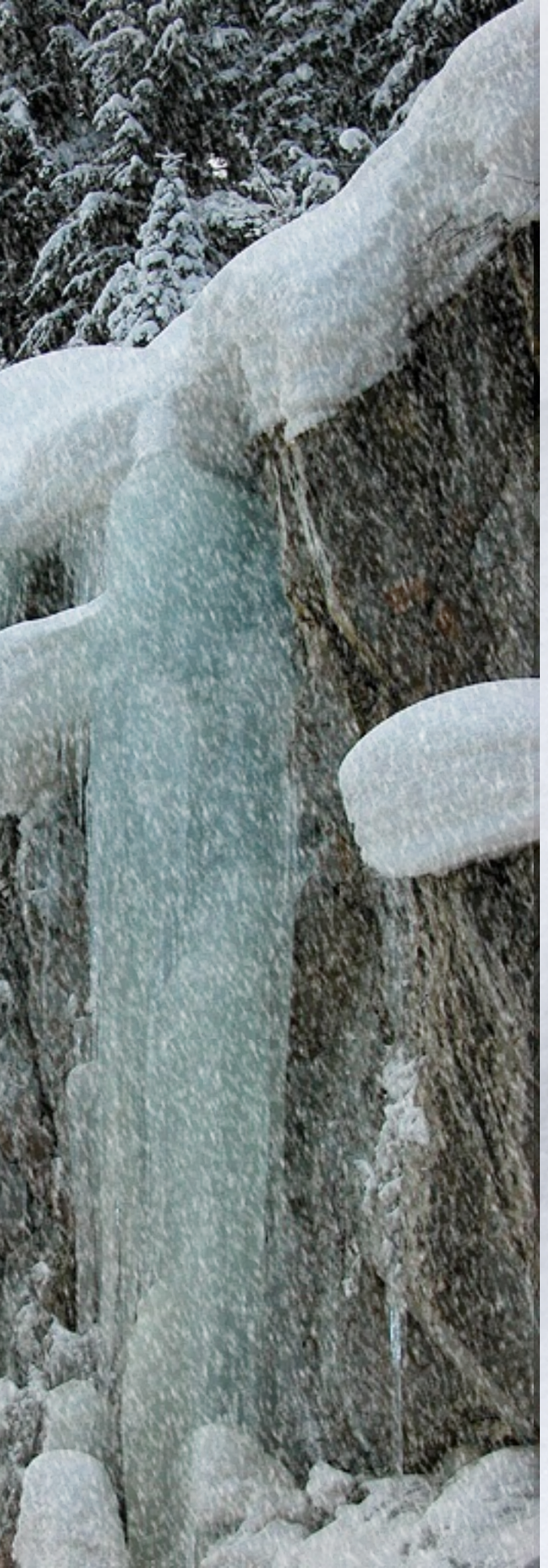
Lecture 5

Basic JavaScript for Client Side Programming

Play

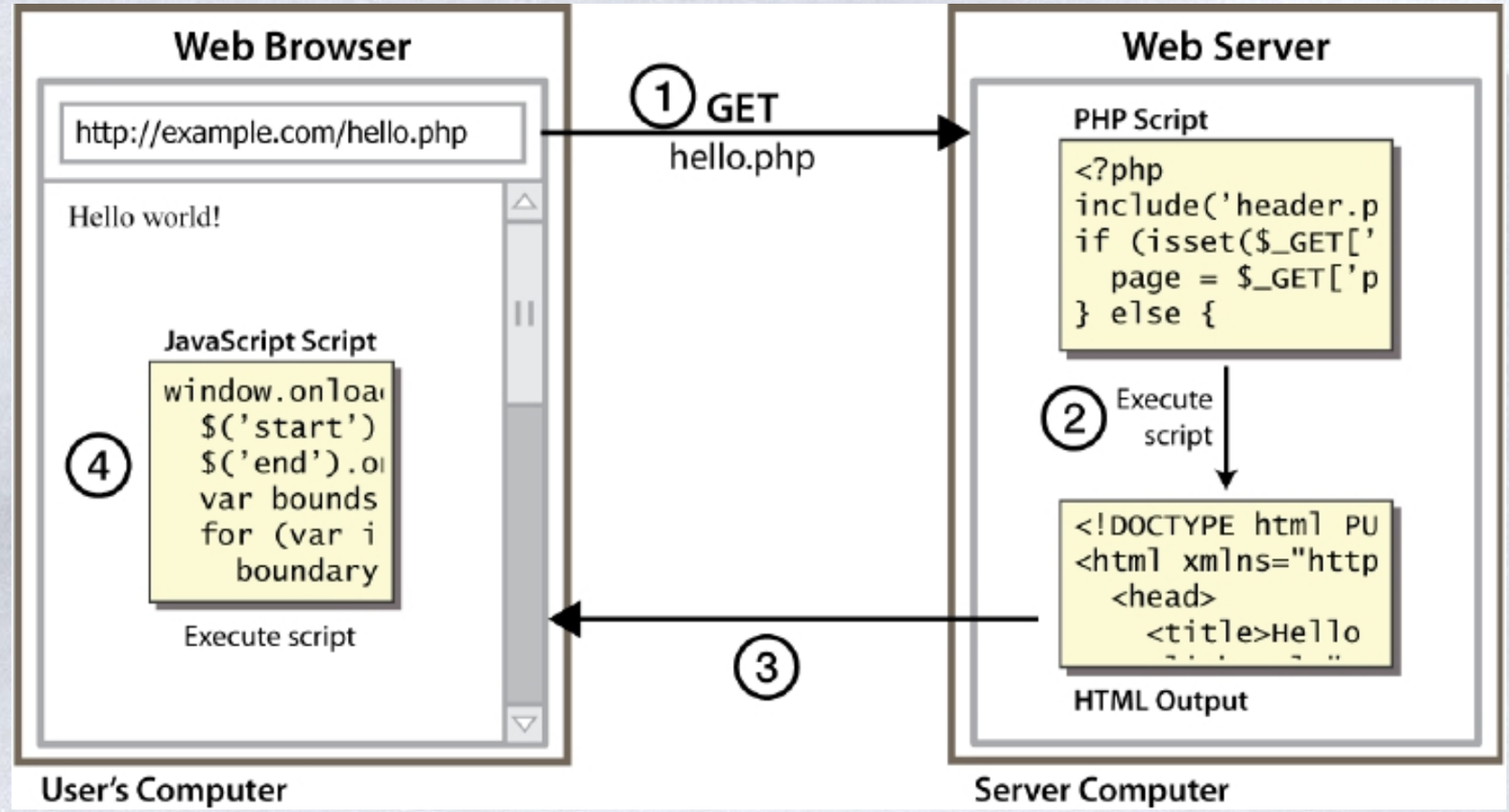


Outline

- Client Side Basics
- Introduction to JavaScript
- JavaScript Basic Syntax
- The DOM tree
- Manipulating DOM

