Objectives

- Review: JSPs, Organization
- JavaScript
  - Some slides in PDF were not covered in class

Review

- Since multiple users can access a servlet at the same time, what problem should we be concerned about?
  - What is the cause of the issue?
  - How do we solve it? What is our process?

Review: Critical Section

- Sections of code that should happen uninterrupted or **atomically**
  - Only one thread can execute at a time
- What is the critical section in this code?
  - The shared file must be read and written atomically
- **Writes** cause trouble

```
// read file
// update local array
// write file
// print results
```

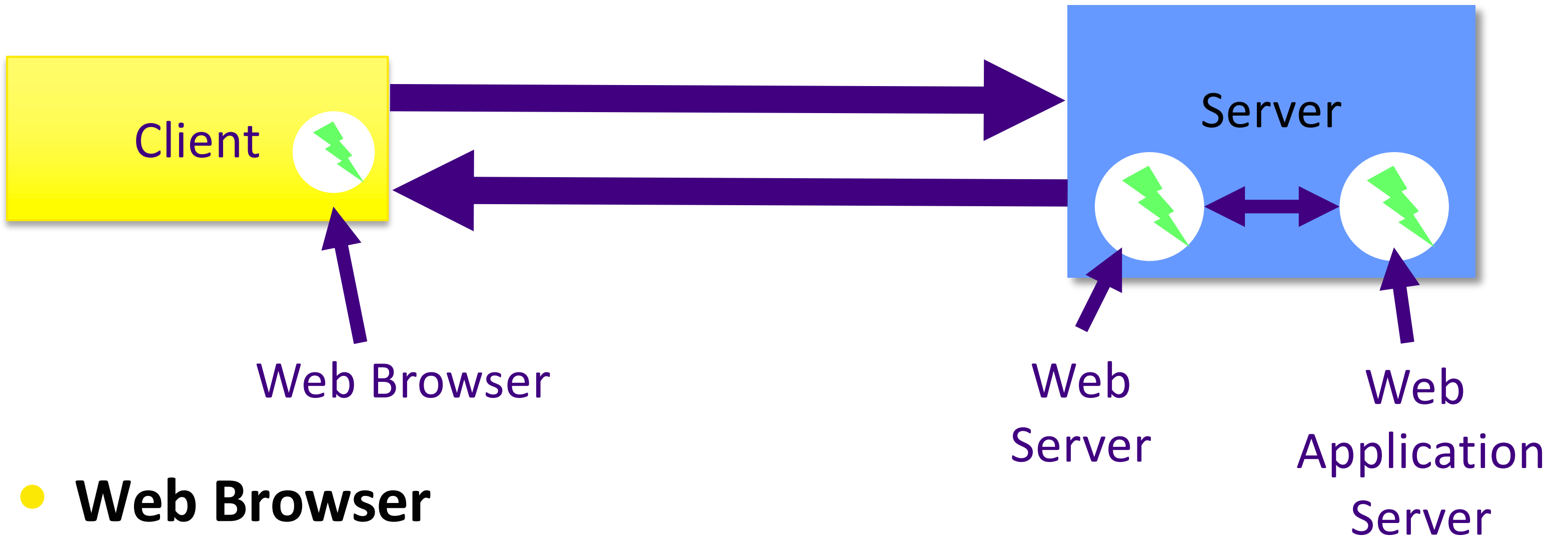
Review: Synchronization Mechanisms

- In code, for in-memory state
  - Synchronized classes
  - Synchronized methods
  - Synchronized statements
- Alternative: database
  - Handles synchronization on *persistent* data
- Expense associated with each of these
  - But without it, get wrong or inconsistent answers!

