<input> VS <textarea> for CSS and HTML point of view:

In HTML, both `<input>` and `<textarea>` elements are used for collecting user input, but they serve different purposes and have distinct characteristics. Here's a comparison of their uses:

`<input>` Element

- **Purpose**: Used for single-line text input fields.
- **Syntax**: `<input type="text">`
- **Use Cases**:
  - **Single-line input**: For short inputs like names, emails, or passwords.
  - **Specialized inputs**: When combined with different `type` attributes (e.g., `email`, `password`, `number`, etc.), the `<input>` element can be used for various data types.
- **Attributes**:
  - `value`: Default value of the input.
  - `maxlength`: Limits the maximum number of characters.
  - `placeholder`: Placeholder text to guide the user.
  - `type`: Defines the input type (e.g., `text`, `password`, `email`).

`<textarea>` Element

- **Purpose**: Used for multi-line text input.
- **Syntax**: `<textarea></textarea>`
- **Use Cases**:
  - **Multi-line input**: For larger text inputs such as comments, messages, or detailed descriptions.
  - **Flexible sizing**: Can be resized both vertically and horizontally, providing more room for text entry.
- **Attributes**:
  - `rows` and `cols`: Specifies the number of rows (height) and columns (width) for the text area.
  - `maxlength`: Limits the number of characters.
  - `placeholder`: Placeholder text.

Key Differences

- **Single-line vs. Multi-line**: `<input>` is typically used for single-line text, while `<textarea>` is used for multi-line text.
- **Element Closing**: `<input>` is a self-closing element, whereas `<textarea>` requires both opening and closing tags (`<textarea></textarea>`).
- **Resizing**: `<textarea>` can be resized by the user (unless disabled by CSS), whereas `<input>` is fixed in size unless controlled by CSS.

Example Usage

html

```
<!-- Input field for single-line text -->
<input type="text" placeholder="Enter your name">

<!-- Textarea field for multi-line text -->
<textarea rows="5" cols="30" placeholder="Enter your
message"></textarea>
```

Use `<input>` for simpler, short inputs and `<textarea>` for longer, multi-line inputs.

To capture the input values from an `<input>` field and a `<textarea>` field in JavaScript, you can use the `value` property of each element and store them in separate strings.

Here's a simple example:

HTML

```html
<!-- Input for single-line text -->
<input type="text" id="nameInput" placeholder="Enter your name">

<!-- Textarea for multi-line text -->
<textarea id="messageInput" rows="5" cols="30" placeholder="Enter your
message"></textarea>

<!-- Button to trigger the action -->
<button onclick="getInputValues()">Submit</button>
```

JavaScript

```javascript
function getInputValues() {
    // Capture the value from the input field (single-line)
    let name = document.getElementById("nameInput").value;

    // Capture the value from the textarea (multi-line)
    let message = document.getElementById("messageInput").value;

    // Display the values in the console or use them as needed
    console.log("Name: " + name);
    console.log("Message: " + message);

    // Optionally, store them in different strings
    let nameString = name;
    let messageString = message;

    // Example: Display them on the webpage (optional)
    document.body.innerHTML += `<p><strong>Name:</strong>
${nameString}</p>`;
```

```
    document.body.innerHTML += `<p><strong>Message:</strong>
${messageString}</p>`;
}
```

Explanation:

1. **`document.getElementById("nameInput").value`**: This fetches the value entered by the user in the input field with the ID `nameInput`.
2. **`document.getElementById("messageInput").value`**: This fetches the value from the textarea with the ID `messageInput`.
3. **Button**: The button calls the `getInputValues()` function when clicked, which retrieves the values and stores them in two separate strings: `nameString` and `messageString`.

When you press the **Submit** button, the values entered by the user will be stored in two different strings, and in this example, they are printed to the console and optionally displayed on the webpage.

The `<input>` and `<textarea>` elements have different properties that can be styled with CSS to control their appearance and behavior. Here's a breakdown of the common and unique properties for both:

Common CSS Properties for Both `<input>` and `<textarea>`

- **`width`**: Specifies the width of the input/textarea.
- **`height`**: Specifies the height of the input/textarea (useful for `<textarea>`).
- **`padding`**: Adds space inside the input/textarea.
- **`margin`**: Adds space outside the input/textarea.
- **`border`**: Sets the border style, width, and color.
- **`border-radius`**: Rounds the corners of the input/textarea.
- **`background-color`**: Sets the background color.
- **`color`**: Changes the text color.
- **`font-size`**: Sets the size of the text inside the input/textarea.
- **`font-family`**: Sets the font family for the text inside the input/textarea.
- **`outline`**: Controls the outline of the input/textarea (often used to remove default focus outlines).
- **`box-shadow`**: Adds shadow effects to the input/textarea (useful for creating depth).
- **`text-align`**: Aligns text inside the input/textarea.
- **`cursor`**: Controls the type of cursor that appears when hovering over the input/textarea.
- **`:hover` and `:focus`**: Pseudo-classes to apply styles when the element is hovered over or focused.

Unique Properties for `<input>`

- **`type`-based Styles**: Different types of `<input>` elements (e.g., `text`, `password`, `email`, `number`) can be styled differently using CSS attributes like:
    - **`input[type="text"]`**
    - **`input[type="password"]`**
    - **`input[type="email"]`** These allow you to apply custom styles to specific types of inputs.

`placeholder-color`: You can style the placeholder text using the following:

```
input::placeholder {
    color: gray;
    font-style: italic;
}
```

Unique Properties for `<textarea>`

`resize`: Controls the ability of the user to resize the textarea. It can be set to `none`, `both`, `horizontal`, or `vertical`.

```
textarea {
    resize: none; /* Prevent resizing */
}
```

`rows` **and** **`cols`** **(HTML attributes)**: Controls the visible dimensions of the `<textarea>` without needing CSS (although `width` and `height` in CSS are usually preferred).
html
Copy code
```
<textarea rows="5" cols="40"></textarea>
```

Example CSS for Styling `<input>` and `<textarea>`

```
/* General input and textarea styles */
input, textarea {
    width: 100%;           /* Make input/textarea take full width */
    padding: 10px;          /* Add padding inside the element */
    border: 1px solid #ccc; /* Add a light gray border */
    border-radius: 5px;     /* Slightly round the corners */
    font-size: 16px;        /* Set font size */
    box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1); /* Add a subtle shadow
*/
}

/* Remove resizing for textarea */
textarea {
    resize: none;
}

/* Change styles when input or textarea is focused */
input:focus, textarea:focus {
    border-color: #4A90E2; /* Change border color on focus */
    outline: none;         /* Remove default outline */
    box-shadow: 0 0 5px rgba(74, 144, 226, 0.5); /* Add focus shadow
*/
}

/* Style for input placeholder text */
input::placeholder, textarea::placeholder {
    color: #999;
    font-style: italic;
}
```

Summary

Common Properties: Many CSS properties like `width`, `padding`, `font-size`, `color`, and `border` are common to both `<input>` and `<textarea>`.

Unique Properties: `<textarea>` has properties like `resize` and uses HTML attributes like `rows` and `cols`. Different types of `<input>` fields can be styled based on the `type` attribute, and placeholders can be styled separately for both elements using the `::placeholder` pseudo-element.