Chapter 8 : Forms

Forms are made up of a <form> element that contains form controls such as input fields, select menus & buttons
Each form control has an initial value that can be specified in the HTML code
this value can be changed by:
- a user entering information or interacting w/ the forms interface
- dynamically using Javascript

A Searching Example

Start with:

```
"<!doctype html>
<html lang='en'>
<head>
<meta charset='utf-8'>
<title>Search</title>
</head>
<body>
<form name='search' action='/search'>
  <input name='searchInput'>
  <button type='submit'>Search</button>
</form>
<script src='main.js'></script>
</body>
</html>"
```

Accessing Form Elements

- the legacy DOM had a useful property called document.forms that returns an HTML collection of all the forms in the order they appear in the markup
- a form object has a method called elements that returns an HTML collection of all elements in the form
  - in this case, the form contains two controls:
    - an input element
    - a button element
- can also access the form controls using their 'name' attribute
  - square bracket notation can also be used instead

Form Properties and Methods

- form.submit() will submit a form automatically
- form.reset() method will reset all the form controls back to initial values specified in the HTML
- a button w/ a type attribute of reset can also be used to do this

Reset Buttons

- generally considered poor for usability
- form.action property can be used to set the action attribute of a form

Form Events

- forms trigger a number of events
- focus event occurs when an element is focused on
  - this is when the cursor is placed inside the element
- blur event occurs when the user moves the focus away from the form element
- change event occurs when the user moves the focus away from the form element after changing it

Submitting A Form

- most important form event is the submit event
- this will send the content of the form to the server to be processed
- stop the form from being submitted to that URL by using the preventDefault() method

Retrieving & Changing Values From a Form

- text input element objects have a value property that can be used to retrieve the text inside the field
- also possible to set the value using JS
  - input.value = 'Search Here'

problem with this ↑ is that text remains in the field and users have to delete before typing
↳ remedied using focus & blur handlers

all can be avoided using placeholder attribute

Form Controls

"Some common types of form control are:

- <input> fields, including text, passwords, check boxes, radio buttons, and file uploads
- <select> menus for drop-down lists of options
- <textarea> elements for longer text entry
- <button> elements for submitting and resetting forms"

New Attributes in HTML5

- input element includes some of the new attributes in HTML5
- autofocus attribute gives focus to this element when a page loads
- maxlength attribute limits the # of characters that can be entered in the field to the value given

“start off by assigning the form to a variable and then adding an event listener for when the form is submitted:

```
const form = document.forms['hero'];
form.addEventListener('submit', makeHero, false);
```

The event listener will invoke the makeHero() function when the form is submitted. This function will return an object based on the information provided in the form”

Use the JSON.stringify() method to convert the object into a JSON string & display it in an alert box

Input Fields

- most common type of form control

Text Input Fields

- default type of input field is text
- type='text' attribute is not required
  - but it is advisable as it makes the intended purpose of the field explicit
  - ↑ helps with maintenance, readability, & future-proofing

Password Input Fields

- input type='password' is used to enter passwords or secret info

Checkbox Input Fields

- created using input fields w/ type='checkbox'
- used to select different options that can be checked (true) or left unchecked (false)
- the user can select more than one checkbox from a list

“Notice that all the checkbox elements have the same 'name' property of 'powers'. This means they can be accessed as an HTML collection, like so:

```
form.powers;
```

We can then iterate over this collection using a for loop to see if each checkbox was checked. Checkbox objects have a checked property that tells us if it has been checked or not.”

