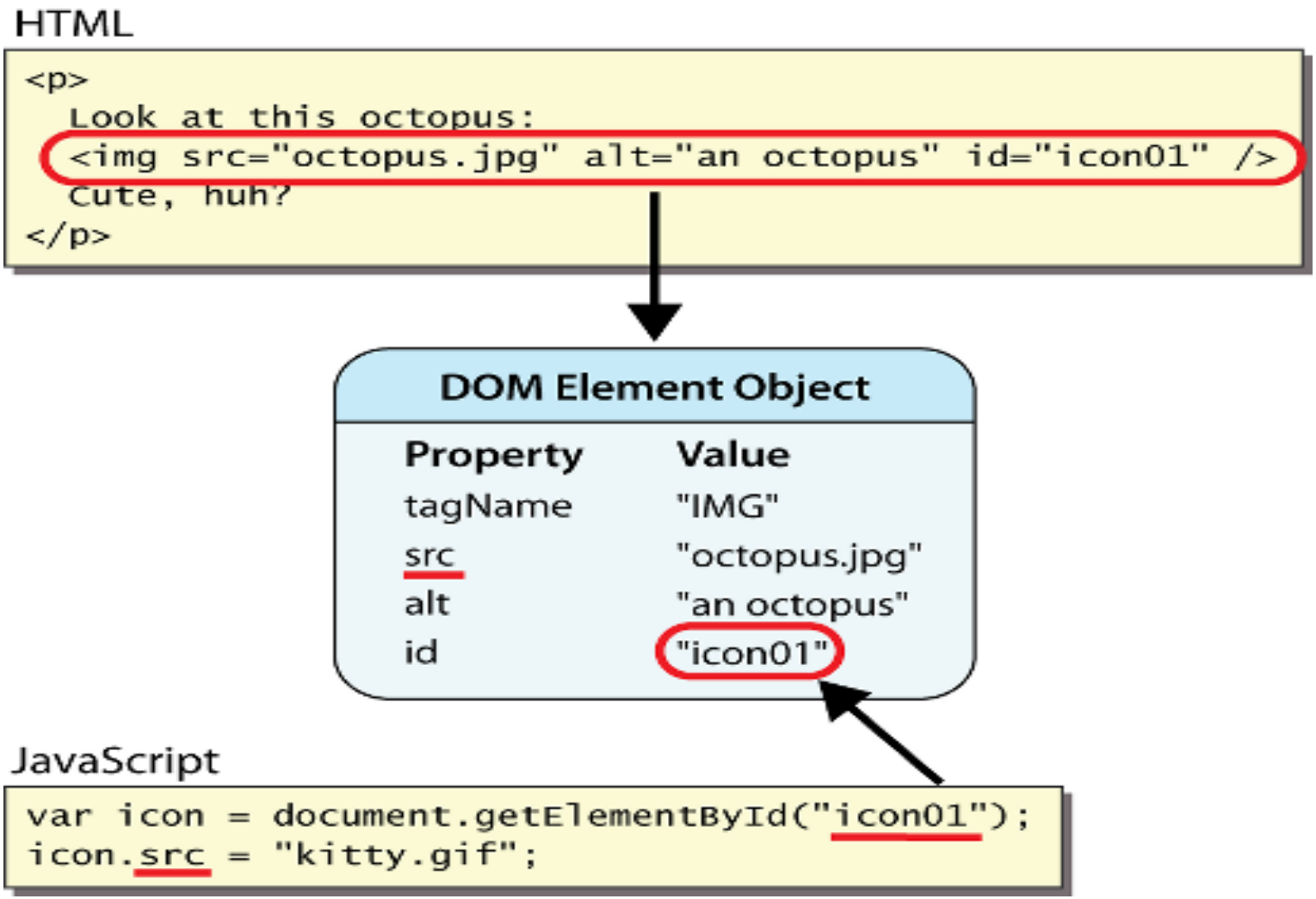
- JavaScript functions can be set as event handlers
  - when you interact with the element, the function will execute
- *onclick* is just one of many event HTML attributes
- but popping up an alert window is disruptive and annoying
  - A better user experience would be to have the message appear on the page...

