Possible break down of points for the test:

15-20: Matching, True/False
0-5: Short Answer
60: Javascript
15: Web APIs

THE MOST IMPORTANT THING TO STUDY FOR THE LAST TEST is the sample code (zip files) for homeworks!
What runs where after publishing

When a user visits one of your published HTML pages, here's what happens.

- The user’s **browser** makes an HTTP request for an HTML page from cis-linux2 (and HTTP get request). The page is delivered by **apache** back to the user's browser (along with any other items referenced by the page, like images, JavaScript files and css files), then the browser renders the page.

When a browser (javascript code) makes an ajax call to request a Web API (JSP page), here's what happens.

- The browser makes the HTTP request which (because it's a JSP page) gets directed to the JSP App server (tomcat) which runs the JSP page (outputting JSON in our case since all of our JSP pages are Web APIs). Then Apache delivers the JSON page back to the back to the user's browser (and the JS callback function is run).
Let's review by software:

<table>
<thead>
<tr>
<th>Software</th>
<th>Local development PC/MAC</th>
<th>Web or DB Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache</td>
<td>invoked (if not already running) whenever you &quot;run&quot; a page from netbeans, serves up web pages requested by a browser. Listens on port 8080 of your machine.</td>
<td>invoked whenever a user requests a page from cis-linux2, serves up web pages requested by a browser</td>
</tr>
<tr>
<td>Glassfish</td>
<td>JSP application server software that converts/compiles JSP pages into servlets, executes those servlets when JSP pages are requested from localhost. servlets generate HTML pages that are given to Apache to give to the browser that requested them.</td>
<td>glassfish is not installed on cis-linux2</td>
</tr>
<tr>
<td>tomcat</td>
<td>tomcat is not installed on your local PC/MAC</td>
<td>JSP application server that converts/compiles JSP pages to servlets and executes those servlets when JSP pages are requested from cis-linux2</td>
</tr>
<tr>
<td>MySQL</td>
<td>we did not install MySQL on your local PC/MAC, but we could have (if you wanted to keep a local db with exact same design as your one on linux2)</td>
<td>Would be installed/running on a DB server (if we had a distinct DB server). MySql is a DBMS (database mgt system) that responds to requests to access a MySql database, listens on port 3306.</td>
</tr>
<tr>
<td>MySQLWorkBench</td>
<td>only installed on your local PC/MAC, for the purpose of converting GUI actions into SQL code that could be sent to MySQL and executed. It has built in tunneling software.</td>
<td>not installed on linux2. it is not server sw.</td>
</tr>
</tbody>
</table>


Images and Colors (Not covered on last test)

- **Image file formats:**
  - gif (older format, supports animation)
  - png (newer format, made to replace gif that was proprietary at the time, supports transparency, lossless compression, large images render blurry to sharp using a technique called interlacing),
  - jpg (millions of colors, lossy compression algorithm, when traditional jpg images are too large, they render top to bottom and left to right, like we read, but there is a newer "progressive jpg" format that renders like interlaced png, blurry to sharp),

- Try to keep the sum of the size of all your image files on a page to be small, under about 750K or so. Do simple editing of image files to reduce file size of images, so that pages do not render slowly:
  - reduce size of image (e.g., make a 800x800 pixel image be 200x200),
  - crop (e.g., cut out and use a 200x200 pixel section of the larger image),
  - reduce quality (e.g., make the large image smaller, looses definition, then make it bigger again, detail that was lost will not be regained).

- **Web Color Codes:**
  - #RRGGBB in hex, e.g., #FF0000 is red, #00FF00 is green, #FF00FF is purple (red+blue), #000000 is black (absence of color), #FFFFFF is white (max color), any other code where RR=GG=BB is a shade of gray. Dark colors will have low numbers (close to black). Light colors will have high numbers (close to white).
  - Saturation: a fully saturated is “far away” from grey (so you’ll see a large difference between the R the G and the B components of the color). If an image has nothing but unsaturated colors, it is like a black and white photo which means all “colors” are shades of grey.
  - You can also specify purple in decimal like this: rgb(255, 0, 255) or with color name like "purple". 255 is the largest value (like FF) that you can have for R or G or B.
  - You can specify a partially transparent color like this: rgba(256,0,256, 0.25). The last parameter is opacity (how solid). The color just specified is purple that is 25% solid (so 75% transparent).

**HTML**

- **Inside of `<html>`...`</html>` you have `<head>`...`</head>` then `<body>`...`</body>`. The body represents the part that you see inside the browser window. In the head area, you put things like an internal style sheet or references to external style sheets, title (shows up in browser tab). You can also put JS code but we usually put it just before the end of the body tag `</body>`.

- **Block vs. Inline elements:**
  - Block elements (by default, have a new line before and after): `<p>` paragraph, `<div>` box, `<ol>`/`<ul>` ordered/unordered lists, `<table>`.
  - Inline elements (do not have a new line before and after): `<img>` image, `<a>` anchor/link, `<span>`. span doesn’t do anything but lets you mark the beginning and ending of an area where (for example) you might want to apply a certain style.
  - In CSS (below), there is a display property in which you can redefine an element to be block, inline, or none (invisible). So, you can redefine the default behavior of an element by styling it differently.