JSPs and Organization Review

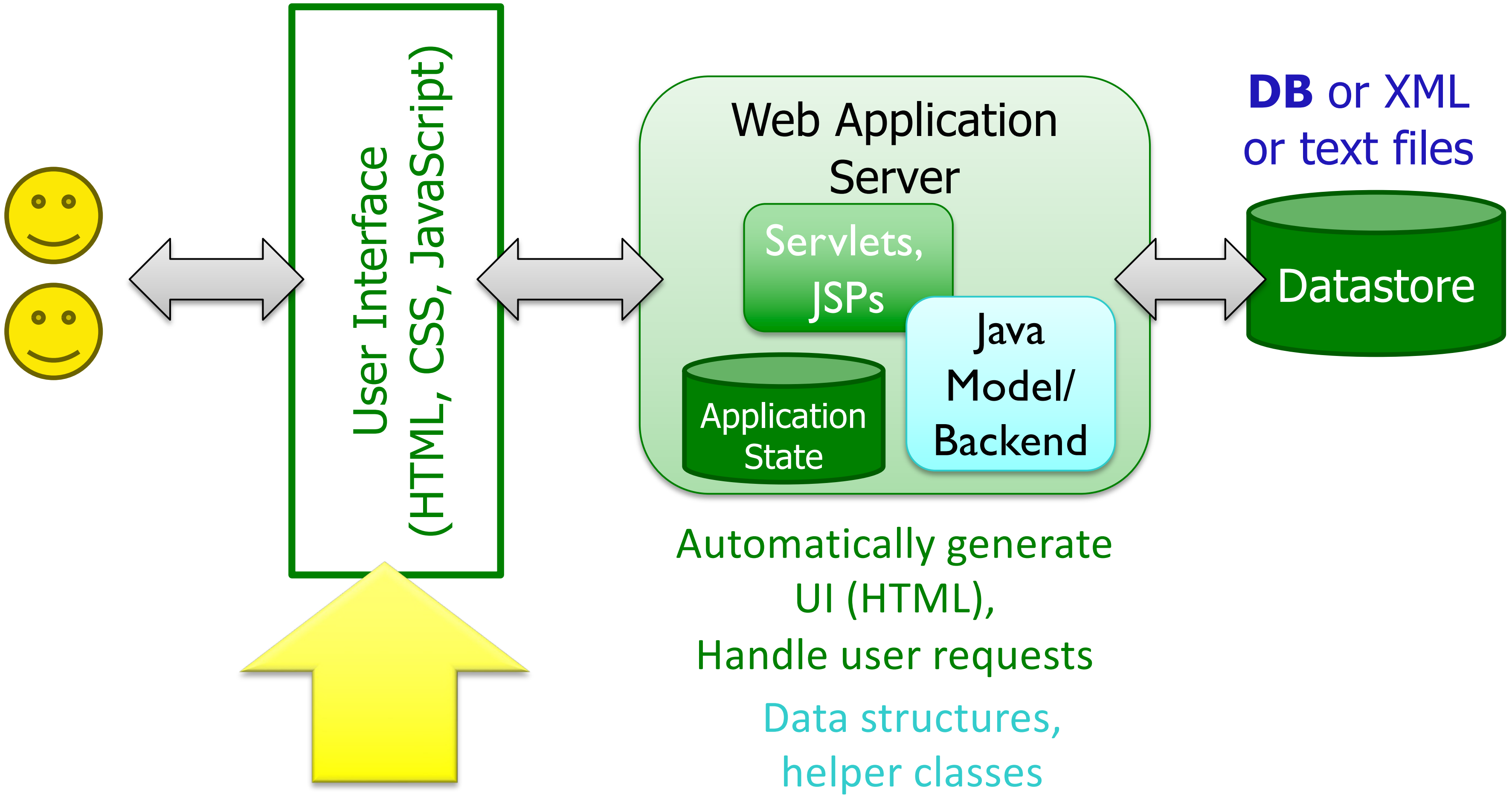
- JSPs
  - What motivated the development of JSPs (in addition to servlets)?
  - What is in a JSP file?
  - How do JSPs execute?
  - How can the JSP use the variable name `request` without declaring/defining it?
- What are your goals when organizing your code for an application: What should be in the servlets? What should be in the JSPs?
- Where can we put JSPs so that users can't directly access them?
  - Why would you want to do that?
- What are the 3 different types of attributes? How are they different?
  - How do you use attributes? (What are they for?)
- True or False: If I mark an input as *required* in the form in HTML, I don't need to check if the parameter is not null on the server side.

