Week 3, Lesson 6 Readings

***Chapter 8: Forms***

<form> = Element that contains form controls such as input fields, select menus, and buttons.

Document.forms = Returns an HTML collection of all the forms in the document in the order they appear in the markup.

Advanced way of using document.forms: *const* form = document.forms.search;

Medium way of using document.forms: *const* form = document.forms[0];

Beginners way of using document.forms: *const* form = document.getElementsByTagname('form')[0];

‘text’ = used for entering a short piece of text, such as username.

‘value’ = is often pre-set in an HTML file using the ‘value’ attribute.

input type='password' is used to enter passwords or secret information. This works in the same way as an input field with type='text', except the characters are concealed as they are entered so they’re unable to be read on the screen.

Form validation is the process of checking whether a user has entered the information into a form correctly. This includes “a required field is completed”, “an email address is valid”, “a number is entered when numerical data is required”, and “a password is at least a minimum number of characters.”

***Using GormData Objects Effectively:***

Must have a “name” attribute within your “form” section in HTML for the name to work within a form.

FormData allows you to send and receive data.

***NDN: Client-Side Form Validation:***

Client-side validation is an initial check and an important feature of good user experience; by catching invalid data on the client-side, the user can fix it straight away. If it gets to the server and is then rejected, a noticeable delay is caused by a round trip to the server and then back to the client-side to tell the user to fix their data.

* "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).

Validation done in the browser is called “client-side” validation.

Validation done on the server is called “server-side” validation.

Developers want to make filling out web forms as easy as possible. But with this, they need to validate them because:

* **We want to get the right data, in the right format.** Our applications won't work properly if our users' data is stored in the wrong format, is incorrect, or is omitted altogether.
* **We want to protect our users' data**. Forcing our users to enter secure passwords makes it easier to protect their account information.
* **We want to protect ourselves**. There are many ways that malicious users can misuse unprotected forms to damage the application.

Types of client-side validation:

* **Built-in form validation** uses HTML form validation features, which we've discussed in many places throughout this module. This validation generally doesn't require much JavaScript. Built-in form validation has better performance than JavaScript, but it is not as customizable as JavaScript validation.
* **JavaScript** validation is coded using JavaScript. This validation is completely customizable, but you need to create it all (or use a library).

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.

When an element is valid, the following are true:

* The element matches the [:valid](https://developer.mozilla.org/en-US/docs/Web/CSS/:valid) CSS pseudo-class, which lets you apply a specific style to valid elements.
* If the user tries to send the data, the browser will submit the form, provided there is nothing else stopping it from doing so (e.g., JavaScript).

When an element is invalid, the following are true:

* The element matches the [:invalid](https://developer.mozilla.org/en-US/docs/Web/CSS/:invalid) CSS pseudo-class, and sometimes other UI pseudo-classes (e.g., [:out-of-range](https://developer.mozilla.org/en-US/docs/Web/CSS/:out-of-range)) depending on the error, which lets you apply a specific style to invalid elements.
* If the user tries to send the data, the browser will block the form and display an error message.

Another useful validation feature is the “pattern” attribute. This known as the “Regular Expression” or “regex.”

Regexps are complex, examples include:

* a — Matches one character that is a (not b, not aa, and so on).
* 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 bs, followed by c. (ac, abc, abbbbbc, and so on).
* 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, and so on).

JavaScript is great if you want to take control over the look and feel of a native error message.

Constraint Validation API uses a set of methods and properties available on the following form element DOM interfaces:

* [HTMLButtonElement](https://developer.mozilla.org/en-US/docs/Web/API/HTMLButtonElement) (represents a [<button>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/button) element)
* [HTMLFieldSetElement](https://developer.mozilla.org/en-US/docs/Web/API/HTMLFieldSetElement) (represents a [<fieldset>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/fieldset) element)
* [HTMLInputElement](https://developer.mozilla.org/en-US/docs/Web/API/HTMLInputElement) (represents an [<input>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input) element)
* [HTMLOutputElement](https://developer.mozilla.org/en-US/docs/Web/API/HTMLOutputElement) (represents an [<output>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/output) element)
* [HTMLSelectElement](https://developer.mozilla.org/en-US/docs/Web/API/HTMLSelectElement) (represents a [<select>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/select) element)
* [HTMLTextAreaElement](https://developer.mozilla.org/en-US/docs/Web/API/HTMLTextAreaElement) (represents a [<textarea>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/textarea) element)