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



Client-Side Programming

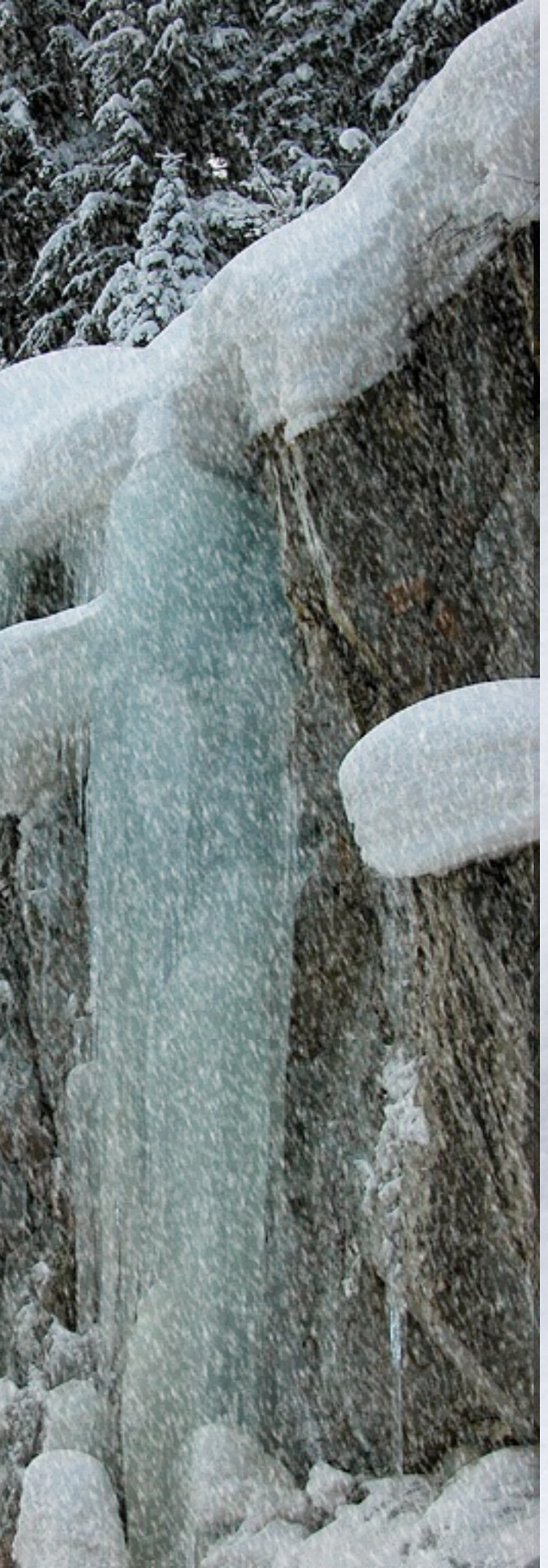
- HTML is good for developing static pages
  - can specify text/image layout, presentation, links, ...
  - Web page looks the same each time it is accessed
  - in order to develop interactive/reactive pages, must integrate programming in some form or another
- client-side programming
  - programs are written in a separate programming (or scripting) language
    - e.g., JavaScript, JScript, VBScript
  - programs are embedded in the HTML of a Web page, with (HTML) tags to identify the program component
    - e.g., <script type="text/javascript"> ... </script>
  - the browser executes the program as it loads the page, integrating the dynamic output of the program with the static content of HTML
  - could also allow the user (client) to input information and process it, might be used to validate input before it's submitted to a remote server

client-side vs. server-side programming

- 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 source code
- server-side programming (PHP) benefits:
  - security: has access to server's private data; client can't see
  - compatibility: not subject to browser compatibility issues
  - power: can write files, open connections to servers, connect to databases, ...

Common Scripting Tasks

- adding dynamic features to Web pages
  - validation of form data (probably the most commonly used application)
  - image rollovers
  - time-sensitive or random page elements
  - handling cookies
- defining programs with Web interfaces
  - utilize buttons, text boxes, clickable images, prompts, etc
- limitations of client-side scripting
  - since script code is embedded in the page, it is viewable to the world
  - for security reasons, scripts are limited in what they can do
    - e.g., can't access the client's hard drive
  - since they are designed to run on any machine platform, scripts do not contain platform specific commands
  - script languages are not full-featured
    - e.g., JavaScript objects are very crude, not good for large project development



Outline

- Client Side Basics
- Introduction to JavaScript
- JavaScript Basic Syntax
- The DOM tree
- Manipulating DOM

Why talk about JavaScript?

- Very widely used, and growing
  - Web pages, AJAX, Web 2.0
  - Increasing number of web-related applications
- Some interesting and unusual features
  - First-class functions - interesting
  - Objects without classes - slightly unusual
  - Powerful modification capabilities - very unusual
    - Add new method to object, redefine prototype, access caller …
- Many security, correctness issues
  - Not statically typed – type of variable may change …
  - Difficult to predict program properties in advance

History

- Javascript Developed by Brendan Eich at Netscape
- JScript created by Microsoft
- IE and Netscape renderings are slightly different
- Standardized by European Computer Manufacturers Association (ECMA)
- http://www.ecma-international. org/ publications /standards/Ecma-262.htm

Essential of JavaScript

- JavaScript is a dialect of the ECMAScript standard and is characterized as a dynamic, weakly typed, prototype-based language with first-class functions. JavaScript was influenced by many languages and was designed to look like Java, but to be easier for non-programmers to work with.
- JavaScript is primarily used in the form of client-side JavaScript, implemented as an integrated component of the web browser, allowing the development of enhanced user interfaces and dynamic websites.