Document Object Model (DOM)

- 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



Access element: `document.getElementById`


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

```
<button onclick="changeText();">Click me!</button>
<span id="output">replace me</span>
<input id="textbox" type="text" />
```
HTML

```
function changeText() {
        var span = document.getElementById("output");
        var textBox = document.getElementById("textbox");

         textbox.style.color = "red";

}
```
JS

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`

Attribute	Property or style object
color	color
padding	padding
background-color	backgroundColor
border-top-width	borderTopWidth
Font size	fontSize
Font famiy	fontFamily

Preetify


```JS
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;                                    JS
```

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

Comments (same as Java)


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

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

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

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

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)


```
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

Boolean type


```
var iLike190M = true;
var ieIsGood = "IE6" > 0; // false
if ("web devevelopment 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);`

for loop (same as Java)


```
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.charAt(i) + s1.charAt(i);
}
// s2 stores "hheellllloo"
```
JS

while loops (same as Java)


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

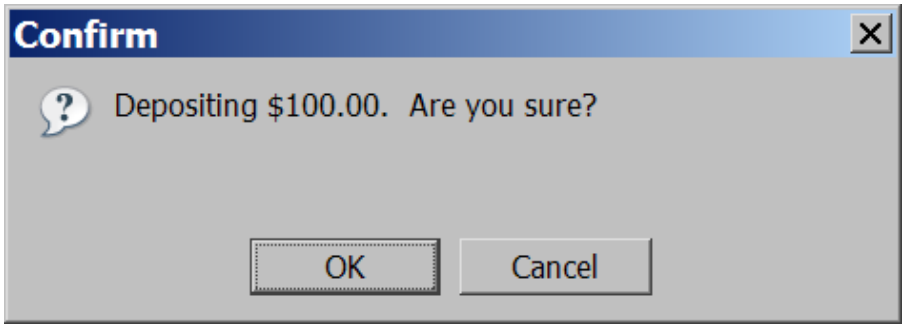
```
do {
    statements;
} while (condition);
```
JS

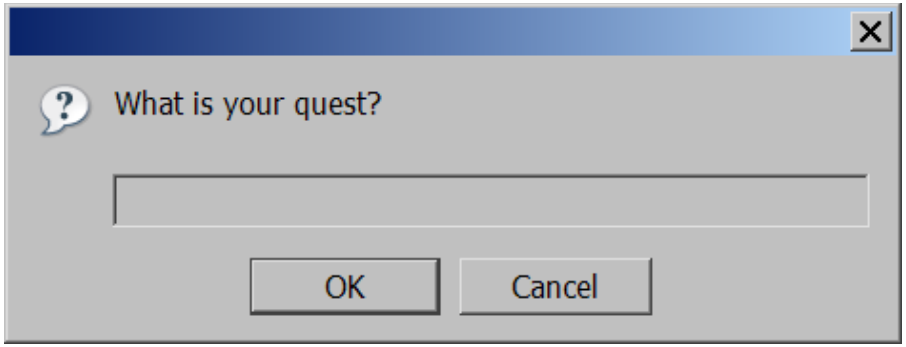
- break and continue keywords also behave as in Java

Popup boxes


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

JS





Arrays


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

JS

```
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
```

JS

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

String type


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

- **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: \' \" \& \n \t \\
- converting between numbers and Strings:

```
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
```
JS

- accessing the letters of a String:

```
var firstLetter = s[0]; // fails in IE
var firstLetter = s.charAt(0); // does work in IE
var lastLetter = s.charAt(s.length - 1);
```
JS