- **Nesting of HTML tags**: a tag that begins inside of a parent tag must complete within that same tag. This is just like other normal nesting rules, e.g., `{..} in java, (...) in arithmetic expressions.
• NOT COVERED ON LAST TEST: Form tag and input tags (We used input tags but didn’t do much with Forms)
  
  o In older style web pages (where all processing happens server side), if a page wants to get input, it needs a form tag containing input tags as well as a submit button. When the user clicks the submit button, all of the inputs inside the form tag (of the submit button) are sent to the page specified in the action attribute of the form tag. For example, if the user typed “sally” and “22” in the two text boxes and then clicked submit, the URL in the address bar would be this:

    ... mypage.jsp?uName=sally&uAge=22

    <form action = "mypage.jsp" method="GET">
        your name? <input type = "text" name="uName">
        your age? <input type = "text" name="uAge">
        <input type="submit">
    </form>

  o In our client side scripting class, we can do without the form tag and we don’t need the submit button. Our (equivalent) code might look like this:

    your name? <input type = "text" name="uName">
    your age? <input type = "text" name="uAge">
    <input type="button" onclick="JavaScript:makeAjaxCall()">

    And

    1. The JS function (e.g., makeAjaxCall) would make an ajax call to invoke a web API.
    2. The web API would take some action (e.g., insert, edit, delete data, or respond with a list of data) and then respond with (out.print) a message (confirmation, error, or data list).
    3. Then Apache (web server software) would invoke the JS callback function (providing the server’s output) and the callback function would update the page to show the user what had been done.

    See ajax example further down in this document.

  o We almost could have ignored the form tag for this class, except we might want to use the new HTML5 validation (rather than having to write JS code to validate). When I say “validate” I mean things like making sure a user entered number is really a number or a date is really a date or ensuring that there is something typed into a required field. Client side validation makes for quicker responding web page and a better user experience, and so should be done (either HTML5 or JS validation). However, if you are writing the Web APIs, you cannot assume that the HTML coder validated properly, so your Web API needs to double check everything and provide good error messages to the HTML page.

  o To use HTML5 validation, you have to set things up like they would have been set up for server side processing (e.g., JSP Page has UI elements like <html>... </html> not just out printing JSON). So, you need a form tag (no action attribute needed you would not actually post the data to a server side program) and you need a submit button (that can be hidden) like this:

    <form id="personForm">
        your name? <input type = "text" name="uName" required>
        your age? <input type = "number" name="uAge">
        <input type="button" onclick="createPerson()">
        <input style="display:none;" type="submit" value="submit" id="submitButton"/>
    </form>

    And this JavaScript code;

    function createPerson() {
        if (!document.getElementById('personForm').checkValidity()) { // means something is not valid
            document.getElementById('submitButton').click(); // shows the validation bubbles
        } else { // process the input, knowing that all input tags passed validation.
            // ...
        }
    }
Example HTML that uses HTML5 validation:
- "Number" is one of the newer HTML5 input types that are validated by the browser.
- "Required" is also another new validation attribute that is new in HTML5

```html
<input type="number" min="5" max="18" name="shoe-size" required />
```

The above HTML5 input tag will not allow the user to enter a non-number and it will not allow the user to enter a number less than 5 or greater than 18.

CSS (terminology) – Covered only indirectly (when I ask you how something renders etc).

A CSS rule-set consists of a selector and a declaration block:

- Each element inherits its style from its parent (surrounding tag), but can also add its own rules. If two rules contradict, the later rule will be used.
- There are three places you can put CSS:
  - internal style sheet (in `<style>...</style> in `<head>`),
  - external style sheet `<link>` in `<head>`, or
  - inline style, right in the html, like `<div style="font-weight:bold">`.
- Types of CSS selectors (examples):
  - `body { ... } /* no character at the beginning, applies to the given html tag */`  
  - `h1, h2, h3 { ... } /* comma in between, applies to all selectors */`  
  - `#footer { ... } /* applies to the one element with <... id="footer" *> */`  
  - `.navLink { ... } /* applies to all elements with <... class="navLink"> */`  
  - `.navLink a { ... } /* applies to "a" tags (links) that are inside of elements with class="navLink"`  
  - `a:hover { ... } /* pseudo-class, only applies when user hovers over an "a" tag (link) */`  
  - `.navLink:hover a { ... } /* applies to "a" tags (links) inside of hovered element with class="navLink"`  

- Some CSS properties style fonts (font-weight, color, size etc).
- Some CSS properties style "boxes" (margin, padding, border, border-radius, background image, etc.).
- CSS position and z-index properties.
  - The default CSS positioning for elements is "static" which means that elements are placed according to the "normal flow" (top to bottom and left to right, like we read, like a word document is laid out).
  - If the element is an inline type (e.g., "a" link tag or "img"), it is laid out like a word in MSWord, realigns when browser narrows. If the element is a block element (e.g., "div" or "p"), then it’s like a paragraph with new line before and after.
  - If all elements had (the default) static positioning, no elements would ever overlap.
  - But once elements we start using other CSS position values (such as fixed or absolute), elements can overlap, so we typically do need to specify a z-index (higher z-index integer is "on top of" a lower z-indexed element).
  - Elements with "relative" (like static) reserve space in the normal flow, but they can be adjusted slightly left/right and/or top/bottom from where they would have been (using top, bottom, left, right CSS style attributes). The space consumed by a relative element is the space it would have consumed without the slight jogging left/right or up/down.
- Elements with position "fixed" are removed from the "normal flow" and positioned w.r.t. the edges of the browser window (top/bottom, left/right).
- Elements with position "absolute" are removed from the normal flow and are positioned w.r.t. the edges of the first non-static parent ("relative" is normally used for this).

