**Assignment CSS Selectors and Specificity**

**Instructions:**

1. Universal Selector: Write CSS rules using the universal selector (\*) to target all HTML elements on a webpage. Apply different styles, such as changing font color, background color, and font size.

2. Element Selector: Create a CSS rule that targets specific HTML elements (e.g., all <h1> elements) and style them differently from other elements on the page.

3. Class Selector: Define multiple CSS classes and apply them to HTML elements with different styles. Create a class selector rule that targets and styles elements with a specific class.

4. ID Selector: Create an HTML element with a unique ID and apply a CSS rule targeting this ID. Style the element uniquely using the ID selector.

5. Selector Specificity: Explain the concept of specificity in CSS. Provide examples of different selectors (element, class, ID) and discuss how they affect the specificity of CSS rules.

6. Inline Specificity: Demonstrate the use of inline styles in HTML. Create an HTML element and apply inline styles to it. Explain how inline styles affect specificity and why they should be used sparingly.

7. Pseudo-Class: Choose an HTML element (e.g., links) and apply pseudo-classes like :hover and :active to create interactive effects. Explain how pseudo-classes work and when they are useful.

8. Pseudo-Element: Create a CSS rule using a pseudo-element (e.g., ::before or ::after) to insert content before or after an HTML element. Describe the purpose and usage of pseudo-elements.

9. Cascading Stylesheet: Explain the concept of a cascading stylesheet in CSS. Discuss the order of importance when multiple styles are applied to the same element.

10. !important: Create a CSS rule with the !important declaration to override other conflicting styles. Discuss when and why using !important should be avoided and provide alternatives for managing style conflicts.

**Submission Guidelines:**

Create an HTML document containing the examples and explanations for each part of the assignment.

Organize your code and explanations clearly.

Ensure your code is well-commented for clarity.

Include a separate section for each part of the assignment.

If you encounter any issues or challenges during the assignment, describe them and how you resolved them (if applicable).