**Assignment 2.2 – Values and Variables**

**Setup**

~~Each week you will be asked to create a new folder under web-231 following a naming convention of “week-<number>.” If we are on week two, the folder name should be “week-2.” All files associated with the weekly assignment will be added to the appropriate weekly folder. All programs must be linked in the index.html landing page under the “Weekly Assignments” section. Projects will be linked under the “Projects” section of the index.html landing page. To be clear,~~ **~~all~~** ~~of the JavaScript, HTML, images, and CSS files associated with a weekly assignment must be placed under the appropriate weekly folder. The page title for all HTML files in this course must say “WEB 231 – Enterprise JavaScript I.” And, all HTML and CSS files must be valid HTML/CSS, tested through the WC3 validator. The links were provided during WEB 200 and were added to the index.html landing page. Also, the blue border around the provided images is to show they are images and should not be included in your submission. In other words, do not add a blue border around your work, unless the instructions explicitly ask for it.~~

**User interface styling and formatting requirements are located in the HTML, CSS, and JavaScript Requirements document.**

HTML: **<yourLastName>-payroll.html**

CSS: **<yourLastName>-payroll.css**

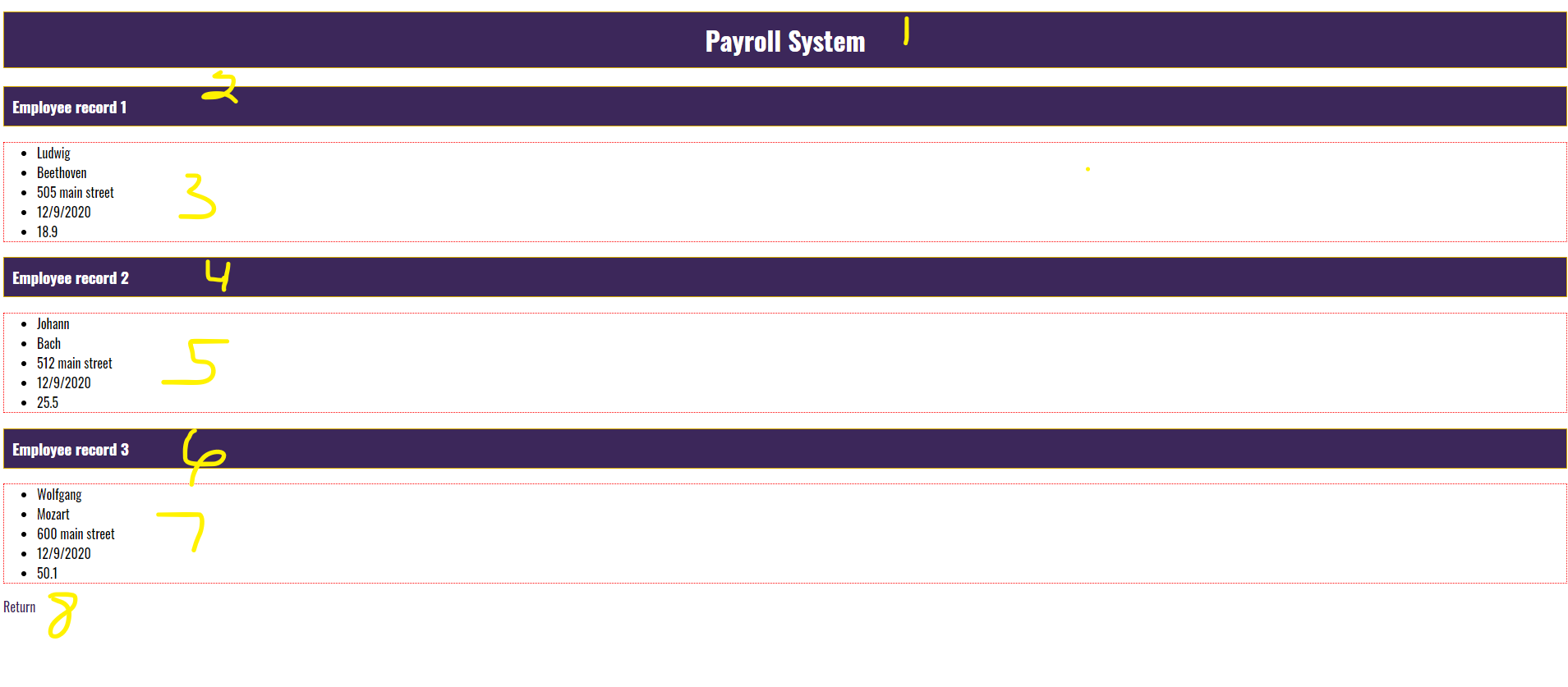
**Grading Reminders**

1. (50%-points) All code sources (.html, .css, .js) must be cited in the opening programmers’ comments, following the format specified in the code attribution document.
2. (25%-points) All code sources (.html, .css, .js) must show evidence of code comments. This means each section of the program (.html, .css, .js) must include code comments that explains what the block of codes purpose is, what the required parameters are (data type, if any), and what the expected output is.
3. (rubric) All code sources (.html, .css, .js) are measured against
   1. Code functionality: Does it work? Does it meet requirements?
   2. Adherence to standards and conventions. Are you using the appropriate data types, including proper indention, are variables named appropriate (variable x is an example of poor naming conventions), is there an appropriate use of whitespace, is the code organized, and are semicolons being used to terminate code sentences?
   3. Efficiency: Use of language features. Are you practicing DRY (Don’t-Repeat-Yourself?), are you leveraging built-in language features where appropriate, and are you using classes/functions to reduce code clutter?
   4. Documentation: Self-documenting, naming conventions, code is maintainable by others. Is the code your write easy to read and maintainable by others?