Web Programming



- **Web Browser**
  - Makes requests, renders responses
  - Executes **JavaScript**, client-side code
- **Web Server**: handles static requests
- **Web Application Server:** handles ***dynamic*** requests

Web Application Architecture Overview



Automatically generate UI (HTML),
Handle user requests

Data structures,
helper classes

Towards full-stack development

JavaScript

- A lightweight programming language (scripting)
- 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

Differences between JavaScript and 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
  - More procedural, less object-oriented
- Contained within a Web page and integrates with its HTML/CSS content

Pros and Cons of JavaScript

- What can be done with JavaScript on the client side and cannot be done on the server side?
  - Monitor user events and take action
  - Some dynamic effects
- What can be done on both client and server sides but are better on the server?
  - Build HTML dynamically when page is loaded
  - Data validation
- What are the drawbacks of JavaScript?
  - Platform dependent
  - Can be turned off
  - Performance; Security (viruses)

JavaScript Guidelines

- Case sensitive
  - `myVar` is not the same as `myvar`

- Extra white space is ignored

Injecting Dynamic Text

```
document.write("message");
```

- document object represents the current HTML document in the browser
  - Can access elements of document through document
- Prints specified text to page
- Can be used to display HTML
- Argument can be a literal string in quotes or a variable

Variables

```
var name = value;
```

```
var clientName = "Connie Client";
var age = 32;
var salary = 44794.45;
```

- Type is not specified but Javascript does have types
  - Dynamic, weakly typed language
  - Values are converted between types automatically as needed

What programming language is this like?

- Variable names are case sensitive
- Explicitly declared using `var` keyword

Creates a *local* variable

- Implicitly declared through assignment
  - Give it a value and it exists!

Creates a *global* variable

JavaScript Reserved Words

abstract boolean break byte case catch char class const continue

debugger default delete do double else enum export extends false final

finally float for function goto

if implements import in instanceof int interface long native new null

package private protected public return short static super switch synchronized

this throw throws transient true try typeof var void volatile while with

Operators

- Similar operators, precedence hierarchy to Java
- `+ - * / % ++ -- = += -= *=`
- `/= %= == != > < >= <= && || !`
- == checks value
  - "5.0" == 5 is true
- === also checks type
  - "5" === 5 is false
- Many operators auto-convert types
  - 5 < "7" is true

`for` loop

- Syntax:

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

- Example:

What will the output of this be?

```
for (var i = 0; i < 10; i++) {
    document.write("<p>" + i + "^2 = " +
                   (i * i) + "</p>");
}
```

`squared.html`

Using Browser Developer Tools: Console

- To help you debug, write output to the console: `console.log`

`squared.html`

Inserting JavaScript in HTML

- JavaScript code can be added to a web page in 3 ways:
  - In the page's body
    - Put here to run when page loads
  - In the page's head
    - Put here to run when events occur
    - Helps separate script from the HTML
  - In a link to an external `.js` script file

JavaScript in HTML body

- Always runs on page *load*
- Useful for generating dynamic text

```
<body>
          …
    <script>
    JavaScript code
    </script>
          …
</body>
```

Comments

- Identical to Java’s comment syntax

```
// single-line comment

/*
multi-line comment
*/
```

Practice Problem: Hello World

- Write a page that displays "Hello World!" using JavaScript.
- Make "Hello World!" appear 1000 times.
  - Make it so there's only one "Hello World!" per line.

```
helloworld.html
hello.html
```

JavaScript in HTML head

- Put code in head so that it does not run unless *functions* are explicitly called
- Useful for event-triggered actions
  - Examples:
    - Pop up an alert message when a user clicks a given element
    - Display a greeting message on refresh

```
<head>
…
<script>
        JavaScript code
</script>
…
</head>
```

Linking to a JavaScript File

- Can be placed in page's head or body
- Script is stored in a `.js` file
- The source file will not contain the `<script>` tag
- *Preferred way* to write scripts for large projects
- Syntax:

```
<script src="filename">
```

- Example:

```
<script src="example.js">
```

String type

```
var s = "this string";
```

- Can be specified with " " or ' '
- Some Methods
  - `charAt`, `indexOf`, `lastIndexOf`, `replace`, `split`, `substring`, `toLowerCase`, `toUpperCase`
  - `charAt` method returns a value of type *String*
    - No char type in JavaScript
- Example:

```
var firstW = s.substring(0, s.indexOf(" "));
```