Constraint validation API makes the following properties available on the above elements:

* validationMessage: Returns a localized message describing the validation constraints that the control doesn't satisfy (if any). If the control is not a candidate for constraint validation (willValidate is false) or the element's value satisfies its constraints (is valid), this will return an empty string.
* validity: Returns a ValidityState object that contains several properties describing the validity state of the element. You can find full details of all the available properties in the [ValidityState](https://developer.mozilla.org/en-US/docs/Web/API/ValidityState) reference page; below is listed a few of the more common ones:
  + [patternMismatch](https://developer.mozilla.org/en-US/docs/Web/API/ValidityState/patternMismatch): Returns true if the value does not match the specified [pattern](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input#attr-pattern), and false if it does match. If true, the element matches the [:invalid](https://developer.mozilla.org/en-US/docs/Web/CSS/:invalid) CSS pseudo-class.
  + [tooLong](https://developer.mozilla.org/en-US/docs/Web/API/ValidityState/tooLong): Returns true if the value is longer than the maximum length specified by the [maxlength](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input" \l "attr-maxlength) attribute, or false if it is shorter than or equal to the maximum. If true, the element matches the [:invalid](https://developer.mozilla.org/en-US/docs/Web/CSS/:invalid) CSS pseudo-class.
  + [tooShort](https://developer.mozilla.org/en-US/docs/Web/API/ValidityState/tooShort): Returns true if the value is shorter than the minimum length specified by the [minlength](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input" \l "attr-minlength) attribute, or false if it is greater than or equal to the minimum. If true, the element matches the [:invalid](https://developer.mozilla.org/en-US/docs/Web/CSS/:invalid) CSS pseudo-class.
  + [rangeOverflow](https://developer.mozilla.org/en-US/docs/Web/API/ValidityState/rangeOverflow): Returns true if the value is greater than the maximum specified by the [max](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input#attr-max) attribute, or false if it is less than or equal to the maximum. If true, the element matches the [:invalid](https://developer.mozilla.org/en-US/docs/Web/CSS/:invalid) and [:out-of-range](https://developer.mozilla.org/en-US/docs/Web/CSS/:out-of-range) CSS pseudo-classes.
  + [rangeUnderflow](https://developer.mozilla.org/en-US/docs/Web/API/ValidityState/rangeUnderflow): Returns true if the value is less than the minimum specified by the [min](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input#attr-min) attribute, or false if it is greater than or equal to the minimum. If true, the element matches the [:invalid](https://developer.mozilla.org/en-US/docs/Web/CSS/:invalid) and [:out-of-range](https://developer.mozilla.org/en-US/docs/Web/CSS/:out-of-range) CSS pseudo-classes.
  + [typeMismatch](https://developer.mozilla.org/en-US/docs/Web/API/ValidityState/typeMismatch): Returns true if the value is not in the required syntax (when [type](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input#attr-type) is email or url), or false if the syntax is correct. If true, the element matches the [:invalid](https://developer.mozilla.org/en-US/docs/Web/CSS/:invalid) CSS pseudo-class.
  + valid: Returns true if the element meets all its validation constraints, and is therefore considered to be valid, or false if it fails any constraint. If true, the element matches the [:valid](https://developer.mozilla.org/en-US/docs/Web/CSS/:valid) CSS pseudo-class; the [:invalid](https://developer.mozilla.org/en-US/docs/Web/CSS/:invalid) CSS pseudo-class otherwise.
  + valueMissing: Returns true if the element has a [required](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input#attr-required) attribute, but no value, or false otherwise. If true, the element matches the [:invalid](https://developer.mozilla.org/en-US/docs/Web/CSS/:invalid) CSS pseudo-class.
* willValidate: Returns true if the element will be validated when the form is submitted; false otherwise.

Constraint Validation API also makes the following methods available on the above elements and the “form” element:

* checkValidity(): Returns true if the element's value has no validity problems; false otherwise. If the element is invalid, this method also fires an [invalid event](https://developer.mozilla.org/en-US/docs/Web/API/HTMLInputElement/invalid_event) on the element.
* reportValidity(): Reports invalid field(s) using events. Useful in combination with preventDefault() in an onSubmit event handler
* setCustomValidity(message): Adds a custom error message to the element; if you set a custom error message, the element is considered to be invalid, and the specified error is displayed. This lets you use JavaScript code to establish a validation failure other than those offered by the standard HTML validation constraints. The message is shown to the user when reporting the problem.