Essential of JavaScript

- JavaScript is a script language
- JavaScript programs are evaluated and executed by JavaScript interpreters / engines
  - Rhino, SpiderMonkey, V8, Squirrelfish
- The mainstream purpose and usage: Exposing objects of an application at runtime, for customizing/embedding user logics.
  - Improve the user interface of a website
  - Make your site easier to navigate
  - Replace images on a page without reload the page
  - Form validation
  - Page embellishments and special effects
  - Dynamic content manipulation
  - Emerging Web 2.0: client functionality implemented at client
  - Many others ...

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
  - comparability: browsers may behave differently upon a JavaScript program
    - different dialects/implementations of the standard (ECMAScript)
    - different objects exposed

Java vs. JavaScript

- Requires the JDK to create the applet
- Requires a Java virtual machine to run the applet
- Applet files are distinct from the XHTML code
- Source code is hidden from the user
- Programs must be saved as separate files and compiled before they can be run
- Programs run on the server side

- Requires a text editor
- Required a browser that can interpret JavaScript code
- JavaScript can be placed within HTML and XHTML
- Source code is made accessible to the user
- Programs cannot write content to the hard disk
- Programs run on the client side

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 user interfaces and interacting with a document;PHP is geared toward HTML output and file/form processing
- JS code runs on the client's browser; PHP code runs on the web server

Using the <script> Tag

- To embed a client-side script in a Web page, use the element:

```
<script type="text/javascript" >
    script commands and comments
</script>
```

- To access an external script, use:

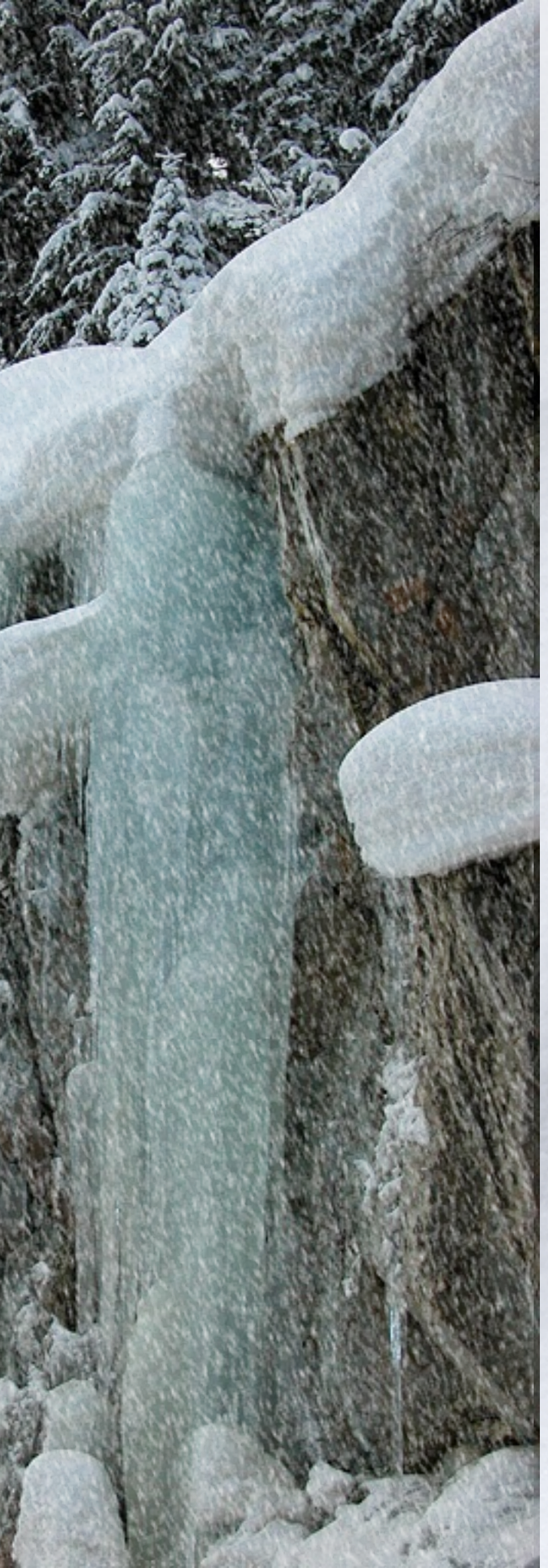
```
<script src="url" type="text/javascript">
    script commands and comments
</script>
```

Linking to a JavaScript file: 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)

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

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



Outline

- Client Side Basics
- Introduction to JavaScript
- JavaScript Basic Syntax
- The DOM tree
- Manipulating DOM

Language basics

- JavaScript is case sensitive
  - HTML is not case sensitive; onClick, ONCLICK, ... are HTML
- Statements terminated by returns or semi-colons (;)
  - x = x+1; same as x = x+1
  - Semi-colons can be a good idea, to reduce errors
- "Blocks"
  - Group statements using { ... }
  - Not a separate scope, unlike other languages
- Variables
  - Define a variable using the var statement
  - Define implicitly by its first use, which must be an assignment
    - Implicit definition has global scope, even if it occurs in nested scope?