**Debugging CSS/HTML**
- A very basic way to debug CSS/HTML is to use noticeable borders (temporarily) around the area of concern. This can show you visually, what's inside of what and where space is coming from (e.g., is it padding or margins, etc).
- Google Chrome F12 has an "elements" tab that can also be helpful. You highlight an area on your page and it shows you what CSS rules are being applied. There is similar functionality in the Firebug plugin for Firefox.
- Remember that fixed and absolute positioned elements are "removed from the flow" which means that the next element after them acts as if the fixed/absolute element did not exist. Remember to use a higher z-index value to make an element (that may be hidden) come to the forefront.

**Responsive Design**
**definition:** designing a web site so that it looks good regardless of the width of the viewport (e.g., mobile device versus computer). You can accomplish this using a CSS framework like Bootstrap (perhaps 2,000 lines of code) or you can “roll your own” using Media Queries (see CSS example below)

```css
body {
  /* for mobile and desktop */
  margin: 0; padding: 0; /* avoid white line just inside browser window */
  font-family: Tahoma, 'Arial Black', san-serif;
  font-size: 18px;
}
.container { width: 100%; }
/* more rule sets */

@media screen and (min-width: 600px) {
  /* for desktop only */
  .container { margin: auto; max-width: 1000px; }
  .left { float: left; width: 49%; margin-right: 1%; }
  .right { float: right; width: 49%; margin-left: 1%; }
  .clear { clear: both; }
  /* more rule sets */
}
```

**JavaScript**
- Although JavaScript syntax is similar to Java, Java is compiled, whereas JavaScript is interpreted. Although NetBeans does a pretty good job of providing syntax errors, you don’t get “compiler” errors that prevent you from running the code. If you run the line of code that has the error, JavaScript just stops running and you wouldn’t see any error message unless you had the Chrome F12 console open.
- Java has strong data typing, whereas js does not. For example, in js, a variable can start out integer and change dynamically to string. In JavaScript, functions are first class data types, treated similarly to other data types (like string or integer or date). To declare a variable in js, you use the word var (for any data type, including functions and/or objects).
• If you never declare a variable and then reference it (on left hand side), js declares the variable automatically (and makes it GLOBAL, which is especially bad). To prevent this auto-declaration, use this line of code at the start of your code (quotes and all):
  "use strict";

• When you declare a variable (using var) inside of a function, the scope of that variable is that function. In ES6, there are two new keywords for declaration: let (has block level scope like java) and const (JavaScript will throw an error if any code attempts to change the value of a variable declared with const).

• You can declare and instantiate an object "on the fly" without ever having to create a class. For example, this is an object with two properties:
  ```javascript
  var myObj = { name: "sally", age: 22 };
  ```

• Here is another example of an object that has one data member (e.g., property) which is actually a function (like method).
  ```javascript
  var myObj = {
    name: "sally",
    sayHello: function() {
      console.log("hello");
    },
    age: 22,
  };
  myObj.sayHello(); // would print "hello" in the console log
  ```

• In java, when you instantiate objects of a class, each object has its own data values, but all objects of a particular class share (one copy of) all the class's methods. Assuming that you do not use "prototypes" (and you are not responsible for nor recommended to use prototypes), every JavaScript object has its own methods (not shared) that can be updated independently. For example:
  ```javascript
  var myObj = {
    name: "sally"
  };
  myObj.sayHello = function() {
    console.log("hello");
  };
  var yourObj = {
    name: "sally",
    sayHello: function() {
      console.log("bonjour");
    }
  };
  myObj.sayHello(); // would print "hello" in the console log
  yourObj.sayHello(); // would print "bonjour" in the console log
  ```

• JavaScript code goes inside of <script>…</script> tags. There can be any number of <script> tags throughout your page. The "global" code in the script tags is run (top to bottom) as the page is rendered. Lines of code can
be in between functions (and run) with no problem. When functions are encountered, they are defined but are not run unless they are called/invoked.

- It is common to put JavaScript code inside a script tag just before the end of the body. This does two things
  1. ensures that all DOM elements exist before you might try to reference them (else you'll get a runtime error).
  2. visual elements on the page load more quickly (only relevant in cases where there is a LOT of js code).
- We use document.getElementById() to get a reference to an HTML element to access or modify its properties. Ids are supposed to be unique within a page. Example:
  
  ```html
  <div id="bigBox">... 
  document.getElementById("bigBox").style.color = "red";
  ```

**Debugging JavaScript**

- It would be very hard to debug JavaScript without using a JavaScript debugger (such as pressing F12 in Chrome). You could use an alert() if you want to stop the action and see values of variables, but using console.log() (with Chrome F12 pressed) is a much better way to debug without stopping the action.
- If you want to see the values of all the properties of an object, you can do something like this:
  
