**WEEK 10 READINGS**

**Validating Forms**

Simply put, form validation is the process of ensuring that all required input fields in an HTML form are filled out in the correct format. Most computer users are exposed to client-side form validation. This validation can be controlled using CSS, HTML and JavaScript.

Form validation is important because it helps ensure the security and protection of important data. There are two types of client-side validation, which are: Built-in form validation and JavaScript validation.

Built-in form validation involves the use of special validation attributes in HTML form elements. These include:

* Required: specifies whether a form element must be filled before submission is possible
* Minlength and Maxlength: specifies the minimum or maximum number of characters for text input
* Min and Max: specifies the minimum and maximum value of numerical inputs
* Pattern: specifies a “Regular Expression” that defines a pattern the entered input must follow.

Coupled with the CSS pseudo-classes, “:valid and :invalid”, HTML forms can be validated before submitting.

JavaScript validation involves using the Constraint Validation API, which consists of a set of methods and properties available on the following 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)

The 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 reference page; below is listed a few of the more common ones:

patternMismatch: Returns true if the value does not match the specified pattern, and false if it does match. If true, the element matches the :invalid CSS pseudo-class.

tooLong: Returns true if the value is longer than the maximum length specified by the maxlength attribute, or false if it is shorter than or equal to the maximum. If true, the element matches the :invalid CSS pseudo-class.

tooShort: Returns true if the value is shorter than the minimum length specified by the minlength attribute, or false if it is greater than or equal to the minimum. If true, the element matches the :invalid CSS pseudo-class.

rangeOverflow: Returns true if the value is greater than the maximum specified by the max attribute, or false if it is less than or equal to the maximum. If true, the element matches the :invalid and :out-of-range CSS pseudo-classes.

rangeUnderflow: Returns true if the value is less than the minimum specified by the min attribute, or false if it is greater than or equal to the minimum. If true, the element matches the :invalid and :out-of-range CSS pseudo-classes.

typeMismatch: Returns true if the value is not in the required syntax (when type is email or url), or false if the syntax is correct. If true, the element matches the :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 CSS pseudo-class; the :invalid CSS pseudo-class otherwise.

valueMissing: Returns true if the element has a required attribute, but no value, or false otherwise. If true, the element matches the :invalid CSS pseudo-class.

willValidate: Returns true if the element will be validated when the form is submitted; false otherwise.

The 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 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 HTML5 validation constraints. The message is shown to the user when reporting the problem

NB: This info was outsourced at: <https://developer.mozilla.org/en-US/docs/Learn/Forms/Form_validation>