Objects

- An object is a collection of named properties
  - Simple view: hash table or associative array
  - Can define by set of name:value pairs
    - objBob = {name: “Bob", grade: 'A', level: 3};
  - New members can be added at any time
    - objBob.fullname = 'Robert';
  - Can have methods, can refer to this
- Arrays, functions regarded as objects
  - A property of an object may be a function (=method)

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

```JS
// single-line comment

/* multi-line comment */
```

Variables and types

- 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

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

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

Number type

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

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

String type

- 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"

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

More about String

- escape sequences behave 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

Boolean type

- 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);

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

Special values: null, NaN, undefined

- NaN
- undefined
- null

Math object

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

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

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

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

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

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

while loops (same as Java)

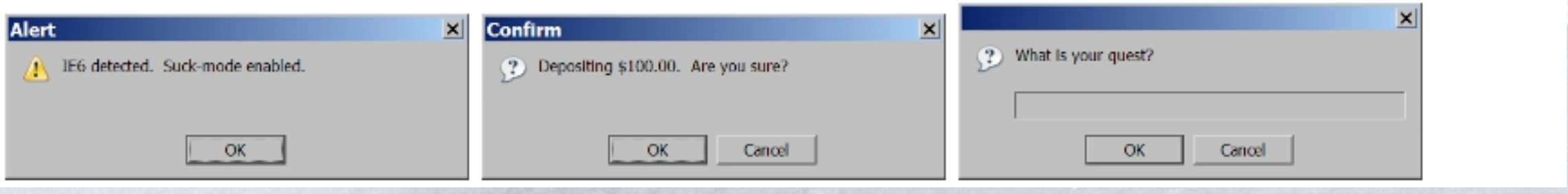
- break and continue keywords also behave as in Java

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

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

Popup boxes

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



Arrays

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

```
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

- 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

