**Quiz 1, Part 4 – Written Questions**

1. **Describe how HTML fits into the broader ecosystem of a website. Contrast the fundamental role of HTML with the primary roles of CSS and JavaScript.** (*5 points*)  
     
   HTML lays the framework for the various files, scripts, and tagged elements that are to be rendered by the browser. This includes both elements that the user will see on screen (e.g., <h1> for headers, <p> for paragraphs, <ol> for ordered lists, etc.) as well as components that reference other files, or otherwise live in the HTML file itself but aren’t shown to the user (e.g., images, icons, CSS files, JS scripts).  
     
   CSS, on the other hand, is used to style the visual elements within an HTML document. CSS styles act as “embellishments” to the HTML document; While CSS styles are not absolutely necessary for the HTML site to display its information to the user, they are used to make the HTML more visually appealing and, as is often the goal with UI/UX Design, optimize the user’s experience of the site/understanding of the information.  
     
   JavaScript, however, plays a more important role than CSS in the broader ecosystem of a website. JavaScript allows for dynamic DOM manipulation, meaning that it can construct both HTML and CSS, and inject these into an HTML file using <script> tags. It can also be used for fetching/parsing data from APIs, as well as creating complex dynamic/animated site visuals and interactive elements (e.g., buttons that trigger/toggle visual events, etc). The capabilities of JavaScript go far beyond the tasks listed here, of course.  
     
   Overall, an HTML file acts as an “aggregator” for all of the visual/non-visual components that comprise the page that gets served to the user.
2. **Explain the difference between HTML structure and HTML semantics. Why is writing semantic HTML considered a best practice? Provide one example of a semantic HTML element and one example of a non-semantic element.** (*10 points*)  
     
   HTML structure refers to the hierarchy (and, consequently, the layout) of elements that comprise an HTML document. HTML structure dictates the logical flow of an HTML document, and is more pertinent to the developer of the site than the user. Structural HTML prioritizes use of tags like <div>, which, on their own, do not carry any semantic meaning (i.e., these are general-purpose tags, used for a variety of different organizational purposes).   
     
   On the other hand, HTML semantics refers to the meaning of the underlying content, with HTML that is more specifically geared towards the content it is meant to represent (e.g., <header> corresponds to header information, <article> corresponds to a self-contained article, <ol> corresponds to an ordered list, <li> corresponds to a list item within a list, etc.).  
     
   Writing semantic HTML is generally considered to be a best practice because it allows for the highest degree of user accessibility. For instance, if a user is visually impaired / unable to see the content of a website, they may take advantage of accessibility tools like screen readers that read aloud the contents of a webpage. If a webpage is written structurally, this may prove to be difficult, as a structural HTML site is clouded with tags that lack semantic meaning and are purely structural, causing the user to struggle to parse the information. By contrast, if a webpage is written semantically, a user will have a much easier time parsing and understanding the information on the site.
3. **What is the "three-tier model" (also known as three-tier architecture) in web systems? Briefly describe the function and responsibility of each of the three tiers.** (*15 points*)  
     
   In web systems, the “three-tier architecture” is a software design pattern that compartmentalizes a web app’s components into three primary groups: (1) the Presentation Tier, (2) the Application Tier, and (3) the Data Tier. Each of these three groups can be characterized by their level of separation from the user experience, with the Presentation Tier being directly concerned with the user experience, and the Data Tier being the most far-removed from it.  
     
   The Presentation Tier refers to all of a web app’s components that are specifically related to (and ultimately define) the user interface. Its function/responsibility is to act as the user’s interface with the underlying web app’s tools and data, providing a level of abstraction from any unnecessary web app information, like implementation details or sensitive data. This tier consists of HTML, CSS, and JavaScript.  
     
   The Application Tier refers to the components of a web app responsible for actually processing/manipulating data. Its function/purpose is to take user requests from the Presentation Tier, and either retrieve data to be processed or store new data in the Data Tier. To put it simply, this tier acts as an interface between the Presentation Tier and the Data Tier, responsible for parsing user requests and executing the processes necessary to either store/manipulate the underlying data to produce the user’s intended result. This tier can consist of a number of different programming languages, including Java, JavaScript, and Python.  
     
   The Data Tier refers to the components that assist in storing and managing the application’s data. Its function/purpose is to handle storing, retrieving, and updating database information. It is important to note that the Data Tier is not responsible for any manipulation/processing of data beyond simply storing, retrieving, and updating (i.e., the Data Tier does not perform any functions or operations to manipulate the data itself), as these are handled in the Application Tier. This tier generally consists of database management tools, like MySQL, MariaDB, and MongoDB, to name a few.
4. **Explain what is meant by a Uniform Interface in a REST API.** (*5 points*)  
     
   In a REST API, Uniform Interface means that the way that a client interacts with an API does not change across different client applications and/or resources that the client may be trying to retrieve (i.e., a client’s interface with an API is uniform/standardized). This works when (1) each resource in the system has a unique identifier; (2) the clients interact with an abstracted, standard representation of a resource; (3) the responses from an API are self-descriptive and contain enough stand-alone information for the client to figure out how to process them; and (4) the responses contain links that guide the client on further actions to be taken, so they do not need to lean on external information for guidance.
5. **Explain how your browser chooses which CSS rule to apply to a tag in the case where there are multiple rules that could apply.** (*15 points*)  
   When there are multiple CSS rules that could apply to a single tag, the browser must consider a number of factors to determine which CSS rule gets applied. These factors are importance, origin, specificity, and source order. To illustrate each of these factors, consider the following example files. Assume that styles.css is linked in the header of index.html. I have omitted this for brevity.

|  |  |  |
| --- | --- | --- |
| index.html |  | styles.css |
| <h1 style=”color: blue”> Hello, world! </h1> |  | h1 { color: red; } |

Importance refers to how rules designated as !important override normal rules. For instance, if I were to designate the “color: red” rule as !important in styles.css, this would override the “color: blue” inline style rule applied in index.html.   
**Result**: The browser would display **Hello, world!** in red.  
  
Origin refers to how inline styles override both internal and external stylesheets. For instance, as the code is written above (with no changes), since inline style rules override internal/external style sheets, and styles.css is considered an external style sheet, the inline style rule “color: blue” would override the “color: red” rule for <h1>.  
**Result**: The browser would display **Hello, world!** in blue.  
  
Specificity refers to the specific hierarchy rules between IDs, classes, and element selectors.  
From highest to lowest specificity, the hierarchy is ordered (1) Inline styles, (2) IDs, (3) Class, attribute, and pseudo-class selectors, and (4) Element selectors. Suppose I made the following **changes** to the above example (I removed the inline color rule as inline styles override everything):

|  |  |  |
| --- | --- | --- |
| index.html |  | styles.css |
| <h1 id=”one”> Hello, world! </h1> |  | h1 { color: red; }  #one { color: green; } |

Since ID selectors (#one) have a higher specificity than element selectors (h1), the rule “color: green” would override the “color: red”.  
**Result**: The browser would display **Hello, world!** in green.  
  
Source order refers to how the browser selects between two rules with the same specificity. The one that appears last/most recently is selected. Suppose I made the following **changes**:

|  |  |  |
| --- | --- | --- |
| index.html |  | styles.css |
| <h1> Hello, world! </h1> |  | h1 { color: red; }  h1 { color: blue; } |

Since the specificity of h1 is equal to h1, the latest rule “color: blue” overrides “color: red”.  
**Result**: The browser would display **Hello, world!** in blue.