Week 10 Readings

MDN: Validating forms

Client Side form validation: Before submitting data to the server, it is important to ensure all required form controls are filled out, in the correct format. This helps ensure data submitted matches the requirements set forth in the various form controls.

What is form validation? Go to any popular site with a registration form, and you will notice that they provide feedback when you don't enter your data in the format they are expecting.

* + "This field is required" (You can't leave this field blank).
  + "Please enter your phone number in the format xxx-xxxx" (A specific data format is required for it to be considered valid).
  + "Please enter a valid email address" (the data you entered is not in the right format).
  + "Your password needs to be between 8 and 30 characters long and contain one uppercase letter, one symbol, and a number." (A very specific data format is required for your data).

Using Built in form validation:

* [required](https://developer.mozilla.org/en-US/docs/Web/HTML/Attributes/required): Specifies whether a form field needs to be filled in before the form can be submitted.
* [minlength](https://developer.mozilla.org/en-US/docs/Web/HTML/Attributes/minlength) and [maxlength](https://developer.mozilla.org/en-US/docs/Web/HTML/Attributes/maxlength): Specifies the minimum and maximum length of textual data (strings)
* [min](https://developer.mozilla.org/en-US/docs/Web/HTML/Attributes/min) and [max](https://developer.mozilla.org/en-US/docs/Web/HTML/Attributes/max): Specifies the minimum and maximum values of numerical input types
* type: Specifies whether the data needs to be a number, an email address, or some other specific preset type.
* [pattern](https://developer.mozilla.org/en-US/docs/Web/HTML/Attributes/pattern): Specifies a [regular expression](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions) that defines a pattern the entered data needs to follow.

Simple Required Built in form validation

<form>

<label for="choose">Would you prefer a banana or cherry? (required)</label>

<input id="choose" name="i\_like" required>

<button>Submit</button>

</form>

CSS:

input:invalid {

border: 2px dashed red;

}

input:invalid:required {

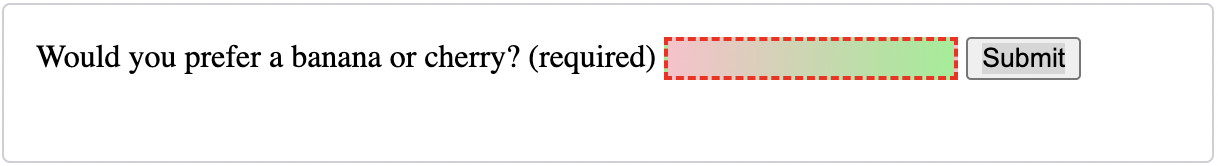
background-image: linear-gradient(to right, pink, lightgreen);

}

input:valid {

border: 2px solid black;

}



A complete built in form

<form>

<p>

<fieldset>

<legend>Do you have a driver's license?<abbr title="This field is mandatory" aria-label="required">\*</abbr></legend>

<!-- While only one radio button in a same-named group can be selected at a time,

and therefore only one radio button in a same-named group having the "required"

attribute suffices in making a selection a requirement -->

<input type="radio" required name="driver" id="r1" value="yes"><label for="r1">Yes</label>

<input type="radio" required name="driver" id="r2" value="no"><label for="r2">No</label>

</fieldset>

</p>

<p>

<label for="n1">How old are you?</label>

<!-- The pattern attribute can act as a fallback for browsers which

don't implement the number input type but support the pattern attribute.

Please note that browsers that support the pattern attribute will make it

fail silently when used with a number field.

Its usage here acts only as a fallback -->

<input type="number" min="12" max="120" step="1" id="n1" name="age"

pattern="\d+">

</p>

<p>

<label for="t1">What's your favorite fruit?<abbr title="This field is mandatory" aria-label="required">\*</abbr></label>

<input type="text" id="t1" name="fruit" list="l1" required

pattern="[Bb]anana|[Cc]herry|[Aa]pple|[Ss]trawberry|[Ll]emon|[Oo]range">

<datalist id="l1">

<option>Banana</option>

<option>Cherry</option>

<option>Apple</option>

<option>Strawberry</option>

<option>Lemon</option>

<option>Orange</option>

</datalist>

</p>

<p>

<label for="t2">What's your e-mail address?</label>

<input type="email" id="t2" name="email">

</p>

<p>

<label for="t3">Leave a short message</label>

<textarea id="t3" name="msg" maxlength="140" rows="5"></textarea>

</p>

<p>

<button>Submit</button>

</p>

</form>

CSS

form {

font: 1em sans-serif;

max-width: 320px;

}

p > label {

display: block;

}

input[type="text"],

input[type="email"],

input[type="number"],

textarea,

fieldset {

width : 100%;

border: 1px solid #333;

box-sizing: border-box;