```
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

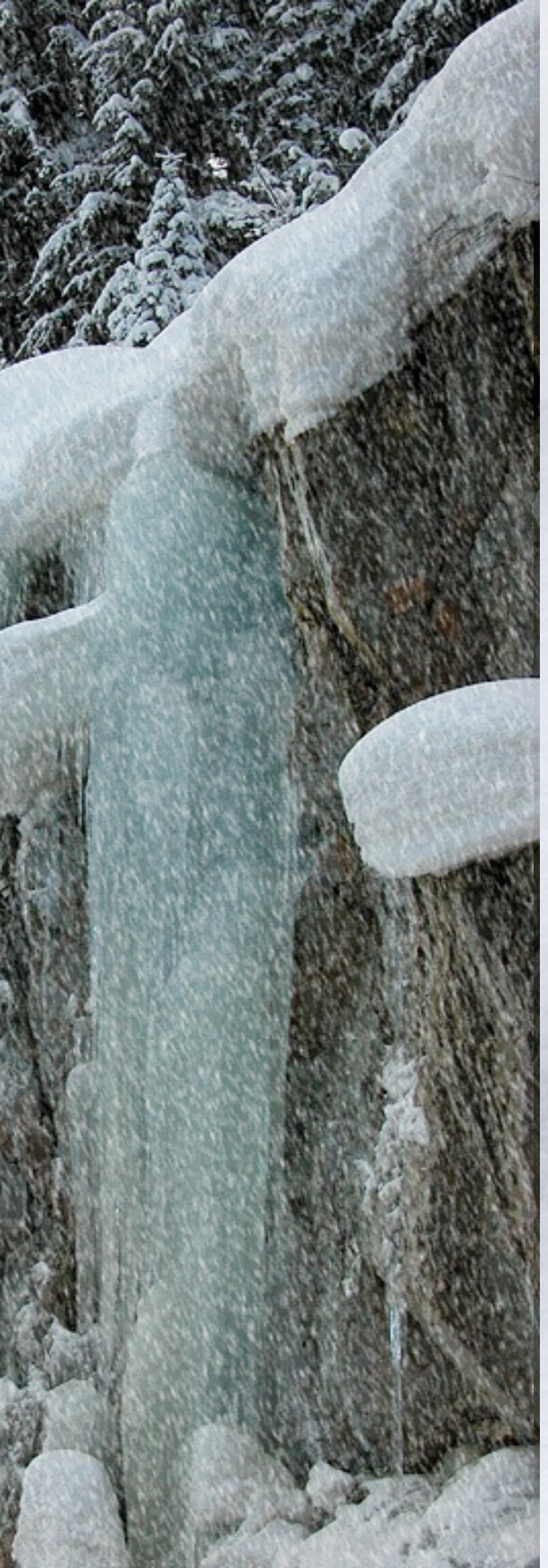
Splitting strings: split and join

- 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

```
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

functions

- Declarations can appear in function body
  - Local variables, “inner” functions
- Parameter passing
  - Basic types passed by value, objects by reference
- Call can supply any number of arguments
  - functionname.length : # of arguments in definition
  - functionname.arguments.length : # args in call
- “Anonymous” functions (expressions for functions)
  - (function (x,y) {return x+y}) (2,3);



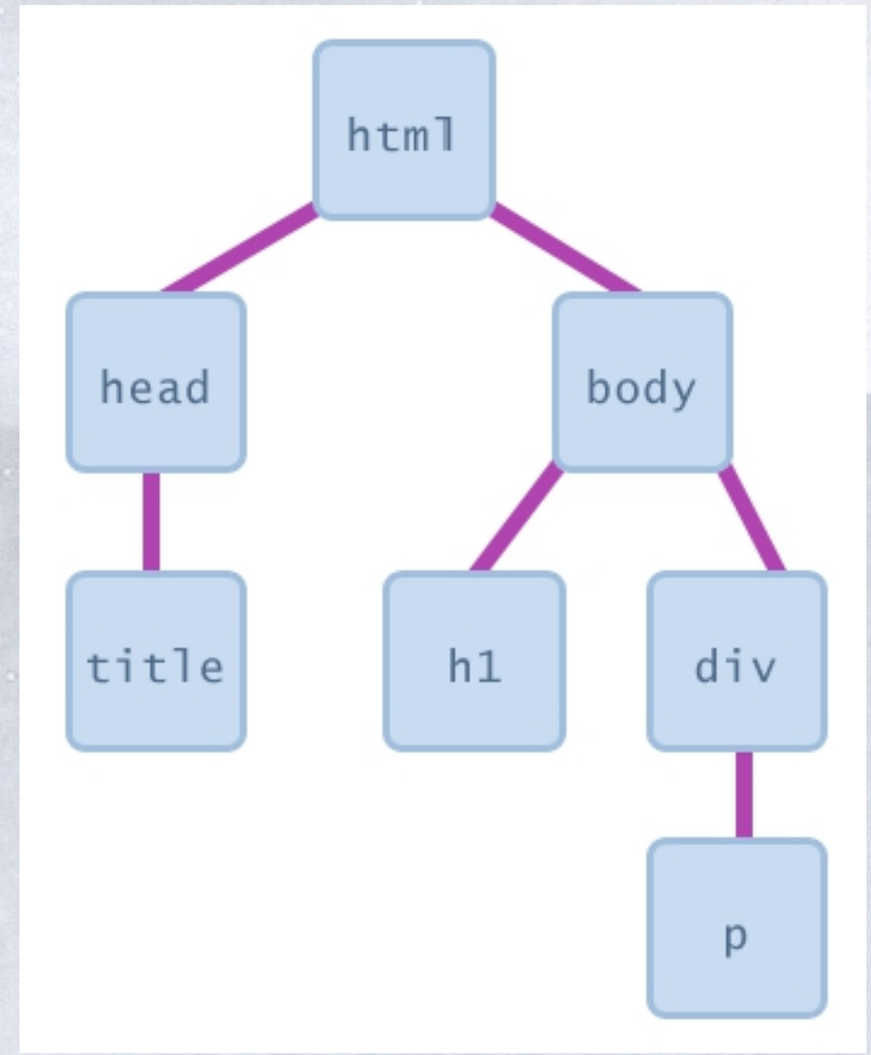
Outline

- Client Side Basics
- Introduction to JavaScript
- JavaScript Basic Syntax
- DOM tree
- Manipulating DOM

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

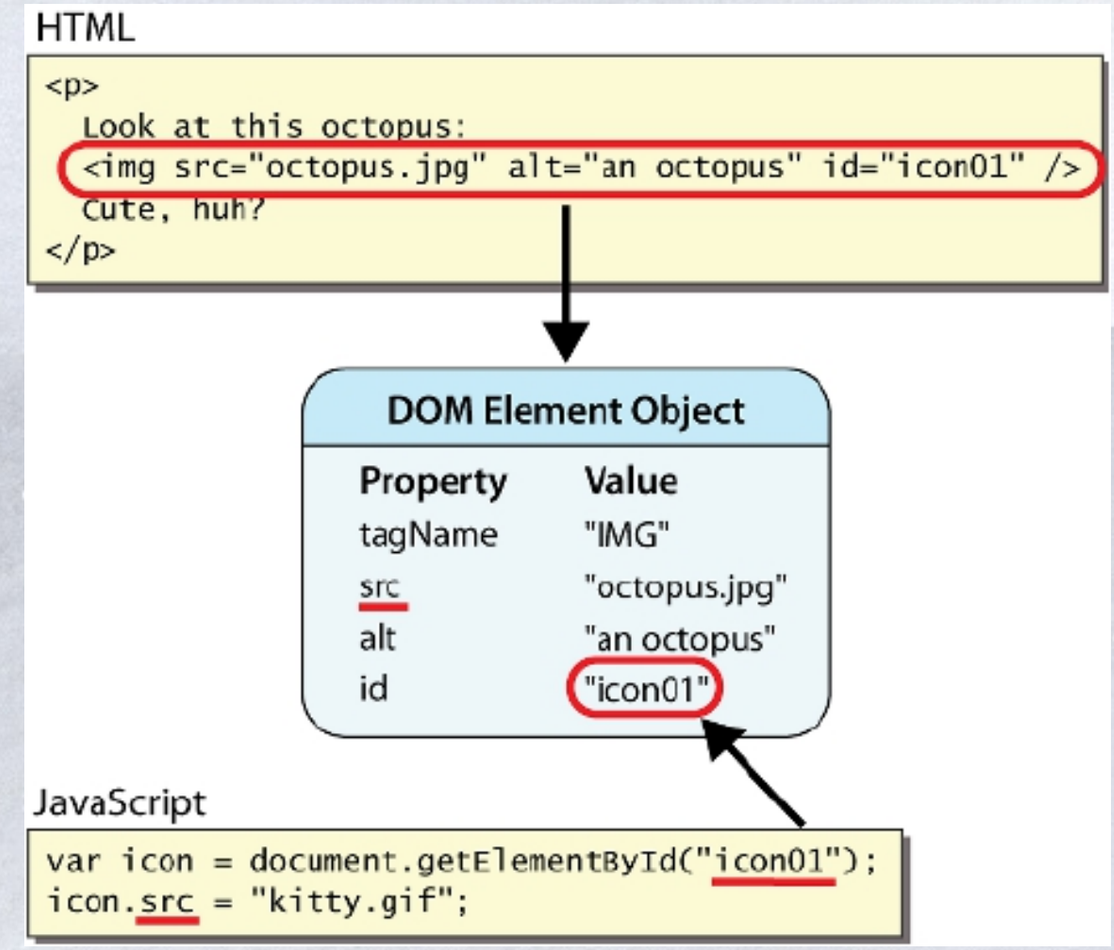


DOM element

- every element on the page has a corresponding DOM object

- access/modify the attributes of the DOM object with objectName.attributeName

- in fact, browsers evaluate a Web page into corresponding DOM objects at runtime



Accessing elements: 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.value = span.innerHTML;
  span.innerHTML = "Hello, how are you?";
}
```
JS

