SUBJECT: Client-Side Form Validation DATE:

Client-Side Form Validation

- is an initial check for errors so corrections can be made before sending form to server

Built-in Form Validation

- Modern-Form controls provide validation w/o using JS.
- Validation Attributes:
  - → required: specifies a mandatory form field
  - → minlength & maxlength: sets min & max lengths of textual data
  - → min & max: sets min & max values of numerical input data

SUBJECT: Built-in Validation DATE:

- More Validation Attributes:

  → type: specifies if the data must be a number, email address, or other specific preset type

  → pattern: specifies a regular expression that defines a data pattern

- If an element is valid, these things are true:

  → the element matches the :valid CSS pseudo-class, which lets you apply a specific style to valid elements

  → the form will be submitted (unless any JS stops it)

SUBJECT: Built-in Validation DATE:

- If an element is invalid, these things are true:

  → the element matches the :invalid CSS pseudo-class... sometimes it also matches other pseudo-classes depending on the error

  → the browser will block the form from being submitted to the server

Ex: a "required" form →

```
<form>
  <label for="choose">Would you prefer a banana or cherry? (required)</label>
  <input id="choose" name="i-like" required />
  <button>Submit</button>
</form>
```

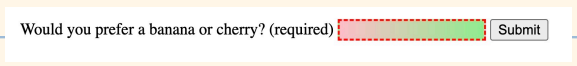
inputs w/no value matches the :invalid pseudo-class

```
input:invalid {
  border: 2px dashed red;
}

input:invalid:required {
  background-image: linear-gradient(to right, pink, lightgreen);
}

input:valid {
  border: 2px solid black;
}
```

gives invalid input a red dashed border



* remember to indicate required input fields

SUBJECT: The pattern Attribute DATE:

- The pattern attribute expects a regular expression as its value

* Note: <textarea> doesn't support the pattern attr.

The Basic Rundown on Regexp's:

- a → matches one character that is a (not b, aa, etc.)
- abc → matches a, followed by b, followed by c
- ab?c → matches a, optionally followed by a single b, followed by c (ac or abc)
- ab*c → matches a, optionally followed by any number of b's, followed by c (ac, abc, abbbc, etc.)
- a|b → matches one character that is a or b
- abc|xyz → matches exactly abc or exactly xyz (but not abcxyz or a or y, & so on)

Example that → only accepts "banana", "Banana", "cherry", or "Cherry" as input

```
<form>
  <label for="choose">Would you prefer a banana or a cherry?</label>
  <input id="choose" name="i-like" required pattern="[Bb]anana|[Cc]herry" />
  <button>Submit</button>
</form>
```

SUBJECT: Constraining Input Size DATE:

maxlength & minlength (strings)

- always try to include a character count feedback in an accessible manner so users can reduce their input until it satisfies the maxlength.

min & max (numbers)

- used on the number input type to provide a valid range of input values

```
<form>
  <div>
    <label for="choose">Would you prefer a banana or a cherry?</label>
    <input
      type="text"
      id="choose"
      name="i-like"
      required
      minlength="6" minlength & maxlength
      maxlength="6" />
  </div>
  <div>
    <label for="number">How many would you like?</label>
    <input type="number" id="number" name="amount" value="1" min="1" max="10"
/>
  </div>
  <div>
    <button>Submit</button>
  </div>
</form>
```

Full Example:

```
<form>
  <p>
    <fieldset>
      <legend>Do you have a driver's license?<span aria-
label="required">*</span></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>
```

SUBJECT: Client-Side Form Validation DATE:

Full Example: (continued)

```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;
}
```


```html
  <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?<span aria-
label="required">*</span></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 email 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>
```

SUBJECT: Validating Forms via JS DATE:

The Constraint Validation API

Form element DOM Interfaces

→ HTMLButtonElement
- represents a <button> element

→ HTMLFieldSetElement
- represents a <fieldset> element

→ HTMLInputElement
- represents an <input> element

→ HTMLOutputElement
- represents an <output> element

→ HTMLSelectElement
- represents a <select> element

→ HTMLTextAreaElement
- represents a <textarea> element

SUBJECT: Constraint API Properties DATE:

Properties available to the elements above:

→ validationMessage

- Returns localized message describing the validation constraints that the control doesn't satisfy (if any)

→ Validity

- Returns a ValidityState obj. containing properties that describe the element's validity state.