More on Strings

- `length` property
  - `s.length` is 13
- Escape sequences behave as in Java and Python
  - `\' \" \& \n \t \\`
- Converting a number to a String

```
var s = new String(myNum);
var sentence = count + " bananas, ah ah ah!"
```

  - Many operators, such as <, automatically convert

More String Methods

- anchor method

```
var txt="Hello world!";
document.write(txt.anchor("myanchor"));
```

  - Result: `<a name="myanchor">Hello world!</a>`

- String style methods
  - bold, italics, fontsize, fontcolor
  - Typically, should be able to use CSS

`hello.html`

Number type

- Integers and real numbers are `Number`s
  - Stored as 64-bit floating point
- Converting a String into a Number
  - Syntax:

```
var integerValue = parseInt(string);
var floatValue = parseFloat(string);
```

  - Examples:

```
parseInt("123hello")
parseInt("booyah")
```

returns 123

returns NaN (not a number)

`if/else` Statement

- Identical structure to Java's if/else statement
- JavaScript is more forgiving about what is in a condition
  - Not just booleans

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

Boolean type

- *Any* value can be used as a Boolean
  - 0, NaN, "", null, and undefined are all `false`
  - All others are `true`

```
if ("CS is great") {  // true, of course!

}
```

- Converting a value into a Boolean explicitly

```
var boolValue = new Boolean(otherValue);
```

`while` Loops

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

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

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

Math object

- Methods
  - abs, ceil, floor, round, log
  - max, min, pow, random, sqrt
  - cos, sin, tan
- Properties
  - E, PI

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

Practice Problem: Random Image

- Randomly display one of two images whenever the page is loaded
- Consider:
  - ➢ Where should script go?
  - ➢ What should script do? (algorithm)
    - How to implement?

F10 F10 F10 F10 F10 F10 F10 F10
F10 F10 F10 F10 F10 F10 F10 F10
F10 F10 F10 F10 F10 F10 F10 F10
ICANHASCHEEZBURGER.COM

https://cs.wlu.edu/~sprenkles/cs335/examples/javascript_examples/randomimg_starter.html

https://jsfiddle.net/b2dvxpm3/1/

Functions

```
function name(parameterName, ..., parameterName) {
    statements;
}
```

- Parameter types and return types are not specified
  - var is not written in parameter declarations
- Functions with no return statement return an undefined value
  - Kind of like void
- Any variables declared in the function are **local** (only exist in that function)

Function Example

- Quadratic Function

```
function quadratic(a, b, c) {
    return -b + Math.sqrt(b*b - 4*a*c) / (2*a);
}
```

  - Again, note no type declarations for parameters, return types

Calling Functions

```
name(parameterValue, ..., parameterValue);
```

```
var root = quadratic(1, -3, 2);
```

- If the wrong number of parameters are passed
  - Too many: extra ones are ignored
  - Too few: remaining ones get an undefined value

Global and Local Variables

```
var count = 1;

function f1() {
    var x = 999;
    count *= 10;
}
function f2() {
     count++;
}

f2();
f1();
```

- Variable count is **global**
  - Seen by all functions
- Variable x is **local**
  - Can be seen by only `f1`
- Both f1 and f2 can use and modify count
- What is count's value at the end?

scope.html

`popup_boxes.html`

3 Types of Popup Boxes

- Alert: Displays message

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

- Confirm: user can confirm or cancel
  - Returns true or false

```
confirm("message");
```

- Prompt: gives text box to user
  - Returns user input string

```
prompt("message" [, "default"] );
```

Date Creation Examples

```
//today
var today = new Date();

//example syntax
var date = new Date( year, monthindex, day );

// Oct 18, 1977: the day
// Reggie Jackson hit 3 homeruns in the World Series
var reggieDay = new Date(1977, 9, 18);

//
var history = new Date('December 17, 1995 03:24:00');
```

Can compare Dates using < > etc.

Date Object Methods

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

Date Object Method Quirks

- `getYear` returns the year since 1900
  - Y2K bug!
  - **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)

Event Handlers

- HTML elements have special ***attributes*** called ***events***
- JavaScript functions can be set as event handlers
- When you interact with the element, the function will execute
- An example of event-driven programming

```
<h2 onmouseover="myFunction();">Click me!</h2>
```

- `onmouseover` is one of many HTML **event** attributes

Practice Problem: Countdown to Graduation

- Write a JavaScript function that will display the number of days until graduation
  - Handle when graduation has past (i.e., when today is after graduation)
- Have the function execute when the mouse hovers over the **h1** element
- Consider: where does the script go? What should the script do?