“This uses the spread operator to turn the node list into an array. This then allows us to use the filter() method that returns an array containing only the check boxes that were checked”

- Checkboxes can also be checked initially using the 'checked' attribute:
  `<input type='checkbox' value='Flight' name='powers' checked>`

Radio Button Input Fields

- created using input fields w/ type='radio'
- allows users to check an option as true but they provide an exclusive choice of options, so only one option can be selected

```
<p>What type of hero are you?</p>
<label for='hero'>Hero:
  <input type='radio' name='category' value='Hero' id='hero'>
</label>
<label for='villain'>Villain:
  <input type='radio' name='category' value='Villain' id='villain'>
</label>
<label for='anti-hero'>Anti-Hero:
  <input type='radio' name='category' value='Antihero' id='anti-hero'>
</label>
```

All these radio buttons have the same 'name' attribute of 'category'. This is used to group them together — only one radio button can be checked in a group that has the same name attribute. It also means we can access an HTML collection of all the radio buttons in that group using the property of the same name — as can be seen in this line of code:

```
form.category;
```

- the value of the selected radio button is stored in form.category.value
- each radio button has a 'checked' property that returns the boolean values true & false depending on selected or not
- can also be checked initially using the 'checked' attribute
  <input type='radio' name='type' value='Villain' checked>

Hidden Input Fields

- can be created using input fields with type='hidden'
- not displayed by the browser, but have a 'value' attribute that can contain info that is submitted w/ the form
- often used to send info such as settings or info that the user has already provided
- info in those fields is in no way secret, as it is visible in the HTML, so shouldn't be used for sensitive data

File Input Fields

- created using input fields w/ type='file'
- used to upload files

Other Input Types

- lots of new input types included in HTML5, such as number, tel, & color
- number input fields also have optional 'min' & 'max' attributes that can be used to limit the input given

Select Drop-Down List

- can be used to select one or more options from list of values
- 'multiple' attribute is required if more than 1 option is to be selected
- 'selected' attribute can be used to set the initial value in the HTML

- if only one item is selected, this will return a reference to that selection otherwise a collection will be returned containing each selection
- each selection object has a value property that is equal to the 'value' attribute of the <option> tag that was selected
- also possible to find out the index of the option that has been selected using the selectedIndex property

Text Areas

- <textarea> element is used to enter long pieces of text over multiple lines
- we access them using the 'name' attribute
- use the value property to see what text was entered

Buttons

- default type is 'submit'
- 'reset', which will reset all the form fields to their initial settings
  <button type='reset'>Reset</button>
- Remember this is not recommended good practice for usability reasons
- other type is simply 'button'. This doesn't need to be inside a form element & has no default behavior
- creates a clickable button that can have an event listener attached to it
- type of 'menu' that can be combined with <menu>, <menuitem>, & tags to create a dropdown menu when its clicked on

Form Validation

"types of validation that occur include ensuring that:

- A required field is completed
- An email address is valid
- A number is entered when numerical data is required
- A password is at least a minimum number of characters"

Validation can occur on the client side using JavaScript, and on the server side. It is advisable to use both client-side and server-side validation. JavaScript should not be relied upon to validate any data before it's saved to a database. This is because it's possible for a user to modify the JavaScript code and bypass the validation rules. It's also very easy to bypass the front-end completely and send arbitrary data to the application's backend. For these reasons, JavaScript validation should be used to enhance the user experience when filling in a form by giving feedback about any errors before it's submitted. This should then be backed up with more validation performed on the server before the data is eventually saved to a database

- HTML5 has its own validation API that can be used, although it lacks the full support from all browsers @ the moment

The API works by simply adding relevant attributes to the form fields. For example, if a field is a required field that must be filled in, all you need to do is add a 'required' attribute to that field and the browser will take care of the rest.

- also possible to implement custom form validation using JS.

"improve the usability of the form further by giving instant feedback, instead of waiting for the form to be submitted. This can be achieved by adding the event listener directly to the input field that will fire when the user presses a key (using the keyup event)"

validateInline() function is called every time the event is triggered

This is a Specific & Perhaps Unrealistic Example

- in a real application, you might end up having to validate many different elements according to various different rules

Disabling The Submit Button

Another useful technique that can aid usability is to disable the submit button if there are errors on the form
submit button can be disable by added the disabled attribute to the <input> element:

```
<button type='submit' id='submit' disabled>Submit</button>
```

“This can be changed programmatically using the disabled property of the <button> element. The following function will disable the button if an input field is empty:

```
function disableSubmit(event) {
   if(event.target.value === ''){
   document.getElementById('submit').disabled = true;
   } else {
   document.getElementById('submit').disabled = false;
   }
}”
```