- Some Available ValidityState properties:

regexp patterns

- patternMismatch – returns true if value doesn't match regexp pattern
  – element matches :invalid if true

maxlength chars

- tooLong – true if string value > maxlength
  – element matches :invalid if true

minlength chars

- tooShort – true if string value < minlength
  – element matches :invalid if true

SUBJECT: Constraint API Properties DATE:

max values ° rangeOverflow — true if value > max
— element matches :invalid
& :out-of-range if true

min values ° rangeUnderflow — true if value < min
— element matches :invalid
& :out-of-range if true

email or url types ° typeMismatch — true if value not in the required syntax when type is an email or url
— matches :invalid if true

all validation constraints ° valid — returns true if element meets all validation constraints
— matches :valid if true & :invalid otherwise

required attributes ° valueMissing — true if element has a required attribute, but no value
— Matches :valid if true

→ willValidate

- Returns true if the element will be validated when the form is submitted

SUBJECT: Constraint API Methods DATE:

- These methods are available to the above elements & form element:

→ checkValidity()
  - returns true if element's value has no validity problems
  - if invalid, it fires an invalid event on the element

→ reportValidity()
  - reports invalid field(s) w/events
  - useful when combined w/preventDefault() in an onSubmit event handler

→ setCustomValidity(message)
  - adds a custom error message to the element

Implementing a Customized Error Msg

- Invalid forms always return automated error messages, but the way it is displayed depends on the browser

→ it is <u>better</u> to use setCustomValidity(msg)

SUBJECT: Custom Error Messages DATE:

Example ⟶ (simple)

```
<form>
  <label for="mail">
    I would like you to provide me with an email address:
  </label>
  <input type="email" id="mail" name="mail" />
  <button>Submit</button>
</form>
```

1. Stores a ref. to email input
2. Adds an eventListener to the input that runs the contained code each time the value in the input is changed.

```
const email = document.getElementById("mail");

email.addEventListener("input", (event) => {
  if (email.validity.typeMismatch) {
    email.setCustomValidity("I am expecting an email address!");
  } else {
    email.setCustomValidity("");  if typeMismatch is false,
  }                               the error msg. is empty
});
```

Example ⟶ (detailed)

1. novalidate attr. turns off browser's auto-validation... it doesn't disable support for constraint validation API or CSS pseudo-classes

```
<form novalidate>
  <p>
    <label for="mail">
      <span>Please enter an email address:</span>
      <input type="email" id="mail" name="mail" required minlength="8" />
      <span class="error" aria-live="polite"></span>
    </label>
  </p>
  <button>Submit</button>
</form>
```

```
body {
  font: 1em sans-serif;
  width: 200px;
  padding: 0;
  margin: 0 auto;
}

p * {
  display: block;
}
```

```
input[type="email"] {
  appearance: none;

  width: 100%;
  border: 1px solid #333;
  margin: 0;

  font-family: inherit;
  font-size: 90%;

  box-sizing: border-box;
}
```

SUBJECT: Custom Error Messages DATE:

Example ⟶

(detailed)

```css
/* This is our style for the invalid fields */
input:invalid {
  border-color: #900;
  background-color: #fdd;
}

input:focus:invalid {
  outline: none;
}

/* This is the style of our error messages */
.error {
  width: 100%;
  padding: 0;

  font-size: 80%;
  color: white;
  background-color: #900;
  border-radius: 0 0 5px 5px;

  box-sizing: border-box;
}

.error.active {
  padding: 0.3em;
}
```

```js
// There are many ways to pick a DOM node; here we get the form itself and the email
// input box, as well as the span element into which we will place the error message.
const form = document.querySelector("form");
const email = document.getElementById("mail");
const emailError = document.querySelector("#mail + span.error");

email.addEventListener("input", (event) => {
  // Each time the user types something, we check if the
  // form fields are valid.

  if (email.validity.valid) {
    // In case there is an error message visible, if the field
    // is valid, we remove the error message.
    emailError.textContent = ""; // Reset the content of the message
    emailError.className = "error"; // Reset the visual state of the message
  } else {
    // If there is still an error, show the correct error
    showError();
  }
});

form.addEventListener("submit", (event) => {
  // if the email field is valid, we let the form submit
  if (!email.validity.valid) {
    // If it isn't, we display an appropriate error message
    showError();
    // Then we prevent the form from being sent by canceling the event
    event.preventDefault();
  }
});
function showError() {
  if (email.validity.valueMissing) {
    // If the field is empty,
    // display the following error message.
    emailError.textContent = "You need to enter an email address.";
  } else if (email.validity.typeMismatch) {
    // If the field doesn't contain an email address,
    // display the following error message.
    emailError.textContent = "Entered value needs to be an email address.";
  } else if (email.validity.tooShort) {
    // If the data is too short,
    // display the following error message.
    emailError.textContent = `Email should be at least ${email.minLength} characters; you entered ${email.value.length}.`;
  }

  // Set the styling appropriately
  emailError.className = "error active";
}
```

SUBJECT: Validating Forms W/O Constraint API DATE:

To validate a form, ask the following questions:

1. What kind of validation should I perform?

→ must decide how to validate your data
- string operations
- type conversion
- regexp ... etc

2. What should I do if the form doesn't validate?

→ must decide how the form should behave
- does it send the data anyway?
- should the error fields be highlighted?
- should an error msg. be displayed?

3. How can I help the user to correct invalid data?

→ MUST provide as much useful info. as possible to help them correct their inputs
- offer suggestions
- give clear error messages