`https://jsfiddle.net/bLvzus7a/` `countdown.html`

Arrays: 3 Ways to Initialize

```
var stooges = new Array();
stooges[0] = "Larry";
stooges[1] = "Moe";
stooges[2] = "Curly";
```

```
var stooges = new Array("Larry", "Moe", "Curly");
```

```
var stooges = ["Larry", "Moe", "Curly"];
```

Arrays

- **Methods**
  - `pop, push`
    - Remove (return) and add from *end*
  - `shift, unshift`
    - Remove (return) and add from *front*
  - `concat, join, reverse, slice, sort, splice, toString`
- **Properties**
  - `length`

What does this code do?

```
var a = new Array();
a.push("Joey");
a.push("Chandler");
a.unshift("Ross");
a.push("Phoebe", "Monica");
x=a.shift();
a.pop();
a.sort();
document.write(x);
```

What is the value of x?
What does a look like?

Answer

friends.html

```
var a = new Array();
a.push("Joey");           // Joey
a.push("Chandler");     // Joey, Chandler
a.unshift("Ross");      // Ross, Joey, Chandler
a.push("Phoebe", "Monica");
// Ross, Joey, Chandler, Phoebe, Monica
x=a.shift(); // Joey, Chandler, Phoebe, Monica
a.pop();         // Joey, Chandler, Phoebe
a.sort();    // Chandler, Joey, Phoebe
document.write(x);
```

What is the value of x? “Ross”

Strings and Arrays: `split` and `join`

- `split` breaks apart a string into an array using a delimiter
- `join` groups an array of strings into a single string, placing the 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"
```

Special Values: `undefined` and `null`

- `undefined` : has not been declared
- `null` : has been declared but not assigned a value

```
var harry;
var sally = 9;

// at this point in the code,
//   harry is null
//   sally is 9
//   caroline is undefined
```

typeof Function

```
typeof(value)
```

- 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"
```

Arrays as Maps

- Indices of a JavaScript array need not be integers
  - Store mappings between an index of any type (***keys***) and value
- Similar to Java's *Map* collection or a hash table data structure

```
var map = new Array();
map[42] = "the answer";
map[3.14] = "pi";
map["champ"] = usc;
```

For Each Loop

- Loops over
  - every index of the array **OR**
  - every property name of the object

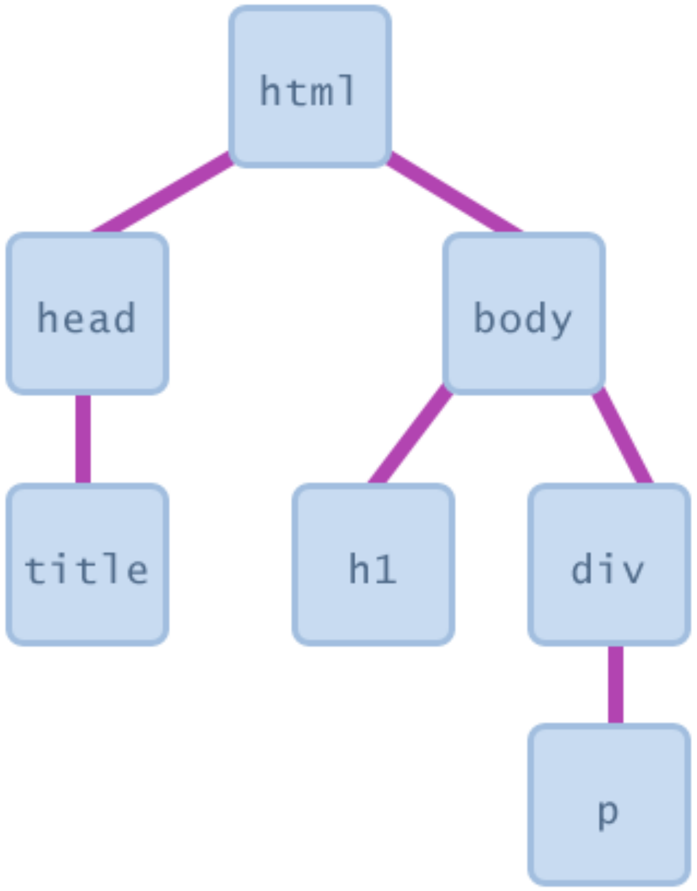
```
for (var name in arrayOrObject) {
      // do something with arrayOrObject[name]
}
```