Click me! replace me

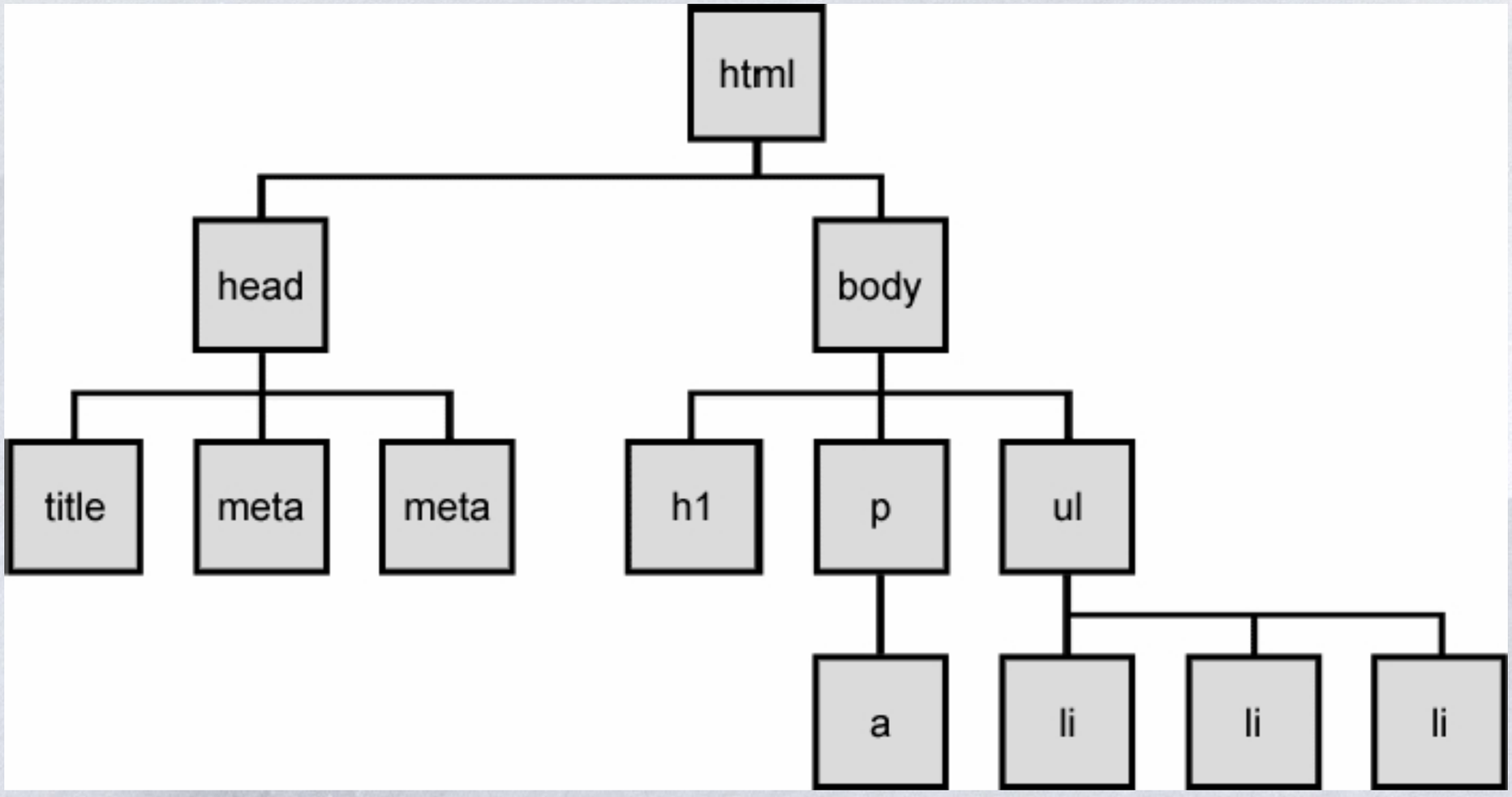
output

- 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

Complex DOM manipulation problems

- How would we do each of the following in JavaScript code? Each involves modifying each one of a group of elements ...
  - When the Go button is clicked, reposition all the divs of class puzzle to random x/y locations.
  - When the user hovers over the maze boundary, turn all maze walls red.
  - Change every other item in the ul list with id of TAs to have a gray background.

The DOM tree



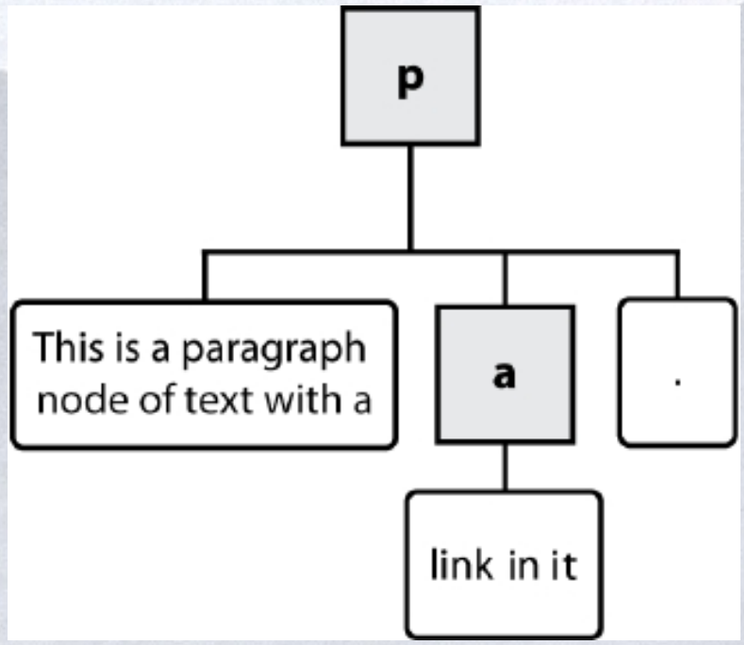
- The elements of a page are nested into a tree-like structure of objects – DOM tree
  - the DOM has properties and methods for traversing this tree