Splitting strings: split and join

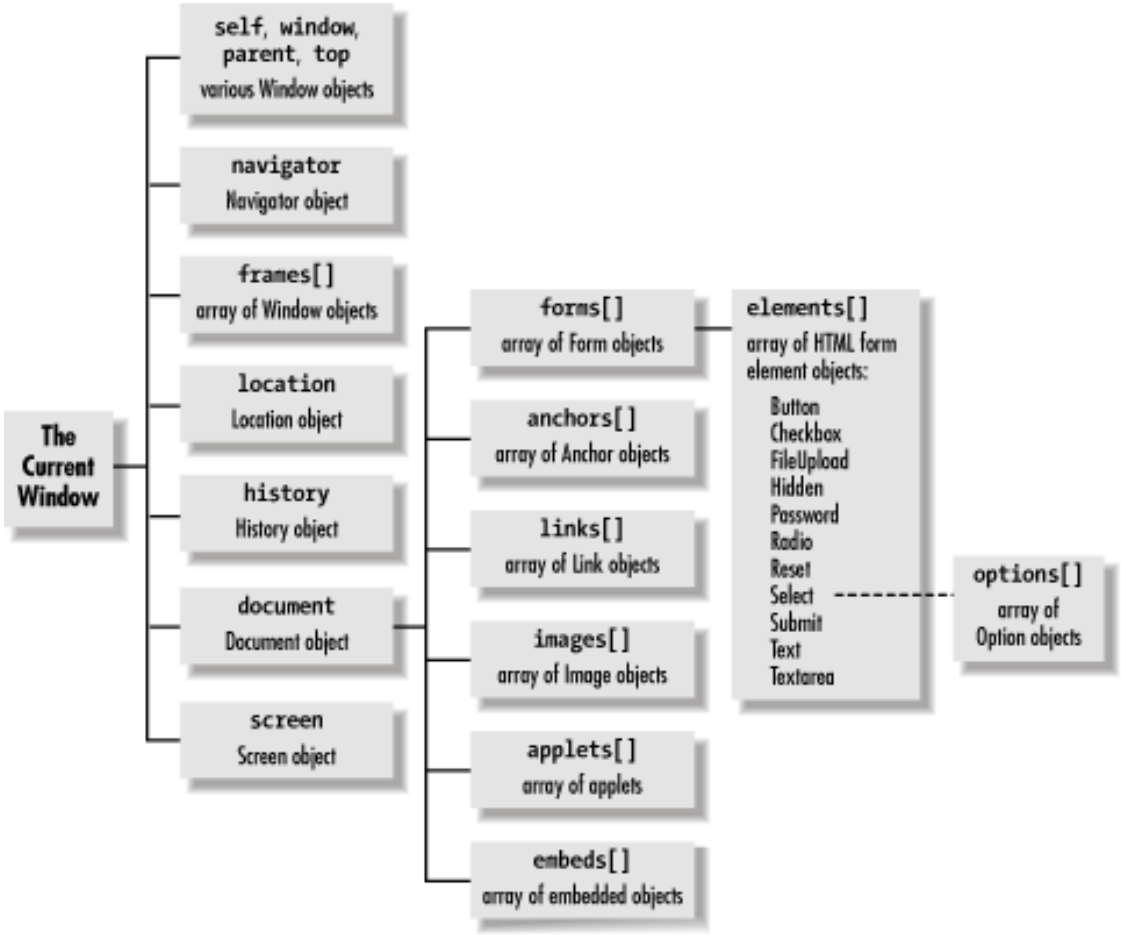

```
var s = "the quick brown fox";
var a = s.split(" "); // ["the", "quick", "brown", "fox"]
a.reverse();          // ["fox", "brown", "quick", "the"]
s = a.join("!");      // "fox!brown!quick!the"
```

JS

- split breaks apart a string into an array using a delimiter
  - can also be used with regular expressions (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)

name	description
document	current HTML page and its content
history	list of pages the user has visited
location	URL of the current HTML page
navigator	info about the web browser you are using
screen	info about the screen area occupied by the browser
window	the browser window

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`
  - `getElementsByName`
  - `getElementsByTagName`
  - `close, 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:

```
if (navigator.appName === "Microsoft Internet Explorer")
{ ...
```
JS

