**Built-in HTML Validation Techniques**

**1. Understanding HTML5 Validation**

HTML5 provides built-in **form validation**, so you can **validate user input** without writing extra JavaScript code. The browser automatically checks the input when the form is submitted.

**1.1 Form Validation Attributes**

These attributes can be added directly to your input fields:

| **Attribute** | **Purpose** |
| --- | --- |
| required | Field must be filled before form submission |
| pattern | Regular expression for matching a specific format |
| min / max | Minimum and maximum values for number/date inputs |
| step | Interval for number/date input (e.g., step="2" → 0, 2, 4, 6...) |
| maxlength | Max number of characters allowed |
| minlength | Minimum number of characters required |

**Example:**

<form>

<input type="text" name="username" required minlength="3" maxlength="10" />

<input type="number" min="1" max="100" step="5" />

<input type="text" pattern="[A-Za-z]{3,}" title="Only letters, min 3" />

<button type="submit">Submit</button>

</form>

**1.2 Input Types with Built-in Validation**

HTML5 introduced input types that automatically validate data.

| **Input Type** | **Example Value** | **Validation Done By Browser** |
| --- | --- | --- |
| email | test@example.com | Must be a valid email format |
| url | <https://example.com> | Must be a valid URL |
| number | 1, 2, 3... | Allows only numeric input |
| date | yyyy-mm-dd | Must be a valid date format |
| tel | Phone numbers | Often used with pattern attribute |

**Example:**

<form>

<input type="email" required />

<input type="url" />

<input type="date" />

<input type="tel" pattern="[0-9]{11}" title="Enter 11-digit phone number" />

<button type="submit">Submit</button>

</form>

**Styling Validation States**

CSS allows us to give **visual feedback** when a form is valid or invalid, using **pseudo-classes**.

**2.1 Pseudo-Classes**

| **Pseudo-Class** | **Description** |
| --- | --- |
| :valid | Input is valid |
| :invalid | Input is invalid |
| :required | Input has required attribute |
| :optional | Input does NOT have required attribute |

**Example:**

input:valid {

border: 2px solid green;

}

input:invalid {

border: 2px solid red;

}

input:required {

background-color: #fffcf5;

}

**2.2 Visual Feedback Example**

<style>

input:valid {

border: 2px solid green;

}

input:invalid {

border: 2px solid red;

}

</style>

<form>

<label>Email:</label>

<input type="email" required />

<br><br>

<label>Username:</label>

<input type="text" required minlength="3" />

<br><br>

<button type="submit">Submit</button>

</form>

**Browser will show green border if input is valid, red if not.**

**3. Common Use Cases**

**3.1 Contact Form Example**

<form>

<input type="text" placeholder="Your Name" required />

<input type="email" placeholder="Email" required />

<textarea placeholder="Message" required minlength="10"></textarea>

<button type="submit">Send</button>

</form>

**3.2 Login Form**

html

CopyEdit

<form>

<input type="text" placeholder="Username" required />

<input type="password" placeholder="Password" required minlength="6" />

<button type="submit">Login</button>

</form>

**3.3 Sign-Up Form**

<form>

<input type="text" placeholder="Username" required />

<input type="email" placeholder="Email" required />

<input type="password" placeholder="Password" required minlength="8" />

<input type="tel" pattern="[0-9]{11}" title="Enter 11-digit number" />

<button type="submit">Register</button>

</form>

**Custom Form Validation with JavaScript**

**1. JavaScript Validation Techniques**

JavaScript allows us to manually validate form input fields. Instead of relying only on the browser's default validation, we can add custom rules using JavaScript.

**Manual Validation Methods**

These methods are used to **read and check** the values of form fields and control how validation works.

**1. input.value**

**What it does:**  
This gets the value entered by the user in an input field.

let nameInput = document.getElementById("name");

console.log(nameInput.value); // shows user input

**Explanation:**  
You can use .value to check what the user has typed. This is helpful for comparing it with your validation rules (e.g., length, format, etc.).

**input.validity**

**What it does:**  
Returns an object that gives you detailed info about which rule failed (e.g., valueMissing, typeMismatch, etc.).

if (!nameInput.validity.valid) {

console.log("Name input is not valid!");

}

**Explanation:**  
You can use .validity to check exactly what went wrong (e.g., if the field is empty or the email is invalid). This helps with **customized error messages**.

**input.checkValidity()**

**What it does:**  
Checks if the input is valid according to HTML5 rules and returns true or false.

if (!nameInput.checkValidity()) {

alert("Please enter a valid name.");

}

**Explanation:**  
You can use this before submitting a form to make sure the fields are valid. It helps stop invalid data from being submitted.

**Custom Validation: setCustomValidity(message)**

**What it does:**  
This method lets you create your **own error messages** for validation instead of using the browser's default.

let ageInput = document.getElementById("age");

if (ageInput.value < 18) {

ageInput.setCustomValidity("You must be at least 18 years old.");

} else {

ageInput.setCustomValidity(""); // Clears the custom error

}

**Explanation:**  
Sometimes the default error isn’t enough. Using setCustomValidity() helps show **meaningful, user-friendly messages**.

**2. Handling Form Validation Events**

You can listen for certain **events** in the form or input fields to apply validation rules dynamically as the user interacts with the form.

**Event Listeners**

| **Event** | **When it triggers** |
| --- | --- |
| input | When user types in a field |
| change | When user changes and leaves the field |
| blur | When the field loses focus |
| submit | When the form is submitted |

**Example:**

let emailInput = document.getElementById("email");

emailInput.addEventListener("input", () => {

if (!emailInput.value.includes("@")) {

emailInput.setCustomValidity("Email must contain '@'");

} else {

emailInput.setCustomValidity("");

}

});

**Explanation:**  
This allows **real-time feedback**. When the user types something wrong, we show an error. When they correct it, we remove the error.