  ```javascript
  console.log(obj); // prints “tree view” showing properties and values of object
  console.log("obj is "+myObject); // prints “obj is myObject” (not helpful).
  ```

**jQuery, minification, CDNs – we didn’t do much actual jQuery coding, but know what it is, how it works.**

- jQuery is a JavaScript library. To reference jQuery, use a CDN (Content Delivery Network) like this:
  
  ```html
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/2.2.2/jquery.min.js"></script>
  ```

  - Remember that all JavaScript (so that includes the jQuery library) runs in the browser and therefore must be downloaded by every user – unless the user’s browser already has it cached.
  - You could download jQuery (to your web site's folder), but why? It would be a very bad idea to modify your local copy of jQuery because then you'd be stuck with that version which someday will be outdated and unsupported. Another benefit of referencing (a commonly used) CDN is that your users probably already have a cached copy of jQuery from having visited other sites that also referenced the same CDN.
  - Should you reference the minified version of jQuery? Yes, because it downloads faster and it runs faster.
    - Minified means that the jQuery JavaScript library has been run through a "minifier" which drastically reduces the size of the jQuery code. Minified code has no whitespace, no comments, and all names (variable, object, function etc) have been made very short. Trying to make sense of a minified JavaScript file would be like trying to make sense of java compiled class code.

- Typically you put your jQuery code inside of this – so that the jQuery code will not error out when it tries to reference a DOM element that does not exist yet.
  
  ```javascript
  $(document).ready(function(){
    // jQuery methods go here...
  });
  ```

- Jquery syntax: Basic syntax is:  `$(selector).action()` – action is also called effect.

Examples:

  - $(this).hide() - hides the current element (e.g., the element involved in a click event, see below)
  - $('p').hide() - hides all <p> elements.
  - $('.test').hide() - hides all elements with class="test".
$('#test').hide() - hides the element with id="test".

- There are not a huge number of jQuery actions (effects). Examples are fadeIn, fadeout, hide, show. One super versatile action is "animate" because you can change any CSS property of any DOM element (and do it fast/slow).

- Here is an example that pops up an alert then hides any paragraph that is clicked.

```javascript
$("p").click(function() {
    alert("The paragraph was clicked.");
    $(this).hide(); // hide the element that was clicked
});
```

- jQuery also provides an ajax call function that is widely used – that does the same thing as our own DIY (Do It Yourself) ajax function.
By Now, You should fully understand the **drop down framework**.
Note the JS file has a single function that does whatever necessary to manage drop down headers (adding on click functions to open drop down contents) and making sure that a click elsewhere closes all drop down contents.
This code does not return a component, because the component has no public functions that the HTML coder might want to call.

The HTML page just has to call the dropdownFw (passing in optional params)

```javascript
"use strict";
function dropdownFw(paramsIn) {
  var params = paramsIn || {}; // if nothing at all is passed in, assume an empty parameter object

  // The parameter object can have three (optional) properties (next three assignment statements).
  // This is the classname of the drop down header menus (check style sheet and nav bar).
  var dropHeaderStyle = params.dropHeaderStyle || "dropHeader";

  // This is the classname of the drop down content menus (check style sheet and nav bar).
  var dropContentStyle = params.dropContentStyle || "dropContent";

  // hiddenRight is a large negative number (how far off screen they are initially before they transition in).
  var hiddenRight = params.hiddenRight || "-500px";

  var headerList = document.getElementsByClassName(dropHeaderStyle); // all dropdown header elements
  for (var i = 0; i < headerList.length; i++) {
    headerList[i].onclick = function() { // add onclick function to each dropdown header element
      // get the parent of the clicked element (drop down header), then from there get the first (and only) drop content element.
      var parent = this.parentElement; // "this" means clicked DOM element.
      var dContent = parent.getElementsByClassName(dropContentStyle)[0];

      // when one dropdown is clicked, close any other ones that the user may have left open.
      var dropContentList = document.getElementsByClassName(dropContentStyle);
      for (var i = 0; i < dropContentList.length; i++) {
        if (dropContentList[i] !== dContent) {
          hide(dropContentList[i]);
        }
      }

      // Tip: JS doesnt understand the initial CSS values (the values set by style sheet instead of by JS), unless you use
      // the getComputedStyle function. To avoid having to use getComputedStyle make your conditions test for
      // the way CSS does NOT have it initially set. (In other words, do not reverse the order of the if/else block.)
      if (dContent.style.visibility === "visible") {
        hide(dContent);
      } else {
        show(dContent);
      }
    }
  }

  ...
```
Continuing the Drop Down Framework ...

