Exam 2 Study Guide

Overview: This exam will assess your ability to analyze and modify HTML and advanced CSS to debug issues and address challenges in accessibility, responsive design, and functionality enhancements for an existing website. You should be comfortable making small, targeted code changes related to the concepts covered in Weeks 4 to 9.

Table of Contents

Key Concepts to Study

1. **Viewing Deployed Pages vs. Source Code**:
    - Understand the difference between viewing a deployed website vs. accessing its source code. If someone shares the link *colleen.github.io*, you will see the **deployed page**, not the underlying code unless explicitly shared. Understand how you can navigate between a deployed website and its source code.
2. **Advanced CSS:**
    - Advanced Selectors:
        - Learn how to select elements in CSS by `name`, `class`, and `ID`. All of these methods are valid ways to target elements, and it's important to understand the specificity that each one brings.
        - **Attribute Selectors:** Target elements based on attributes or attribute values, such as `input[type="checkbox"]`.
    - CSS Combinators:
        - Child (>): Targets direct children (e.g., `ul > li`).
        - Descendant ( ): Targets any matching descendants within an element (e.g., `div p`).
        - Next-sibling (+): Targets an element that is placed immediately after another specific element (e.g., `div + p`).
        - Selector list (,): Targets all the matching elements separated by a comma (e.g. `div, p`).
    - Box Model:
        - When calculating the total width and height of an element, remember to account for padding, borders, and margins. For example, if an element has a `width: 200px`, `padding: 10px`, and a `2px` border, the total width would be **224px**. Understanding the CSS box model will help you troubleshoot layout issues.
        - Understand that if you set `box-sizing: border-box;` on an element, padding and border are included in its width and height.
    - Display:
        - The `display` property specifies the display behavior (the type of rendering box) of an element.
        - `block`: Element takes up the full width available.
        - `inline`: Element only takes as much space as needed.
        - `none`: Hides the element.
        - `flex` and `grid`: Layout systems for arranging elements in containers.
    - Pseudo-classes:
        - Learn about different CSS pseudo-classes such as `:hover`, `:focus`, `:visited`, and `:active`. Understand when each is applied to elements (e.g., `:hover` is used when a user hovers over an element).
    - Pseudo-elements:

        - Learn about different CSS pseudo-elements such as `::before` and `::first-letter`. These allow you to style specific parts of an element's content without modifying the HTML.
        - `::before`: Inserts content before the actual content of an element. For example, `h1::before { content: "→"; }` adds an arrow before every `h1`. This is useful for adding decorative elements like icons or separators.
        - `::first-letter`: Styles the first letter of a block-level element, often used to create drop caps or emphasize the initial letter. For example, `p::first-letter { font-size: 2em; }` enlarges the first letter of each paragraph.
3. **Grid:**
    - Refer to these notes, created by Blair Bocklet; they're a great resource to learn all about the grid!
4. **Accessibility Practices:**
    - Validator Tools:
        - Familiarize yourself with the key tools that can help you validate your code for errors.
        - **HTML Validators**: Ensure your HTML structure is correct and follows web standards.
        - **Accessibility Auditing Tools**: Tools like WAVE and aXe help you check for accessibility issues on your site, such as missing `alt` attributes or color contrast problems.
    - Skip Links:
        - People using keyboard, screen readers, switch controls and other assistive technologies use skip links to reach main content or other important sections easier and faster.
        - Skip to Main Content:
            - The most common skip link, Skip to Main Content, is the first interactive element on a page. It takes the user to the main content, past the global elements like the logo, search and navigation. It is almost always hidden until it receives focus.
            - Understand how to implement these links without modifying the HTML or JavaScript
5. **DevTools for Debugging:**
    - Inspect Element:
        - **HTML Structure**: Use the Inspect Element tool to view the structure of your webpage and inspect individual elements.
        - **Applied Styles**: Inspect Element shows all the CSS rules applied to an element, including those overridden by more specific or later rules.
        - **Computed Styles**: The Computed Styles tab displays the final rendered CSS properties, after all cascading and inheritance has been applied.

        - **Box Model Visualization**: Inspect Element allows you to visualize the CSS box model, showing the padding, borders, and margins of elements.
        - **Diagnosing Visibility Issues**: Use the tool to identify if an element is hidden, clipped, or off-screen. It's particularly helpful for finding elements affected by `z-index` issues, `display: none`, or overflow issues.
    - Console:
        - Used to log errors and debug JavaScript.
        - Can test CSS changes dynamically to check how they impact the page.
        - Can view missing files, invalid API calls, etc.
6. **Responsive Design:**
    - Mobile-first Design:
        - A development strategy where the layout and styles are initially designed for small screens (like mobile devices), and then enhanced for larger screens through media queries.
        - This approach ensures that a website works well on the smallest screens by default and becomes progressively more complex and feature-rich for larger devices.
        - Start with basic CSS for mobile devices and use **media queries** to apply additional styles for larger screens.
    - Breakpoints:
        - Breakpoints define screen widths at which the layout or design needs to change for optimal display on different devices.
            - Understand why we use `min-width` to define breakpoints as an accessibility-first approach
    - Media Queries:
        - Media queries allow you to apply CSS styles conditionally based on the characteristics of the user's device (such as screen width, resolution, or user preferences). They help create **responsive designs** that adapt to different screen sizes and user settings, improving accessibility and usability.
        - `prefers-reduced-motion`: This checks if the user has enabled reduced motion settings on their device. This helps accommodate users who experience motion sensitivity by disabling or minimizing animations and transitions.
            - Understand how to implement this using CSS.
        - `prefers-color-scheme`: The prefers-color-scheme media query adjusts the site's color scheme based on the user's system preference (e.g., dark mode or light mode). This ensures the design integrates seamlessly with the user's environment, improving both aesthetics and usability.
            - Understand how to implement this using CSS.
7. **CSS Transitions:**
    - Transitions allow you to animate changes to CSS properties smoothly over a specified duration.