Type of DOM nodes

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

- 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



Traversing the DOM tree

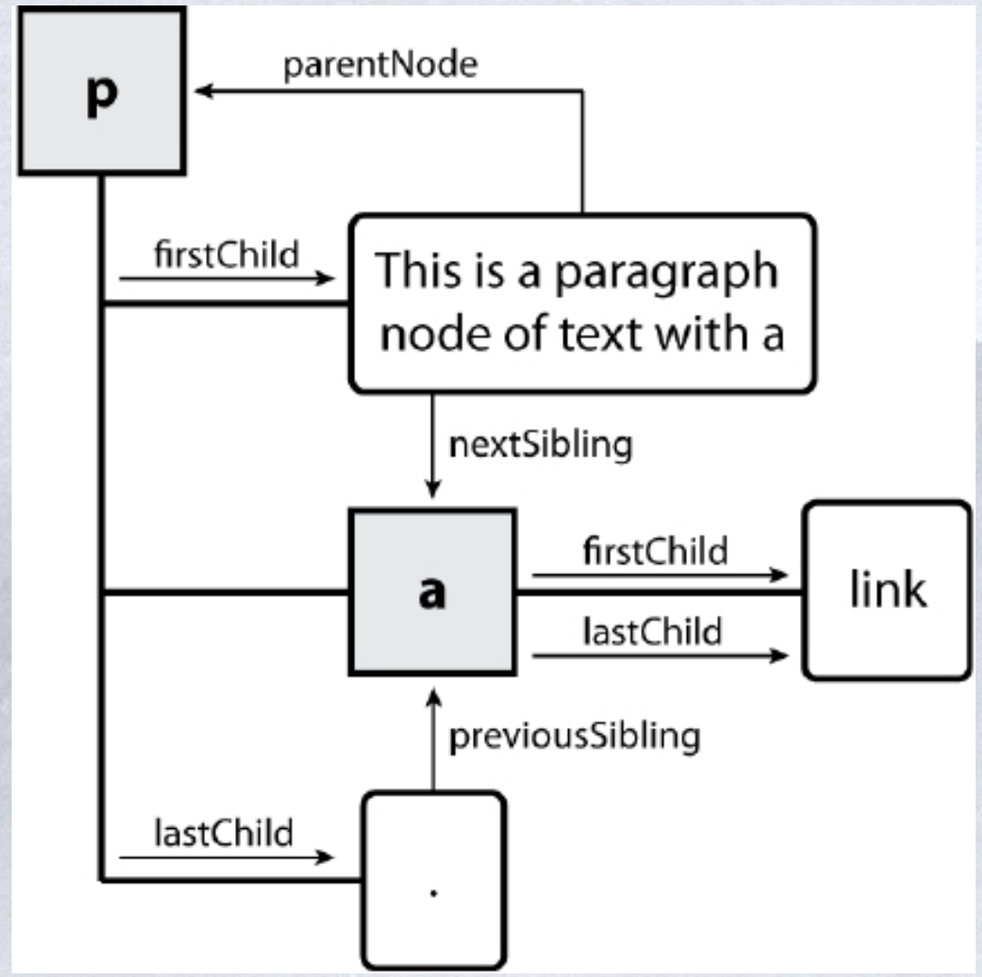
- every node's DOM object has the following properties:

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 (IE sucks)

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>
```



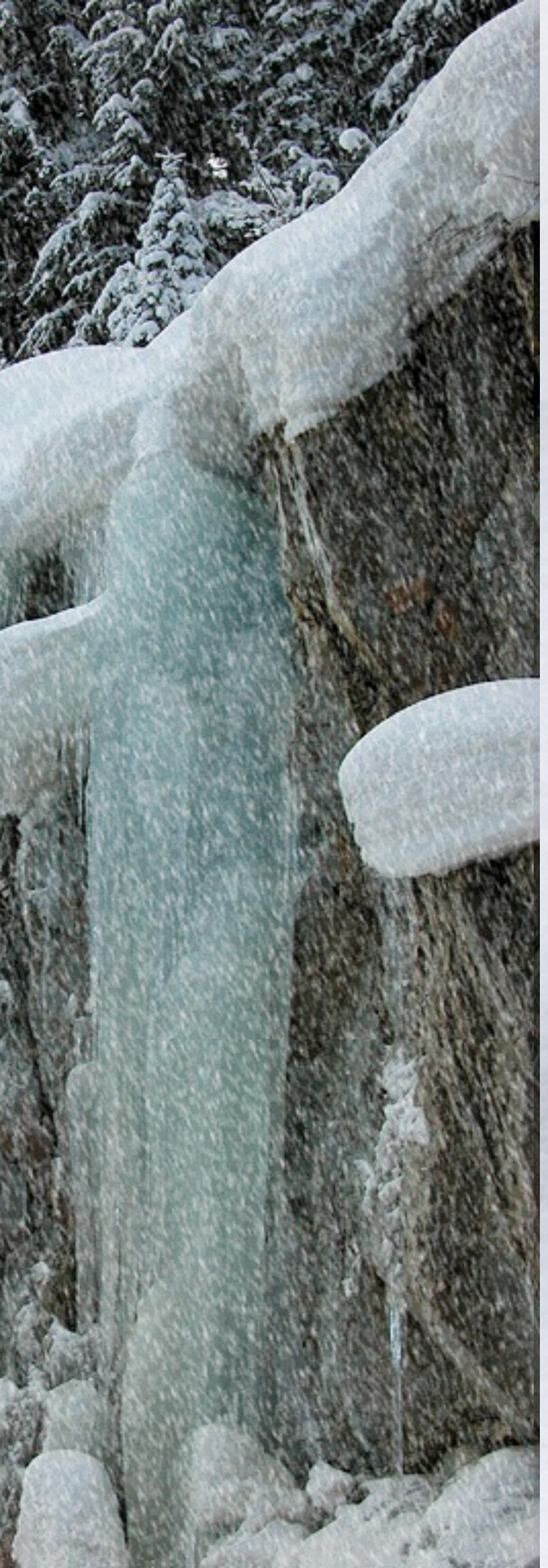
Element vs. text nodes

```
<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? The a tag?



Outline

- Client Side Basics
- Introduction to JavaScript
- JavaScript Basic Syntax
- The DOM tree
- **Manipulating DOM**

Problems with JavaScript

JavaScript is a powerful language, but it has many flaws:

- the DOM can be chunky to use
  - document.getElementById , more than 20 key strikes!
- the same code doesn't always work the same way in every browser
  - code that works great in Firefox, Safari, ... will fail in IE and vice versa
- many developers work around these problems with hacks (checking if browser is IE, etc.)

Prototype framework

```
<script src="http://ssw2p.3322.org/public/scripts/prototype/prototype-1.6.0.3.js"
 type="text/javascript"></script>
```
JS

```
<!-- or link to Prototype home site -->
<script src="http://prototypejs.org/assets/2008/9/29/prototype-1.6.0.3.js"
 type="text/javascript"></script>
```
JS

- the Prototype JavaScript library adds many useful features to JavaScript:
  - many useful extensions to the DOM
  - added methods to String, Array, Date, Number, Object
  - improves event-driven programming
  - many cross-browser compatibility fixes
  - makes Ajax programming easier (seen later)

The $ function

```
$("id")
```
JS

- returns the DOM object representing the element with the given id
- short for document.getElementById("id")
- often used to write more concise DOM code:

```
$("footer").innerHTML = $("username").value.toUpperCase();
```
JS