The screen object

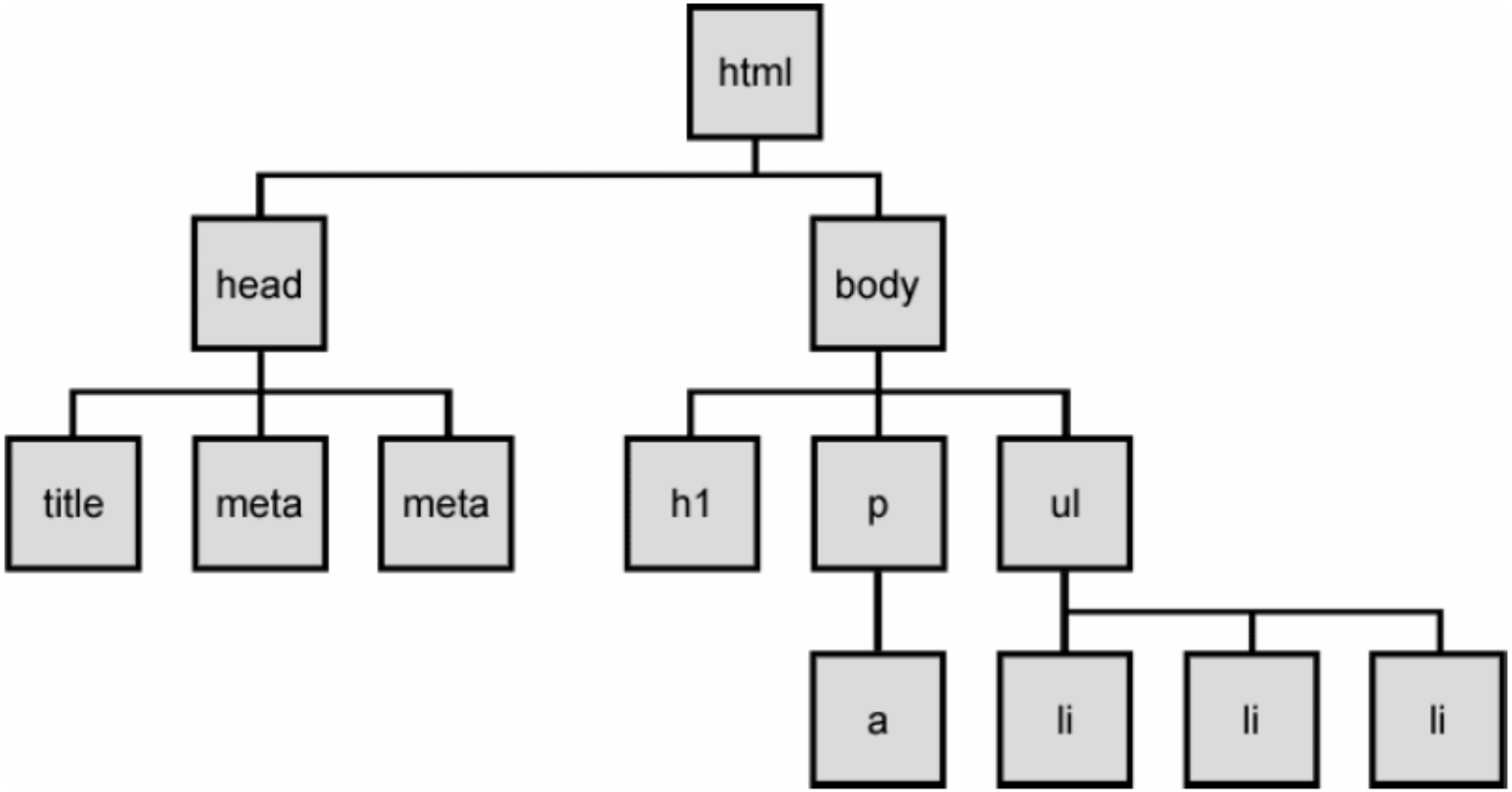
- *information about the client's display screen*
- properties:
  - `availHeight, availWidth, colorDepth, height, pixelDepth, width`