// private function, make element invisible and off to the right (for animation coming back in).
function hide(ele) {
    ele.style.right = hiddenRight;
    ele.style.visibility = "hidden";
}

// private function, makes element visible.
function show(ele) {
    ele.style.visibility = "visible";
    ele.style.right = "0px";
}

function hideAllDropContents() {
    var dropContentList = document.getElementsByClassName(dropContentStyle);
    for (var i = 0; i < dropContentList.length; i++) {
        hide(dropContentList[i]);
    }
}

// Close all dropdown content menus whenever the user clicks anything but a drop down header
window.onclick = function (event) {
    if (!event.target.matches('.' + dropHeaderStyle)) {
        hideAllDropContents();
        //console.log("hiding all drop contents");
    }
};

Routing Framework (code in HTML page):

var myRoutes = [];
    // Declare an array to store our routes:

    // Add link/content entries into the route array. Remember that with JS associative array syntax, the keys look like
    // index values even though they are character strings, not integers. The value stored is a function not a character string.
    myRoutes["/"] = home;  // home is a function stored in js/components/home.js
    myRoutes["/home"] = home;
    myRoutes["/blog"] = blog; // blog is a function stored in js/components/blog.js

    // routeFw returns an object that has private and public data members and methods. This "consumer code" specifies
    // input parameters (array that specifies the js function to be run for each link of browser, id where content is to be injected,
    // and initial starting link.
    var myRouteFw = routeFw({
        routeArray: myRoutes,
        contentId: "view",
        startingPath: '/home'
    });

    // print the routes (check console.log). This is just an example how to call a public method of the component/object
    // returned by the routing FW.
    myRouteFw.printRoutes();
Routing Framework (code in JS file, the function that makes and returns the routing component):

"use strict";
function routeFw (params) {

    var fw = {}; // creating and adorning this object to be passed back to the HTML page.

    var startingPath = params.startingPath || '/home';
    var contentId = params.contentId || "view";

    if (!params.routeArray || params.routeArray[0]) {
        alert("parameter object must specify array 'routeArray' with at least one element");
        return;
    }

    var routes = params.routeArray; // Declare a (private) array to store our routes.

    function router() { // private function that will be called whenever a link is clicked (or href changed)
        var path = location.hash.slice(1) || '/'; // remove leading # from string that holds the clicked link
        console.log('path (with no #) is ' + path); // prints something like /home

        // Use the path like an index to the routing array (JS associative array notation) to find which function to run.
        // If a link is clicked for which a route was never set, give error message.
        if (!routes[path]) {
            document.getElementById(contentId).innerHTML = "<p>Error: path '' + path + '' was never added to the routing.</p>";
        } else {
            routes[path](contentId);
        }
    }

    fw.printRoutes = function () {
        console.log("routes will be printed on the next line ");
        console.log(routes);
    };

    // Whenever a link is clicked (or window.location.hash changes), invoke function router (defined below).
    window.addEventListener('hashchange', router);

    // Make sure there's always a browser URL change on page refresh. So you never see empty content on page refresh.
    window.location.hash = "/xxx";

    // content for when page is first rendered.
    window.location.hash = startingPath;

    return fw;
}
**JSON Compared to XML** (data formats for the web)

- XML and JSON are both human readable data formats that are used on the web.
- XML looks like HTML except that with XML, the tag names are programmer created, whereas with HTML, the tag names are pre-determined (like `<body>` `<table>` etc).
- JSON is short for JavaScript notation (see example below). The `{ }` starts/stops an object. The `[ ]` is a list.
- In either case (XML or JSON), you can have any level of nesting of objects and lists.

*We used JSON in the Web API homework. JSON is important. You should understand what it is and how to use it (convert from JSON to JS object and back). As for XML, just know what it is (see below) – a different web data format.*

<table>
<thead>
<tr>
<th>JSON</th>
<th>XML</th>
</tr>
</thead>
<tbody>
<tr>
<td>`[</td>
<td><code>&lt;Rockbands&gt;</code></td>
</tr>
<tr>
<td>{</td>
<td>&lt;Rockband&gt;</td>
</tr>
<tr>
<td>&quot;Name&quot;: &quot;Beatles&quot;,</td>
<td><code>&lt;Name&gt;Beatles&lt;/Name&gt;</code></td>
</tr>
<tr>
<td>&quot;YearFormed&quot;: 1959,</td>
<td><code>&lt;YearFormed&gt;1959&lt;/YearFormed&gt;</code></td>
</tr>
<tr>
<td>&quot;Members&quot;: [&quot;Paul&quot;,&quot;John&quot;,&quot;George&quot;,&quot;Ringo&quot;]</td>
<td><code>&lt;Members&gt;</code></td>
</tr>
<tr>
<td>},</td>
<td><code>&lt;Member&gt;Paul&lt;/Member&gt;</code></td>
</tr>
<tr>
<td>{</td>
<td><code>&lt;Member&gt;John&lt;/Member&gt;</code></td>
</tr>
<tr>
<td>&quot;Name&quot;: &quot;Rolling Stones&quot;,</td>
<td><code>&lt;Member&gt;George&lt;/Member&gt;</code></td>
</tr>
<tr>
<td>&quot;Country&quot;: &quot;England&quot;,</td>
<td><code>&lt;Member&gt;Ringo&lt;/Member&gt;</code></td>
</tr>
<tr>
<td>&quot;YearFormed&quot;: 1962,</td>
<td>&lt;/Members&gt;</td>
</tr>
<tr>
<td>&quot;Members&quot;: [&quot;Mick&quot;,&quot;Keith&quot;,&quot;Charlie&quot;,&quot;Bill&quot;]</td>
<td><code>/Rockband&gt;</code></td>
</tr>
<tr>
<td>}</td>
<td><code>&lt;Rockband&gt;</code></td>
</tr>
<tr>
<td></td>
<td>&lt;Name&gt;Rolling Stones&lt;/Name&gt;</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Country&gt;England&lt;/Country&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;YearFormed&gt;1962&lt;/YearFormed&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;Members&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;Member&gt;Mick&lt;/Member&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;Member&gt;Keith&lt;/Member&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;Member&gt;Bill&lt;/Member&gt;</code></td>
</tr>
<tr>
<td></td>
<td>&lt;/Members&gt;</td>
</tr>
<tr>
<td></td>
<td><code>/Rockbands</code></td>
</tr>
</tbody>
</table>