Browser Object Model (BOM)

- `window` : the browser window
- `navigator` : info about the web browser you're using
- `screen` : info about the screen area occupied by the browser
- `history` : list of pages the user has visited
- `document` : current HTML page
  - Document Object Model (DOM): Our focus

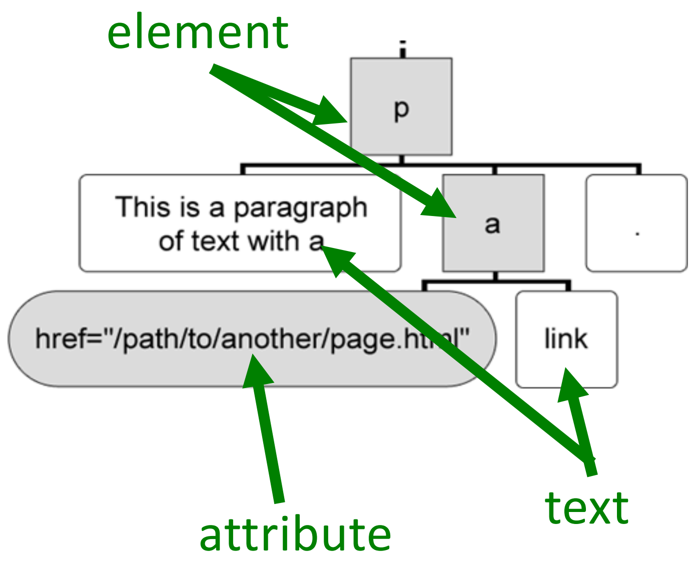
Document Object Model (DOM)

- A representation of the current web page as a set of JavaScript objects
- Allows you to view/modify page elements in script code



Types of Nodes

- **Element**
  - Can have children and/or attributes
- **Text** (text in a block element)
  - A child within an element node
  - Cannot have children or attributes
- **Attribute**
  - Attribute/value pair inside the start of a tag


element
p
This is a paragraph of text with a
a
.
href="/path/to/another/page.html"
link
attribute
text

DOM Node/Object Properties

- `firstChild, lastChild` : start/end of this node's list of children
- `childNodes` : array of all this node's children
- `nextSibling, previousSibling` : neighboring nodes that have the same parent
- `parentNode` : the element that contains this node

See MDN or W3Schools for all DOM node properties, browser incompatiblity information

DOM Element Properties

- DOM objects for all HTML **elements** contain the following properties:
  - `className, id, style, title`
- `style` property
  - Represents the combined styles that apply to element
  - Contains same properties as CSS style properties, **except** names are *capitalized* instead of hyphenated
  - Examples: `backgroundColor, borderLeftWidth, fontFamily`

Selected DOM Node Methods

Method	Description
`appendChild(node)`	places the given node at the end of this node's child list
`insertBefore(newChild, oldChild)`	places the given new node in this node's child list just before `oldChild`
`removeChild(node)`	removes the given node from this node's child list
`replaceChild(newChild, oldChild)`	replaces the given child node with the given new node

Creating New Elements

- `document.createElement("tag")`
  - Constructs a new empty DOM node representing an element of that tag type
- The created node's properties can be set just like any other DOM node's
- After appropriate properties are set, the node can be added to the page

Event HTML Attributes

- Window Events (`body`, `frameset`):
  - `onload`, `onunload`
- Form Element Events (`form`):
  - `onchange`, `onsubmit`, `onreset`, `onselect`, `onblur`, `onfocus`
- Keyboard Events:
  - Available on non-window, non-style elements
  - `onkeydown`, `onkeypress`, `onkeyup`
- Mouse Events
  - Available on non-window, non-style elements
  - `onclick`, `ondblclick`, `onmousedown`, `onmousemove`, `onmouseout`, `onmouseover`, `onmouseup`