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>
```

Property	Description	Example
tagName	element's HTML tag	$("main").tagName is "DIV"
className	CSS classes of element	$("main").className is "foo bar"
innerHTML	content inside element	$("main").innerHTML is "\n <p>Hello, ve...
src	URL target of an image	$("icon").src is "images/borat.jpg"

DOM properties for form controls

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

Property	Description	Example
value	the text in an input control	$("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

Abuse of innerHTML

```
// bad style!
var paragraph = document.getElementById("welcome");
paragraph.innerHTML = "<p>text and <a href="page.html">link</a>";
```
JS

- 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
  - (later, we'll see a better way to inject content with HTML tags in it)

Adjusting styles with the DOM

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

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

JS

Color Me

output

property	Description
style	lets you set any CSS style property for an element

- 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

```
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 styling

```
function okayClick() {
  this.style.color = "red";
  this.className = "highlighted";
}
```
JS

```
.highlighted { color: red; }
```
CSS

- 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

Prototype's DOM element methods

absolutize	addClassName	classNames	cleanWhitespace	clonePostion
cumulativeOffset	cumulativeScrollOffset	empty	extend	firstDescendant
getDimensions	getHeight	getOffsetParent	getStyle	getWidth
hasClassName	hide	identify	insert	inspect
makeClipping	makePositioned	match	positionedOffset	readAttribute
recursivelyCollect	relativize	remove	removeClassName	replace
scrollTo	select	setOpacity	setStyle	show
toggle	toggleClassName	undoClipping	undoPositioned	update
viewportOffset	visible	wrap	writeAttribute	

- categories: CSS classes, DOM tree traversal/manipulation, events, styles

Prototype's DOM tree traversal methods

method(s)	description
ancestors, up	elements above this one
childElements, descendants, down	elements below this one (not text nodes)
siblings, next, nextSiblings, previous, previousSiblings, adjacent	elements with same parent as this one (not text nodes)

```
// alter siblings of "main" that do not contain "Sun"
var sibs = $("main").siblings();
for (var i = 0; i < sibs.length; i++) {
  if (sibs[i].innerHTML.indexOf("Sun") < 0) {
    sibs[i].innerHTML += " Sunshine";
  }
}
```

DOM element

JS

- Prototype strips out the unwanted text nodes
- notice that these are methods, so you need ()

Selecting groups of DOM objects

- methods in document and other DOM objects for accessing descendents:

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

Getting all elements of a certain type

- highlight all paragraphs in the document:

```
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

- highlight all paragraphs inside of the section with ID "address":

```
var addrParas = $("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

Prototype's methods for selecting elements

- Prototype adds methods to the document object (and all DOM element objects) for selecting groups of elements:

getElementsByClassName	array of elements that use given class attribute
select	array of descendants that match given CSS selector, such as "div#sidebar ul.news > li"

```
var gameButtons = $("game").select("button.control");
for (var i = 0; i < gameButtons.length; i++) {
  gameButtons[i].style.color = "yellow";
}
```
JS

The $$ function

```
var arrayName = $$("CSS selector");
```
JS

```
// hide all "announcement" paragraphs in the "news" section
var paragraphs = $$("div#news p.announcement");
for (var i = 0; i < paragraphs.length; i++) {
  paragraphs[i].hide();
}
```
JS

- $$ returns an array of DOM elements that match the given CSS selector (in CSS 3, much more powerful than CSS 2)
  - like $ but returns an array instead of a single DOM object
  - a shorthand for document.select
- useful for applying an operation each one of a set of elements

Common $$ issues

- many students forget to write . or # in front of a class or id

```
// get all buttons with a class of "control"
var gameButtons = $$("control");
var gameButtons = $$(".control");
```
JS

- $$ returns an array, not a single element; must loop over the results

```
// set all buttons with a class of "control" to have red
$$(".control").style.color = "red";
var gameButtons = $$(".control");
for (var i = 0; i < gameButtons.length; i++) {
  gameButtons[i].style.color = "red";
}
```
JS

- Q: Can I still select a group of elements using $$ even if my CSS file doesn't have any style rule for that same group? (A: Yes!)

Creating new nodes Web

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

- Every DOM element object has these methods:

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!";
$("main").appendChild(p);
```
JS

A paragraph!

A paragraph!

Removing a node from the page

- each DOM object has a removeChild method to remove its children from the page
- Prototype adds a remove method for a node to remove itself

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

DOM versus innerHTML hacking

- Why not just code the previous example this way?

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

- Imagine that the new node is more complex:
  - ugly: bad style on many levels (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

Problems with reading/changing styles

```html
<button id="clickme">Click Me</button>
```
HTML

```
window.onload = function() {
  $("clickme").onclick = biggerFont;
};
function biggerFont() {
  var size = parseInt($("clickme").style.fontSize);
  size += 4;
  $("clickMe").style.fontSize = size + "pt";
}
```
JS

Click Me

output

- style property lets you set any CSS style for an element
- problem: you CANNOT (usually) read existing styles with it

Accessing styles in Prototype

```
function biggerFont() {
  // turn text yellow and make it bigger
  var size = parseInt($("clickme").getStyle("font-size"));
  $("clickme").style.fontSize = (size + 4) + "pt";
}
```
JS

Click Me

output

- getStyle function added to DOM object allows accessing existing styles
- addClassName, removeClassName, hasClassName manipulate CSS classes

Common bug: incorrect usage of existing styles

```
this.style.top = this.getStyle("top") + 100 + "px";
```
JS

- the above example computes e.g. "200px" + 100 + "px" , which would evaluate to "200px100px"
- a corrected version:

```
this.style.top = parseInt(this.getStyle("top")) + 100 + "px";
```
JS

Useful Prototype $□ functions

- $(), $$()
- $w(): string to array
  - $w('raspberries pears peaches kiwis'); // ['raspberries' , 'pears' , 'peaches' , 'kiwis']
- $A(): an universal converter that turns just about anything roughly collection-like (or, if you prefer, “array-compatible”) into an actual Array object.
  - var ps = $A(document.getElementsByTagName('p')); ps.each(Element.hide);
- $F( ) takes a form field (or its ID) and returns the field’s value
  - alleviating pains of different algorithms for textarea, radio, checkbox, list ....
- $H() for Hash, $R() for Range, both are useful objects in Prototype

Thanks!!!