- They enhance user experience by making style changes (like color or position shifts) feel more fluid and interactive.
- Understand how to combine CSS selectors, combinators, pseudo-classes, pseudo-elements and apply transitions on them.
    - For example: When the user hovers over the button, the background color changes to blue over 0.3 seconds with an ease-in-out timing function.

8. **Forms:**
    - It is crucial to understand the tags used in forms, the different input types, and how to ensure accessibility compliance.
    - Elements:
        - `form`: The container for all form elements. It defines where form data is sent when submitted.
        - `fieldset`: Groups related form inputs into logical sections. It helps organize the form visually and semantically by creating a distinct boundary for related fields (e.g., multiple radio buttons for selecting options).
        - `legend`: This serves as the title or description for the contents inside the `<fieldset>`. It gives the group of inputs meaning by describing what the inputs represent.
        - `label`: Associates a descriptive label with an input field, making the form accessible by allowing users (especially those using screen readers) to understand what each input is for.
        - `input`: Creates various interactive elements where users enter data (e.g., text fields, checkboxes, radio buttons).
        - `name`: This attribute identifies form data during submission. The value assigned to name is used as the key in the data sent to the server.
    - Input Types:
        - Implement various input types to improve the user experience by ensuring that the correct input field is used for specific data types.
        - Each input type presents a different UI element to the user.
        - Understand the various input types, such as `text`, `email`, `password`, `checkbox`, etc.

Example Scenarios

1. **Troubleshooting HTML/CSS**:
    - You may be asked to troubleshoot why text or images might not be displaying as expected. Be prepared to figure this out using the various debugging tools at hand, such as Inspect Element and the Console.
2. **Writing Advanced CSS Rules**:
    - Write Advanced CSS rules to target specific elements in particular states, like adding a border to an image upon hover.
3. **Inspecting and Modifying Styles in Real-Time**:
    - Use Chrome's Inspect Element to identify which CSS rules are being applied or overridden. Modify them in real-time to see the effects on layout and visibility.
4. **Creating Responsive Designs:**
    - Modify the photo gallery to be a 2-column grid for screen sizes smaller than 700px (e.g. a mobile phone), and a 4-column grid otherwise.
5. **Ensuring Accessible Design using Media Queries:**
    - Identify the spinning logo on the website and replace it with a static logo if the user prefers reduced motion on their device.

Study Tips

- Review the code and concepts discussed in class, focusing on accessibility best practices and implementing **skip links** to improve navigation for assistive technology users.
- Practice using **Inspect Element** and **Console** in Chrome to inspect HTML structure, modify CSS styles in real-time, and troubleshoot missing files, layout issues, and visibility problems.
- Study the cascading nature of CSS, with special attention to specificity, inheritance, and rule order. Understand how these concepts affect which CSS rules are applied or overridden.
- Experiment with adjusting styles and inspecting the results to strengthen your knowledge of the CSS **box model**, including how padding, borders, and margins affect layout and element positioning.
- Test various **media queries** by resizing the browser window or using Chrome's responsive mode. Focus on implementing **breakpoints**, prefers-reduced-motion, and prefers-color-scheme queries to ensure your designs are responsive and accessible.
- Analyze and practice building **grid layouts** using display: grid and grid-template-columns. Create layouts that switch between 2 and 4 columns based on screen size, simulating scenarios likely to appear in the exam.
- Work with **forms** to understand how to properly use and style tags like label, name, and input. Practice validating forms by experimenting with different input types (e.g., text, email, checkbox) and ensuring proper fieldset grouping using <fieldset> and <legend>.
- Simulate troubleshooting common visibility issues by hiding elements with display: none and diagnosing their absence using **DevTools** (like Inspect Element and Console).
- Practice writing **advanced CSS selectors** (e.g., attribute selectors, child and descendant combinators). Create CSS rules to target specific elements in different states (e.g., :hover, :nth-child, ::before).
- Familiarize yourself with **accessibility auditing tools** (like WAVE or aXe) to check for common accessibility issues, such as missing alt attributes or color contrast problems.

By mastering these topics and using the **Inspect Element tool + Console** effectively, you will be well-prepared for both the conceptual and practical parts of the test.