   5. Error trapping/handling. Are there errors in the program? Is there evidence of coding best practices to reduce user errors?
   6. Assignment Specific Compliance. Does the delivered solution follow the instructions, as they are written? Does the output match what was provided in the screenshots (including spaces, styling, etc.)?

**Required Modifications**

* Cite any sources in your opening programmer’s comment
* Link the appropriate CSS files and Google fonts

**Exhibit A. User Interface (final solution)**

****

1. ~~h1 tag with an id of payroll-header and a value of “Payroll System.” This styling is coming from the site.css file, which means you will need to link the site.css file in this HTML page. This is a special circumstance, meaning the rest of the assignments in this course will not link the site.css in their HTML file.~~
2. ~~h3 tag with a text value of “Employee record 1” and styling from the site.css file.~~
3. ~~an unordered list, with five list items. Give each list item an id that starts with the prefix~~ **~~txt~~** ~~and the name of the record you are creating. For example, txtFirstNameLudgwig, txtLastNameBeethoven, txtAddressLudgwig, txtHireDateLudwig, txtPayRateLudwig. The unordered list must have a dotted border, a one-pixel thickness, and be red. Don’t forget, one of the criteria’s you are graded against is naming convention best practices.~~

**JavaScript Requirements**

1. ~~Create JavaScript variables for all three sets of unordered lists. Each set of unordered lists must include variables for firstName, lastName, address, payRate, and hireDate. This means you will have a total of 3 variables for firstName, 3 variables for lastName, 3 variables for address, 3 variables for payRate, and 3 variables for hireDate.~~
2. ~~As shown above, format the payRate to one decimal place, which will require you to use the toFixed() position. Example,~~ **~~“12.999”.toFixed(1)~~**~~.~~
3. As shown above, format the hireDate as an actual JavaScript date object and covert to a local date string. Example, **new Date().toLocalDateString(“en-US”);**
4. ~~To bind the values, you will need to use innerHTML on each of the list items. of this document. Essentially, for every variable there will need to be a document.getElementById(“id”).innerHTML call that binds the variable value to the list~~ item div. If you are not sure how to do this, please see the red highlighted note I left at the top

**Code Example:**

document.getElementById(“txtFirstNameLudwig”).innerHTML = firstNameVariable;

1. h3
2. an unordered list
3. h3
4. an unordered list
5. anchor tag with a CSS class of return-home and a link back to the index.html landing page. The styling for the CSS class is coming from the global.css file.