**JavaScript example 1: initializing objects**

// `{ }` bundles properties into an object. `[ ]` is a list. You can have a list of objects or an object with lists in it. // practically unlimited nesting.

```javascript
var person = { fname: "Sally", age: 38, kids: ["Chris", "Frank", "Maria"] };
```

// Pops up this message: “I am Sally, I am 38, and my oldest child is Chris”.

```javascript
alert("I am "+person.fname+", I am "+person.age+", and my oldest child is "+person.kids[0]);
```

**JavaScript example 2: turning JSON data (String) into JS object**

```javascript
var personData = '{ "fname": "Sally", "age": 38, "kids": ["Chris", "Frank", "Maria"] }';
var person = JSON.parse(personData);
```

// Pops up this message: “I am Sally, I am 38, and my oldest child is Chris”.

```javascript
alert("I am "+person.fname+", I am "+person.age+", and my oldest child is "+person.kids[0]);
```
AJAX (stands for Asynchronous JavaScript And XML - even though we usually use JSON instead of XML)

- Ajax is a technique in which JavaScript (running in the browser) makes a HTTP request and receives data from a server asynchronously. When the request is made, a "call back (JavaScript) function" is specified. When the server responds, the browser invokes that JavaScript callback function. Asynchronous means the code is non-blocking, in other words, it does not pause waiting for server code to supply the data.
- With ajax, you do not have to get a whole new web page to simply some information on the page. An ajax call can be made to its own web site or to a different web site.
- When using 3rd party Web APIs, you can easily get a CORS error (Cross Origin Resource Sharing): "No Access-Control-Allow-Origin header is present on the requested resource". This error is a security violation that says “the domain of the HTML page making the request is different than the domain of the Web API”. If you get this error, it would have to be resolved by the administrators of the two domains that are involved. This is why I have stopped asking students to find a 3rd party Web API to call for their homework – instead we write our own Web APIs and avoid the problem altogether.
- Be aware of the order of how this executes. Order is “normal” except that there is a “big” delay before the callback function is called (handleResponse in this example). Test question could ask you to show what would print in console.log to prove you understand how this asynchronous code works.

AJAX Example (JavaScript)

```javascript
function sendRequest(artist) {
    var httpReq = new XMLHttpRequest(); // browser supplies this object
    var url = "http://ws.audioscrobbler.com/2.0/?method=artist.getinfo&artist=" + artist + "&api_key=aff04cb3e4b21ab37d92659bdea550b1&format=json";
    httpReq.open("GET", url);
    httpReq.onreadystatechange = handleResponse; // specify the callback function (see bold below)
    httpReq.send(null); // initiates the call
}

function handleResponse() {
    if (httpReq.readyState === 4) {
        if (httpReq.status === 200) {
            var jsonObj = JSON.parse(httpReq.responseText);
            if (jsonObj.artist) { // artist found - display bio info
                document.getElementById("bio").innerHTML = jsonObj.artist.bio.summary;
            } else { // artist not found - display error message
                document.getElementById("bio").innerHTML = jsonObj.message;
            }
        } else { // error making Web API call, no response from server
            document.getElementById("bio").innerHTML = "Error making ajax call to " + url;
        }
    }
}
```
Understand how a function (like our “Make” functions) can act as constructor, returning an object that includes properties and methods.

- Understand the difference (in JavaScript) between public and private properties and functions. The term “closure” in JavaScript means that a variable is not destroyed after the function that declared it is done executing - we use this to create “private” variables for the objects created in our “make” functions. This is rather like static variables in java (retain their values, not destroyed).

- Understand JavaScript namespaces (declaring objects/functions inside of other objects/functions to prevent name conflicts).

- Why passing a parameter object (instead of a parameter list) can provide 2 benefits: code is more self documenting, (2) easy handling of optional parameter properties.

- Understand ”dependency injection” (e.g., JS Make Function cannot assume anything about the HTML page except whatever the HTML page passed into it).