**Validation on Form Submission**

**What it does:**  
Prevents the form from submitting if the validation fails.

let form = document.getElementById("myForm");

form.addEventListener("submit", (e) => {

if (!form.checkValidity()) {

e.preventDefault(); // Stops the form from submitting

alert("Form has errors. Please correct them.");

}

});

**Explanation:**  
This ensures only **valid data is submitted**. If the form fails validation, the submission is blocked and the user sees an alert.

**3. Displaying Validation Messages**

Showing errors in a clear way is important for user experience.

**Custom Error Messages using setCustomValidity()**

**What it does:**  
Allows you to **replace the default browser messages** with your own custom message.

if (password.value.length < 6) {

password.setCustomValidity("Password must be at least 6 characters.");

}

**Explanation:**  
This makes your messages more **understandable and user-friendly**, especially if your form is in a local language or needs custom wording.

**Real-Time Validation Feedback**

**What it does:**  
Shows or hides **custom messages** while user is typing.

**HTML:**

<input type="text" id="username" required />

<small id="usernameError" style="color:red;"></small>

**JavaScript:**

let username = document.getElementById("username");

let errorMsg = document.getElementById("usernameError");

username.addEventListener("input", () => {

if (username.value.length < 4) {

username.setCustomValidity("Too short");

errorMsg.textContent = "Username must be at least 4 characters.";

} else {

username.setCustomValidity("");

errorMsg.textContent = "";

}

});

**Explanation:**  
This gives **instant feedback**, so users don’t have to wait until form submission to know what’s wrong.

**Complete Working Example**

<form id="registerForm">

<label>Name: <input type="text" id="name" required /></label><br />

<label>Age: <input type="number" id="age" required /></label><br />

<label>Email: <input type="email" id="email" required /></label><br />

<button type="submit">Register</button>

</form>

<script>

let form = document.getElementById("registerForm");

let age = document.getElementById("age");

let email = document.getElementById("email");

age.addEventListener("input", () => {

if (age.value < 18) {

age.setCustomValidity("You must be at least 18.");

} else {

age.setCustomValidity("");

}

});

email.addEventListener("input", () => {

if (!email.value.includes("@")) {

email.setCustomValidity("Invalid email format.");

} else {

email.setCustomValidity("");

}

});

form.addEventListener("submit", (e) => {

if (!form.checkValidity()) {

e.preventDefault();

alert("Form is invalid. Please fix errors.");

}

});

</script>

**Explanation:**  
This is a full working example of a form with manual and custom validation, using input events and validation messages.

**Advanced Form Validation Techniques in JavaScript**

**1. Combining Built-in and Custom Validation**

**What does this mean?**

We can use:

* **Built-in validation** using HTML5 attributes like:
  + required
  + type="email"
  + minlength, etc.
* And also add **extra rules** using JavaScript for full control.

**Example:**

<form id="myForm">

<input type="email" id="email" required />

<span id="emailError" style="color:red;"></span>

<button type="submit">Submit</button>

</form>

javascript

CopyEdit

let email = document.getElementById("email");

let emailError = document.getElementById("emailError");

email.addEventListener("input", function () {

if (!email.validity.valid) {

emailError.textContent = "Please enter a valid email.";

} else {

emailError.textContent = "";

}

});

**Explanation:**

* We are using HTML’s type="email" and required → this is **built-in validation**.
* Then using JavaScript, we check if the email is valid.
* If not valid, we show a custom error message.

**2. Real-time Validation Handling**

**What is Real-time Validation?**

It means **checking the input while the user is typing**.  
No need to wait for form submission.

**Problem:**

If we check input on every key press, it becomes too fast and heavy.

**Solution: Debouncing**

**Debouncing = Delay the function call** until the user stops typing.

**Example with Debounce:**

<input type="text" id="username" />

<span id="usernameError" style="color:red;"></span>

javascript

CopyEdit

let username = document.getElementById("username");

let usernameError = document.getElementById("usernameError");

let timeout;

username.addEventListener("input", () => {

clearTimeout(timeout); // Cancel old timer

timeout = setTimeout(() => {

validateUsername(username.value);

}, 500); // Wait for 500ms

});

function validateUsername(value) {

if (value.length < 4) {

usernameError.textContent = "Username must be at least 4 characters.";

} else {

usernameError.textContent = "";

}

}

**Explanation:**

* We add an input listener.
* When user types, we wait for **0.5 seconds** before validating.
* If user types again within 0.5s, we cancel the last check.
* This saves performance and gives smooth experience.

**3. Validation Best Practices**

**A. Avoiding Over-Validation**

**Problem:**  
Too many validations = user gets frustrated.

**Best Practice:**

* Only validate important things like:
  + Required fields
  + Email format
  + Password strength
* Avoid checking too many small things.

**B. Providing Helpful Feedback**

**Good UI helps the user fix mistakes quickly.**

Show:

* Clear error messages
* Colored borders (red/green)
* Small hints near the input field

**Example:**

<input type="password" id="pass" />

<small id="passError" style="color:red;"></small>

javascript

CopyEdit

let pass = document.getElementById("pass");

let passError = document.getElementById("passError");

pass.addEventListener("input", () => {

if (pass.value.length < 6) {

passError.textContent = "Password must be at least 6 characters.";

pass.style.borderColor = "red";

} else {

passError.textContent = "";

pass.style.borderColor = "green";

}

});

**Explanation:**

* When the user types the password, we check the length.
* If it’s less than 6 characters:
  + Show a message
  + Make the border red
* If okay:
  + Remove the message
  + Make the border green