}

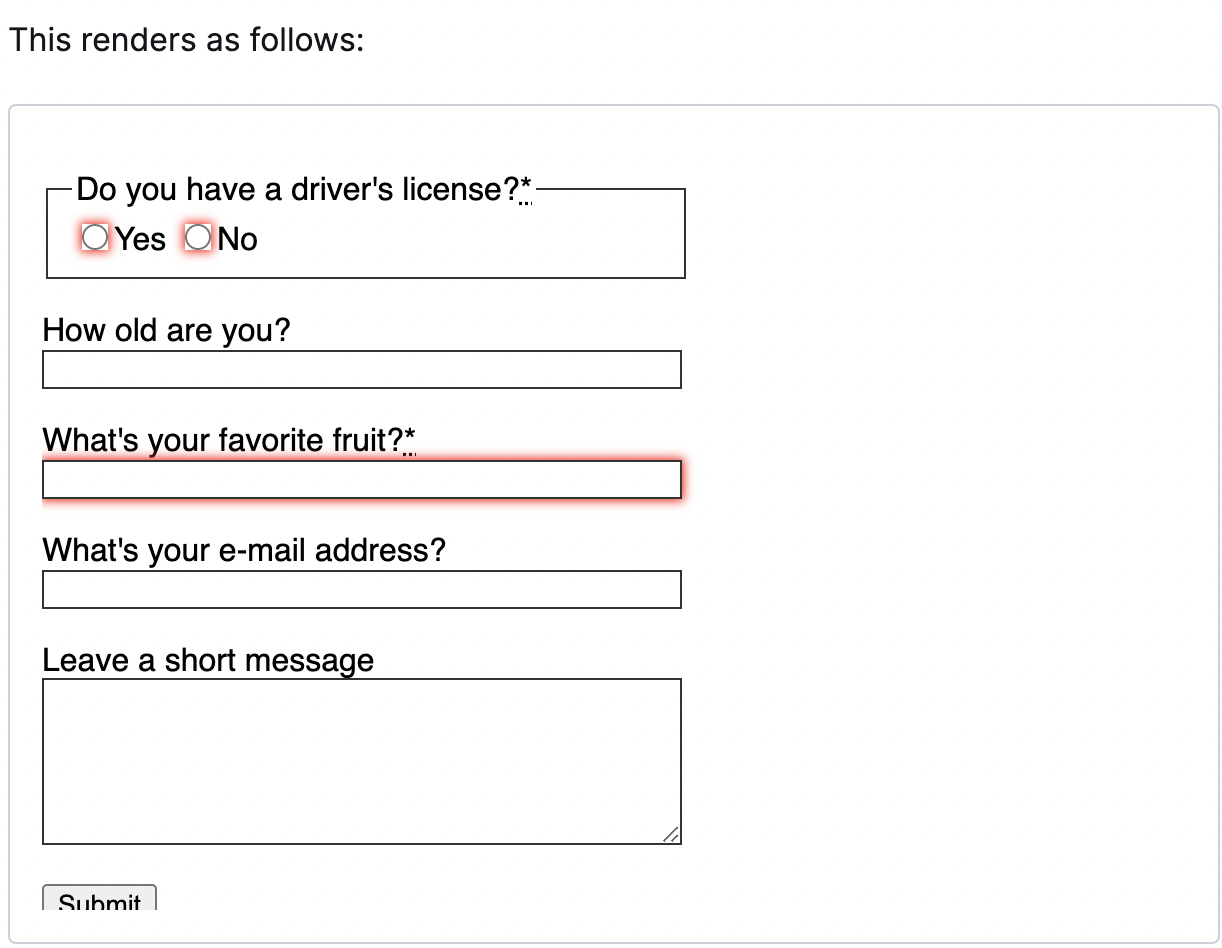
input:invalid {

box-shadow: 0 0 5px 1px red;

}

input:focus:invalid {

box-shadow: none;

}

**Using the Fetch API**

The [Fetch API](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API) provides a JavaScript interface for accessing and manipulating parts of the HTTP pipeline, such as requests and responses. It also provides a global [fetch()](https://developer.mozilla.org/en-US/docs/Web/API/fetch) method that provides an easy, logical way to fetch resources asynchronously across the network.

Basic fetch request

fetch('http://example.com/movies.json')

.then(response => response.json())

.then(data => console.log(data));

**[Uploading JSON data](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch#uploading_json_data)**

Use [fetch()](https://developer.mozilla.org/en-US/docs/Web/API/fetch) to POST JSON-encoded data.

const data = { username: 'example' };

fetch('https://example.com/profile', {

method: 'POST', // or 'PUT'

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify(data),

})

.then(response => response.json())

.then(data => {

console.log('Success:', data);

})

.catch((error) => {

console.error('Error:', error);

});

**[Uploading a file](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch#uploading_a_file)**

Files can be uploaded using an HTML <input type="file" /> input element, [FormData()](https://developer.mozilla.org/en-US/docs/Web/API/FormData/FormData) and [fetch()](https://developer.mozilla.org/en-US/docs/Web/API/fetch).

const formData = new FormData();

const fileField = document.querySelector('input[type="file"]');

formData.append('username', 'abc123');

formData.append('avatar', fileField.files[0]);

fetch('https://example.com/profile/avatar', {

method: 'PUT',

body: formData

})

.then(response => response.json())

.then(result => {

console.log('Success:', result);

})

.catch(error => {

console.error('Error:', error);

});