- If you don’t declare variables (e.g., var, let, const), then JS auto-declares a GLOBAL variable which pollutes the global namespace. Include "use strict"; in your JS code so you are forced to always declare your elements.

```javascript
/* function MakeCounter returns an object that has two private data members (count and name) * as well as a public method (increment) and private method (print). The data members * retain their values (called “closure”) even after function MakeCounter has finished executing (“closed”). */

function MakeCounter(counterName, initValue) { // Ideally would be in separate JS file...
  var count = initValue;
  var name = counterName;
  counterObj = { }; // empty object. Below add a public method and return.
  counterObj.increment = function () { // / public method
    count++;
    print(); // call private method print
  };
  function print() { // private method, accessible only inside MakeCounter
    console.log("Counter " + name + " value is " + count);
  }
  print(); // initial print of name and count
  return counterObj;
}

// MAIN PROGRAM, ideally in an HTML page.
var first = new MakeCounter("first counter", 10); // prints: first counter value is 10
first.increment(); // prints: first counter value is 11
var second = MakeCounter("second counter", 100); // prints: second counter value is 100
first.increment(); // prints first counter value is 12
second.increment(); // prints second counter value is 101
console.log("private property: " + second.count); // error: count is a private property
console.log("private property: " + first.print); // if comment previous line, get error: print is a private function
```

Advanced JavaScript
Study all concepts and sample code on my Advanced JavaScript page:

http://cis-linux2.temple.edu/~sallyk/tutorials_js_Adv/index.php
Client Side Data Persistence:

- Cookies have been around for some time. Using these, you could write a name/value pair (to a text file on the client’s PC) and then later read it back.
- Newer/easier/better method is to use “local storage”. Code examples for data persistence using local storage:
  
  ```javascript
  localStorage.setItem("lastname", "Smith");             // Store “Smith” into local storage
  var theLastName = localStorage.getItem("lastname");  // Retrieves “Smith” from local storage
  localStorage.removeItem("lastname");                 // Removes “Smith” from local storage
  ```

Databases – Important but not on the Last Test

- Definitions
  
  - A database is a set of related tables (data) that can ONLY be accessed (read/modified) by DBMS software.
  - Know these terms: table, field/column, record/row, primary key (PK) constraint, uniqueness constraint.

- Database Management System (DBMS) like MySql
  
  - Definition. Software that
    
    - lets you design a database (e.g., data types, required vs. optional, PKs, FKs).
    - lets you and your programs access and modify a database, but only according to the design rules. The DBMS is the ONLY way to access a database (which is why the DBMS is able to enforce all the rules all the time).
    - lets a Database administrator set up users, user groups, and sets user and group privileges (at a fine grained level, e.g., table level, field level).
    - Installed/runs on cis-linux2 like daemon waiting for requests.
  
  - Examples of DMBSs: MSAccess (for individual or small group usage), MS SQL server (large scale, affordable), MySql (open source, used for web apps), oracle (very large scale, expensive).
  
  - MySql is a DBMS (service) that also provides a command line interface (black screen) where you can type in the SQL commands that you saw MySqlWorkbench creating for you.
  
- MySqlWorkbench is a GUI front end to MySql. It allows you to point and click to say what you want to do. Without MySqlWorkbench, you’d have to type (syntactically correct) SQL commands into a black screen (you’d be talking directly to MySQL, running on cis-linux2).

- SQL:
  
  - SQL (Structured Query Language) is a standard language for accessing databases, regardless of which DBMS is used.
  - Understand that SQL is a large language with commands that are grouped (as shown below, but you are not responsible for memorizing the acronyms):
    
    - DML (data manipulation language), such as insert, update, delete, select.
    - DCL (data control language) grant, revoke (creation of users/passwords, allocation of authorization)
    - DDL (data def’n language) create table, alter table, add constraints.
  
  - Understand how these SQL statements work: INSERT, UPDATE, DELETE and how they might throw an exception, e.g., uniqueness violation, Foreign Key violation (we didn’t really use these), Primary Key violation (since we set up our table PKs as auto-increment, we never ran into this).

SERVER SIDE CODE:

- Understand the Jar files:
  
  - JAR means Java ARchive file, like a zip file of compiled java code
  - MySql database driver (allows java code to communicate with the MySQL DBMS)
  - and the GSON Jar files (converts POJOs to JSON and vice versa, POJO means Plain Old Java Object)
UNDERSTAND what these important Java classes do:

- java.sql.Connection
- java.sql.PreparedStatement
- java.sql.ResultSet

STUDY dbUtils.DbConn – a wrapper class for java.sql.Connection.

- Wrapper meaning that it “enhances” java.sql.Connection. Here are the things that DbConn does that goes beyond what java.sql.Connection provides: (1) manages attaching to the mySQL database driver (2) supplies the database credentials (3) determines where the web app is running (your development PC or cis-linux2) and provides the appropriate database connection string.

Understand what is a database connection leak (a code path that opens but does not close a database connection - results in database becoming unavailable to some users under heavy usage).