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



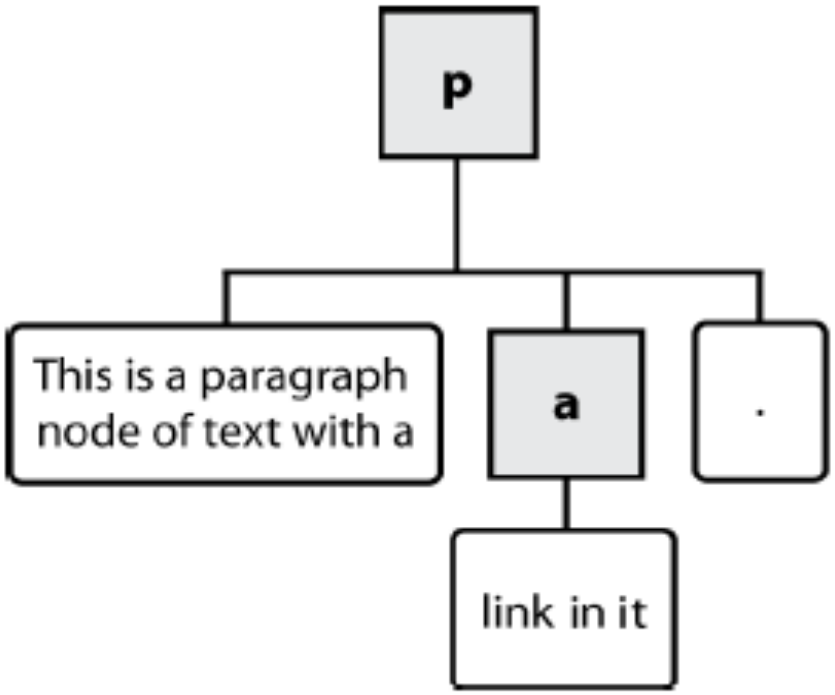
Types of DOM nodes


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

- 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

Types of DOM nodes


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



Traversing the DOM tree

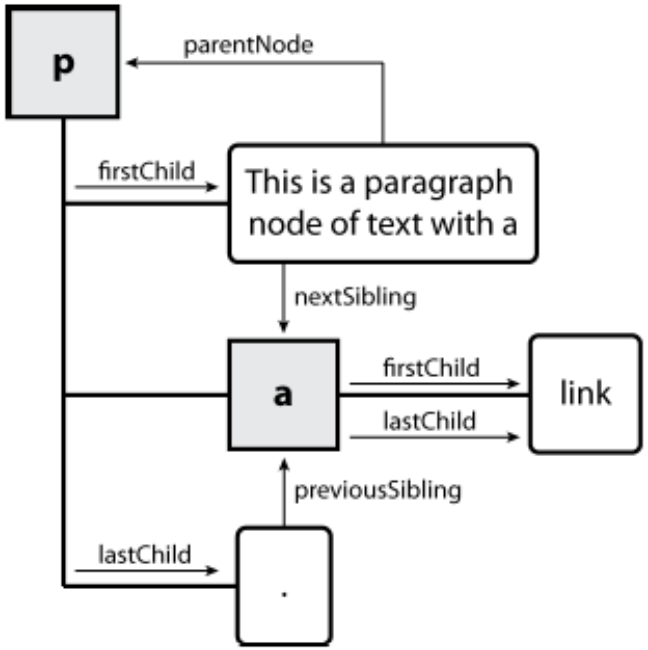
name(s)	description
firstChild, lastChild	start/end of this node's list of children
childNodes	array of all this node's children
nextSibling, previousSibling	neighboring nodes with the same parent
parentNode	the element that contains this node

complete list of DOM node properties
browser incompatiblity information

DOM tree traversal example


```
<p id="foo">This is a paragraph of text with a
<a href="/path/to/another/page.html">link</a>.</p>
```

HTML



Elements vs text nodes


```
<div>
        <p>
                This is a paragraph of text with a
                <a href="page.html">link</a>.
        </p>
</div>
```
HTML

- 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:

name	description
getElementsByTagName	returns array of descendents with the given **tag**, such as "div"
getElementsByName	returns array of descendants with the given **name attribute** (mostly useful for accessing form controls)

Getting all elem. of a certain type


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

```
<body>
      <p>This is the first paragraph</p>
      <p>This is the second paragraph</p>
      <p>You get the idea...</p>
</body>
```
HTML

Combining with getElementById


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

JS

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

HTML

Creating new nodes

name	description
document.createElement("tag")	creates and returns a new empty DOM node representing an element of that type
document.createTextNode("text")	creates and returns a text node containing given text

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

JS

- 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

name	description
appendChild(node)	places given node at end of this node's child list
insertBefore(new, old)	places the given new node in this node's child list just before old child
removeChild(node)	removes given node from this node's child list
replaceChild(new, old)	replaces given child with new node

```
var p = document.createElement("p");
p.innerHTML = "A paragraph!";
document.getElementById("main").appendChild(p);
```

JS

Removing a node from the page


```
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();
      }
   }
}
```

JS

- 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?

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

- 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

```
function slideClick() {
	this.innerHTML += "<p style='color: red; " +
	"margin-left: 50px;' " +
	"onclick='myOnClick();'>" +
	"A paragraph!</p>";
}
```
JS

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)


```
<button id="ok" onclick="okayClick();">OK</button>
```

HTML

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

JS

- 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


```
// where element is a DOM element object
element.event = function;
```
JS

```
document.getElementById("ok").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
- Where should we put the above code?

When does my code run?


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

```
// global code
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

When does my code run?


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

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

- 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

A failed attempt to be unobtrusive


```
<head>
<script src="myfile.js" type="text/javascript"></script>
</head>
<body>
<div><button id="ok">OK</button></div>
```
HTML

```
// global code
document.getElementById("ok").onclick = okayClick;
// error: $("ok") is 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

The `window.onload` event


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

JS

- 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

An unobtrusive event handler


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

HTML

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

JS

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 okButton = document.getElementById("ok");
	okButton.onclick = okayClick;
};
function okayClick() {
	alert("booyah");
}
```

JS

The keyword `this`


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

JS

- 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`


```
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;
```

JS

- event handlers attached unobtrusively are **bound** to the element
- inside the handler, that element becomes this (rather than the window)

Canvas

<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
3. https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial/Basic_animations