Accessing Nodes by id, tag, or name

- `document.getElementById("id")`
  - Returns an object representing the HTML element with the given id attribute
    - null if not found
- `document.getElementsByName("name")`
  - Returns an array of all elements with the given name
- `element.getElementsByTagName("tag")`
  - Returns an HTMLCollection object of all children of the given tag name ("p", "div", etc.)
  - Can be called on the document or on a specific node/element

Using document object’s `getElementById` method

```
function makeRed() {
    var para = document.getElementById("announce");
    para.style.color = "red";
}
```

```
<h2 onmouseover="makeRed();">Sell</h2>
<p id="announce">Get it while it's hot!</p>
```

`hot.html`

Buttons: `<button>`

`button.html`

- Button's text appears inside `button` tag
- `onclick` event handler specifies button's behavior
- Difference between `input` created buttons and these buttons:
  - **`buttons`** can contain content like text or images
  - No content within **`input`** tags

```
<button onclick="function();">
<img src="image.jpg" alt="..." />
</button>
```

The DOM innerHTML Property

ididit.html

- innerHTML refers to the HTML text inside of an element:

```
<p>this is the innerHTML of the p tag </p>
```

- Event handler can modify the innerHTML of another element

```
<p><button id="b1" onclick="update('I did it!');">
Click me! </button></p>
<p id="target">This text will be replaced. </p>

function update(text) {
    var p = document.getElementById("target");
    p.innerHTML = text;
}
```

The DOM **innerHTML** Property

ididit.html

- innerHTML refers to the HTML text inside of an element:

```
<p>this is the innerHTML of the p tag </p>
```

- Event handler can modify the innerHTML of another element

```
<p><button id="b1" onclick="update('I did it!');">
Click me! </button></p>
<p id="target">This text will be replaced. </p>

function update(text) {
    var p = document.getElementById("target");
    p.innerHTML = text;
}
```

Check out the source after click

Two Examples of Changing Images

```
function makeCooler() {
  document.images[ "cool" ].src ="cooltext.png";
}
```

```
<p><img src="cool.png" name="cool" alt="cool" /></p>
```

What happens when makeCooler() is called?

```
<img src="cool.png"
      onmouseover="this.src='cooltextMouseOver.png';"
      onmouseout="this.src='cool.png';" />
```

cooler.html

select Element

- DOM properties: `disabled, length, multiple, name, selectedIndex, size, value` (selected item's text)
- DOM methods: `add(option, index), remove(index)`

```
function addAwards() {
  var selectElem = document.getElementById("awards");
  count++;
  var newOption = document.createElement('option');
  newOption.text = count;
  newOption.value = count;
  newOption.selected = 'selected';
  try {
     selectElem.add( newOption, null);
  } catch( ex ) {  // for IE
     selectElem.add(newOption);
  }
}
```

add_fields.html

select Element

- Attach onchange handler to select to cause behavior on each selection

```
<p>Who is your favorite Voice judge?</p>

<select onchange="alert('You chose ' + this.value);">
  <option>Ariana</option>
  <option>Blake</option>
  <option>John</option>
  <option>Kelly</option>
</select>
```

select_example.html

`<input>`'s DOM properties

- DOM properties for type="text" and type="password":
  - `disabled`, `maxLength`, `readOnly`, `size`, `value` (text in field)

Practice Problem

- Write the HTML and JavaScript code to present a text area and three on/off options for lions, tigers, and bears.
- When the user checks each box, it will add or remove that animal from the text area's text.

Form Validation

- Don’t allow submission through browser until certain criteria is met
- Reduce network traffic, work that server does
- **Not the *only* place to do validation**
  - Still need to check on server-side
  - JavaScript can be turned off
  - Bad guys might not use browser

`form_validation.html`

Using Browser’s JavaScript Tools

- Error Console
- Breakpoints

jQuery

- Commonly used API for writing JavaScript
- Free, open source
- Works across a variety of browsers
- We are *not* using jQuery at this point, just “regular” JavaScript

`https://jquery.com/`

TODO

- Lab 7 - JavaScript practice
  - Due today at 11:59 p.m.