Sample JSP Web API (that uses/imports java classes). RUNS ON SERVER. RESPONDS TO AJAX CALL.

```jsp
<%@page contentType="application/json"; charset=UTF-8" pageEncoding="UTF-8"%>
<%@page language="java" import="dbUtils.*" %>
<%@page language="java" import="model.webUser.*" %>
<%@page language="java" import="java.sql.PreparedStatement" %>
<%@page language="java" import="java.sql.ResultSet" %>
<%@page language="java" import="com.google.gson.*" %>
<%
String sql = "SELECT web_user_id, user_email, birthday FROM web_user ORDER BY web_user_id ";
StringDataList strDataList = new StringDataList();
DbConn dbc = new DbConn();
strDataList.dbError = dbc.getErr(); // returns "" if connection is good, else db error msg.
if (strDataList.dbError.length() == 0) { // if got good DB connection,
    try {
        PreparedStatement stmt = dbc.getConn().prepareStatement(sql);
        ResultSet results = stmt.executeQuery();
        while (results.next()) {
            StringData sd = new StringData(); // all fields "" (empty string, nothing null).
            sd.webUserId = results.getString("web_user_id");
            sd.userEmail = results.getString("user_email");
            sd.birthday = FormatUtils.formatDate(results.getObject("birthday"));
            strDataList.add(sd);
        }
        results.close();
        stmt.close();
    } catch (Exception e) {
        System.out.println("*** Exception thrown, messages is: " + e.getMessage());
        StringData sd = new StringData();
        sd.errorMsg = "Exception thrown, messages is: " + e.getMessage();
        strDataList.add(sd);
    }
    dbc.close(); // EVERY code path that opens a db connection, must also close it - no DB Conn leaks.
}%```
Gson gson = new Gson(); // Gson object (from the GSON library) can to convert between JSON <-> POJO (plain old java object)
out.print(gson.toJson(strDataList).trim());

%>

The StringData class:
We had model.webUser.StringData and you created another model.xxx.StringData for your “other” database table.

package model.webUser;
import dbUtils.FormatUtils;
import java.sql.ResultSet;

/* The purpose of this class is just to "bundle together" all the character data that the user might type in when they want to
* add a new Customer or edit an existing customer. Because the data is "pre-validated" data, all fields are type String even if
* in the database a particular field might be another data type. There are no getter or setter methods since we are not trying to
* protect this data in any way. We want to let the JSP page have easy access to put data in or take it out. */

public class StringData {

    public String webUserId = "";
    public String userEmail = "";
    public String userPassword = "";
    public String image = "";
    public String birthday = "";
    public String membershipFee = "";
    public String errorMsg = "";

    public StringData() { // default constructor leaves all data members with empty string (Nothing null).
    }

    public StringData(ResultSet results) { // overloaded constructor sets all data members by extracting from resultSet.
        try {
            this.webUserId = FormatUtils.formatInteger(results.getObject("web_user_id"));
            this.userEmail = FormatUtils.formatString(results.getObject("user_email"));
            this.userPassword = FormatUtils.formatString(results.getObject("user_password"));
            this.image = FormatUtils.formatString(results.getObject("image"));
            this.birthday = FormatUtils.formatDate(results.getObject("birthday"));
            this.membershipFee = FormatUtils.formatDollar(results.getObject("membership_fee"));
        } catch (Exception e) {
            this.errorMsg = "Exception thrown in model.webUser.StringData (the constructor that takes a ResultSet): " + e.getMessage();
        }
    }
}

19
This is the model.webUser StringDataList class (yours for your “other” table would have been similar). The properties of this object get converted (via the GSON/JSON code/class) to JSON format (in the Web API JSP).

```java
package model.webUser;
import java.util.ArrayList;
import java.sql.ResultSet;

// The purpose of this class is to bundle a database error message with an array of StringData objects so that we can
// write all this in JSON format to communicate to the client side.

public class StringDataList {
    public String dbError = ""
    public ArrayList<StringData> webUserList = new ArrayList();

    // Default constructor leaves StringDataList with database error "" (empty string) and 0 elements in the list.
    public StringDataList() {
    }

    // Adds one StringData element to the array list of StringData elements
    public void add(StringData stringData) {
        this.webUserList.add(stringData);
    }

    // Adds creates a StringData element from a ResultSet (from SQL select statement),
    // then adds that new element to the array list of StringData elements.
    public void add(ResultSet results) {
        StringData sd = new StringData(results);
        this.webUserList.add(sd);
    }
}

SINGLE PAGE APPLICATION: A newer approach where a single HTML page uses JavaScript (and possibly AJAX) to inject content into a page. Older approach was for each link to get a whole new web page. Popular SPA Frameworks:

- Angular (written/supported by google and open source community) and
- Vue (written/supported by an ex Google employee and open source community).

React competes with the above two SPA frameworks even though React is a library (less invasive, less “bossy” than a framework). Currently React seems to be more popular than the other two (based on volumes of downloads).

React – study the concepts from this React tutorial page and study the associated code (zip file):

http://cis-linux2.temple.edu/~sallyk/tutorials_ReAct/index.php

The tutorial and associated code includes coverage of new ES6 syntax:

- let/const AND fat arrow functions,
- destructuring objects/arrays AND back tick and binding,
- Not covering the “class” and “inherits” keywords...

As well as these concepts from JavaScript functional programming:

- forEach, map, sort.

- Study the first two React sample projects: (1) navigation & (2) display data from:
  http://cis-linux2.temple.edu/~sallyk/tutorials_ReAct/index.php
Matching Answers

a. CDN: Content Delivery Network  
b. ES6 (EcmaScript 6)  
c. Single Page Application Framework  
d. SPA Library  
e. Responsive Design  
f. Dependency Injection  
g. PreparedStatement  
h. window.location.hash  
i. AJAX  
j. WinSCP  
k. Bootstrap  
l. jQuery  
m. GSON  
n. XML  
o. MVC  
p. JSON  
q. @media  
r. JSP  
s. fixed  
t. Chrome  
u. absolute  
v. z-index  
w. Apache  
x. ResultSet  
y. CSS  
z. float  
bb. this  
cc. Chrome  
dd. JSX  
ee. Connection  
ff. localStorage