WEB230: JavaScript 1

Module 7: Forms

Forms

- Originally designed for the pre-JavaScript Web
    - Allow web sites to send user-submitted information to a server
    - Assumes that interaction with the server always navigates to a new page
    - That has changed with modern JS but we won't be covering that

DOM

- Form elements are part of the DOM
    - A number of properties and events that are not present on other elements
    - Make it possible to inspect and control form fields with JavaScript
    - Add new functionality to a form or use forms as building blocks in a JavaScript application

Form fields

- A web form consists of any number of input fields grouped in a `<form>` tag.
- HTML allows several different styles of fields:
    - simple on/off checkboxes
    - text input fields
    - drop-down menus
    - etc.

`<input>` Fields

- Most form fields use the `<input>` tag
- The `type` attribute selects the field's style
- Commonly used `<input>` types:
    - `text` A single-line text field
    - `password` Same as text but hides the text that is typed
    - `checkbox` An on/off switch
    - `radio` (Part of) a multiple-choice field
    - `file` Allows the user to choose a file from their computer

Form-less fields

- Fields do not have to appear in a `<form>` tag
- Form-less fields cannot be submitted (only a form can)
- Can use them with JavaScript
- JavaScript interface for such elements differs with the type of the element

Example

```
<p><input type="text" value="abc" /> (text)</p>
<p><input type="password" value="abc" /> (password)</p>
<p><input type="checkbox" checked /> (checkbox)</p>
<p>
  <input type="radio" value="A" name="choice" />
  <input type="radio" value="B" name="choice" checked />
  <input type="radio" value="C" name="choice" /> (radio)
</p>
<p><input type="file" /> (file)</p>
```

`<textarea>` Field

- Multiline text field
- Requires a matching `</textarea>` closing tag
- uses the text content, instead of the `value` attribute, as starting text

```
<textarea>
one
two
three
</textarea>
```

`<select>` Field

- Used to create a field that allows the user to select from a number of predefined options
- Whenever the value of a form field changes, it will fire a "change" event

```
<select>
  <option>Pancakes</option>
  <option>Pudding</option>
  <option>Ice cream</option>
</select>
```

Focus

- Form fields can get keyboard focus
- When clicked or activated they become the currently active element and wil get keyboard input
- You can type into a text field only when it is focused
- Other fields respond differently to keyboard events
    - `<select>` menu tries to move to the option that contains the text the user typed and responds to the arrow keys by moving its selection up and down

Giving Focus

- `.focus()` method moves focus to the DOM element it is called on
- `.blur()` method removes focus
- The value of `document.activeElement` corresponds to the currently focused element

Example

```
<input type="text" />
<script>
  document.querySelector('input').focus();
  console.log(document.activeElement.tagName);
  // → INPUT
  document.querySelector('input').blur();
  console.log(document.activeElement.tagName);
  // → BODY
</script>
```

`autofocus` Attribute

- HTML provides the `autofocus` attribute
- give that element focus when the page is opened

`tabindex` Attribute

- User can move the focus through the document by pressing the TAB key
- Can set the order in which elements receive focus with the `tabindex` attribute
- The following example document will let the focus jump from the text input to the OK button, rather than going through the help link first:

```
<input type="text" tabindex="1" /> <a href=".">(help)</a>
<button onclick="console.log('ok')" tabindex="2">OK</button>
```

- Most types of HTML elements cannot be focused
    - make it focusable by adding a `tabindex` attribute
- `tabindex="-1"` makes tabbing skip over an element

Disabled fields

- Form fields can be disabled through their `disabled` attribute
- It is a boolean attribute (can be specified without value)

```
<button>I'm all right</button> <button disabled>I'm out</button>
```

- Disabled fields cannot be focused or changed
- Browsers display them as gray and faded

The Form as a Whole

- Fields contained in a `<form>` element will have a `form` property
    - linking back to the form's DOM element
- The `<form>` element has a property called `elements`
    - contains an array-like collection of the fields inside it
- The `name` attribute of a form field determines the way its value will be identified when the form is submitted
- Also used as a property name on the form's `elements` property
    - acts both as an array (accessible by number) and an object (accessible by name)

Example

```
<form action="example/submit.html">
  Name: <input type="text" name="name" /><br />
  Password: <input type="password" name="password" /><br />
  <button type="submit">Log in</button>
</form>

<script>
  let form = document.querySelector('form');
  console.log(form.elements[1].type);
  // → password
  console.log(form.elements.password.type);
  // → password
  console.log(form.elements.name.form === form);
  // → true
</script>
```

Submit Button

- A button with `type="submit"` will cause the form to be submitted
- Pressing ENTER when a form field is focused has the same effect
- Before that happens, a "submit" event is fired
- You can handle this event with JavaScript and prevent this default behavior by calling `.preventDefault()` on the event object

Example

```
<form action="example/submit.html">
  Value: <input type="text" name="value" />
  <button type="submit">Save</button>
</form>
<script>
  let form = document.querySelector('form');
  form.addEventListener('submit', event => {
    console.log('Saving value', form.elements.value.value);
    event.preventDefault();
  });
</script>
```

Intercepting `submit` Events

Why intercept the `submit` event?

- Form validation - verify that the values make sense and immediately show an error message
- Can disable submitting the form and have our program handle the input

Text fields

- Fields created by `<input>` tags with a `type="text"`, `type="password"`, and `<textarea>` tags, share a common interface
- These DOM elements have a `value` property that holds their current content as a string
- Setting this property to another string changes the field's content

`selectionStart` and `selectionEnd`

- Provide information about the cursor and selection in the text
- When nothing is selected, these two properties hold the same number, indicating the position of the cursor
- 0 indicates the start of the text, and 10 indicates the cursor is after the 10th character
- When part of the field is selected, the two properties will differ, giving us the start and end of the selected text.
- These properties may also be written to

Example

Imagine you are writing an article about Khasekhemwy but have some trouble spelling his name. The following code wires up a `<textarea>` tag with an event handler that, when you press F2, inserts the string "Khasekhemwy" for you.

```
<textarea></textarea>
<script>
  let textarea = document.querySelector('textarea');
  textarea.addEventListener('keydown', event => {
    if (event.keyCode == 113) {
      // The key code for F2
      replaceSelection(textarea, 'Khasekhemwy');
      event.preventDefault();
    }
  });
  function replaceSelection(field, word) {
    let from = field.selectionStart,
      to = field.selectionEnd;
    field.value = field.value.slice(0, from) + word + field.value.slice(to);
    field.selectionStart = from + word.length; // Put the cursor after the word
    field.selectionEnd = from + word.length;
  }
</script>
```

Explanation of Example

- `replaceSelection`
    - replaces the currently selected part of a text field with the given word an
    - then moves the cursor after that word
- The `keydown` event fires when a key is pressed

`change` Event

- The `change` event for a text field fires when the field loses focus after its content was changed
- To respond immediately to changes in a text field, you should register a handler for the `input` event
    - fires every time the user types a character, deletes text, or otherwise changes the field's content

Counter Example

The following example shows a text field and a counter displaying the current length of the text in the field:

```
<input type="text" /> length: <span id="length">0</span>
<script>
  let text = document.querySelector('input');
  let output = document.querySelector('#length');
  text.addEventListener('input', () => {
    output.textContent = text.value.length;
  });
</script>
```

Checkboxes and Radio Buttons

- A checkbox field is a binary toggle
- Get value from `checked` property - Boolean value

```
<label> <input type="checkbox" id="purple" /> Make this page purple </label>
<script>
  let checkbox = document.querySelector('#purple');
  checkbox.addEventListener('change', () => {
    document.body.style.background = checkbox.checked ? 'mediumpurple' : '';
  });
</script>
```

`<label>` Tag

- Associates a piece of document with an input field
- Clicking anywhere on the label will activate the field
    - text field - focuses it
    - checkbox or radio button - toggles its value

Radio Buttons

- A radio button is similar to a checkbox
- implicitly linked to other radio buttons with the same name
- only one of them can be active at any time

Example

```
Color:
<label> <input type="radio" name="color" value="orange" /> Orange </label>
<label> <input type="radio" name="color" value="lightgreen" /> Green </label>
<label> <input type="radio" name="color" value="lightblue" /> Blue </label>
<script>
  let buttons = document.querySelectorAll('[name=color]');
  for (let button of buttons) {
    button.addEventListener('change', () => {
      document.body.style.background = button.value;
    });
  }
</script>
```

`select` fields

- Conceptually similar to radio buttons
    - allow the user to choose from a set of options
- appearance of a `<select>` tag is determined by browser

`multiple` Attribute

- Select fields variant that is more like a list of checkboxes
- With `multiple` attribute, a `<select>` tag will allow the user to select any number of options

`select` field Value

- Each `<option>` tag has a value
    - This value can be defined with a `value` attribute
    - When not given, the text inside the option will count as its value
- The value property of a `<select>` element reflects the currently selected option

`option` Tag

- The `<option>` tags can be accessed as an array-like object using `options` property
- Each option has a bolean property called `selected`
    - Indicates whether that option is currently selected
    - Can also be written to select or deselect an option

Example

Hold control (or command on a Mac) to select multiple options.

```
<select multiple>
  <option value="1">0001</option>
  <option value="2">0010</option>
  <option value="4">0100</option>
  <option value="8">1000</option>
</select>
= <span id="output">0</span>
<script>
  let select = document.querySelector('select');
  let output = document.querySelector('#output');
  select.addEventListener('change', () => {
    let number = 0;
    for (let option of select.options) {
      if (option.selected) {
        number += Number(option.value);
      }
    }
    output.textContent = number;
  });
</script>
```

`file` Field

- `file` field was designed to upload files from the user
- Also provides a way to read such files from JavaScript programs
- The field acts as a gatekeeper
    - It gives the browser permission to read the file
- A `file` field is a button labeled with “Choose File” or “Browse”, with information about the chosen file next to it

Example

```
<input type="file" />
<script>
  let input = document.querySelector('input');
  input.addEventListener('change', () => {
    if (input.files.length > 0) {
      let file = input.files[0];
      console.log('You chose', file.name);
      if (file.type) console.log('It has type', file.type);
    }
  });
</script>
```

`file` field Properties

- `.files` - an array-like object containing the files chosen in the field

    - It is initially empty
    - Also support a `multiple` attribute, which makes it possible to select multiple files
- Objects in `files` have properties such as `name`, `size`, and `type`
- Does not have is a property that contains the content of the file
    - Getting at that is a little more involved

Storing Data Client-Side

- `localStorage` object is used to store data in a way that survives page reloads
- Allows you to store string values under names

Adding items to `localstorage`

- Add items with `localStorage.setItem(name, value)`
- `name` and `value` are strings

Reading items from `localstorage`

- Read items with `localStorage.getItem(name)`
- `name` is a strings
- Returns a string with the value

Removing items from `localstorage`

- Remains in the browser until it is overwritten
    - It can be removed with `localStorage.removeItem(name)`
    - Or if the user clears their local data

Example

```
localStorage.setItem('username', 'marijn');
console.log(localStorage.getItem('username'));
// → marijn
localStorage.removeItem('username');
```

`localstorage` Details

- Can only store strings
- Sites from different domains get different storage compartments
- A website can only read its own data
- Limit to the data stored per site
    - Prevents using too much space