Module 4

Handling DOM Events

Creating HTML Dynamically

Updating HTML Content Dynamically

Managing Web Forms with JavaScript

DOM events

- what are DOM events?
  - the `click` event.
  - the `onchange` event.
  - the `onkeyup` event.
  - the `mouseover` and `mouseout` events
  - the `scroll` event
- the addEventListener function.
  - adding event handlers in a loop

HTML Events

HTML events are "things" that happen to HTML elements. When JavaScript is used in HTML pages, JavaScript can "react" on these events.

An HTML event can be something the browser does, or something a user does.

Here are some examples of HTML events:

- **An HTML web page has finished loading**
- **An HTML input field was changed**
- **An HTML button was clicked**

Often, when events happen, you may want to do something, and JavaScript lets you execute code when events are detected.

HTML Events

HTML allows event handler attributes, with JavaScript code, to be added to HTML elements.

With single quotes:

```
<element event='some JavaScript'>
```

With double quotes:

```
<element event="some JavaScript">
```

HTML Events

In the following example, an onclick attribute (with code), is added to a `<button>` element, and when clicked the JavaScript code changes the content of the element with id="demo":

```
<button onclick="document.getElementById('demo').innerHTML = Date()">
    The time is?
</button>
<p id="demo"></p>
```

In the next example, the code changes the content of its own element (using this.innerHTML):

```
<button onclick="this.innerHTML = Date()">
    The time is?
</button>
```

Codepen example

HTML Events

Since JavaScript code is often several lines long, it is more common to see event attributes calling functions:

HTML:

```
<button onclick="displayDate()">The time is?</button>

<p id="demo3"></p>
```

JavaScript:

```
function displayDate() {
  document.getElementById("demo3").innerHTML = Date();
}
```

Codepen example, same as above

Assign Events Using the HTML DOM

The HTML DOM allows you to assign events to HTML elements using JavaScript:

```
<button id="myBtn">Try it</button>
<p id="demo4"> </p>
```

```
document.getElementById("myBtn").onclick = displayDate;

function displayDate() {
  document.getElementById("demo4").innerHTML = Date();
}
```

In the example above, a `function` named `displayDate` is assigned to an HTML element with the `id="myBtn"` .

The function will be executed when the button is clicked.

Codepen example, same as above

Common HTML Events

Event	Description
`onchange`	An HTML element has been changed
`onclick`	The user clicks an HTML element
`onmouseover`	The user moves the mouse over an HTML element
`onmouseout`	The user moves the mouse away from an HTML element
`onkeydown`	The user pushes a keyboard key
`onload`	The browser has finished loading the page
`scroll`	An scrollbar is being scrolled

Source: HTML DOM Events

What can JavaScript Do?

Event handlers can be used to handle and verify user input, user actions, and browser actions:

- Things that should be done every time a page loads
- Things that should be done when the page is closed
- Action that should be performed when a user clicks a button
- Content that should be verified when a user inputs data
- And more ...

What can JavaScript Do?

Many different methods can be used to let JavaScript work with events:

- HTML event attributes can execute JavaScript code directly
- HTML event attributes can call JavaScript functions
- You can assign your own event handler functions to HTML elements
- You can prevent events from being sent or being handled
- And more ...

`onclick` Event

The onclick event occurs when the user clicks on an element.

In HTML:

```
<element onclick="myScript">
```

In JavaScript:

```
object.onclick = function(){myScript};
```

In JavaScript, using the addEventListener() method:

```
object.addEventListener("click", myScript);
```

You may click any element.

Another example on how to change the color of a `<p>` element by clicking on it:

```
<p id="demo" onclick="myFunction()">
    Click me to change my text color.
</p>
```

```
function myFunction() {
  document.getElementById("demo").style.color = "red";
}
```

In this example, the content of the `<h1>` element is changed when a user clicks on it:

```
<h1 onclick="this.innerHTML = 'Ooops!'">Click on this text!</h1>
```

Codepen example

The `onchange` Event

The `onchange` event is often used in combination with validation of input fields.

Below is an example of how to use the `onchange`. The `upperCase()` function will be called when a user changes the content of an input field.

```
<input type="text" id="fname" onchange="upperCase()">
```

```
function upperCase() {
  var x = document.getElementById("fname");
  x.value = x.value.toUpperCase();
}
```

Codepen example

The `onmouseover` and `onmouseout` Events

The `onmouseover` and `onmouseout` events can be used to trigger a function when the user mouses over, or out of, an HTML element:

```
<div onmouseover="mouseOver(this)" onmouseout="mouseOut(this)">
    Mouse Over Me
</div>
```

```
function mouseOver(obj) {
  obj.innerHTML = "Thank You";
  obj.style.backgroundColor = "lightgreen";
}

function mouseOut(obj) {
  obj.innerHTML = "Mouse Over Me";
  obj.style.backgroundColor = "#D94A38";
}
```

Codepen example, same as above

There are 3 ways to register event handlers for a DOM element.

Registering event listeners 1:3

HTML attribute

```
<button onclick="alert('Hello world!')">
```

Warning: This method should be avoided! It inflates the markup, and makes it less readable. Concerns of content/structure and behavior are not well-separated, making a bug harder to find.

Registering event listeners 2:3

DOM element properties

```
// Assuming myButton is a button element
myButton.onclick = function(event){alert('Hello world')}
```

The function can be defined to take an event parameter. The return value is treated in a special way, described in the HTML specification.

The problem with this method is that only one handler can be set per element and per event.

Registering event listeners 3:3

`EventTarget.addEventListener`

```
// Assuming myButton is a button element
myButton.addEventListener('click', greet, false);

function greet(event){
    // print and have a look at the event object
    // always print arguments in case of overlooking
    // any other arguments
    console.log('greet:', arguments);
    alert('hello world');
}
```

This is the method you should use in modern web pages.

The addEventListener() method

Add an event listener that fires when a user clicks a button:

HTML:

```
<button id="myBtn">Try it</button>
<p id="demo"></p>
```

JavaScript:

```
const btn = document.getElementById("myBtn");

btn.addEventListener("click", () => {
  document.getElementById("demo").innerHTML = Date();
});
```

Codepen example

The addEventListener() method

The `addEventListener()` method attaches an event handler to the specified element.

The `addEventListener()` method attaches an event handler to an element without overwriting existing event handlers.

You can add many event handlers to one element.

You can add many event handlers of the same type to one element, i.e two "click" events.

You can add event listeners to any DOM object not only HTML elements. i.e the `window` object.

The addEventListener() method

The `addEventListener()` method makes it easier to control how the event reacts to **bubbling**.

When using the `addEventListener()` method, the JavaScript is separated from the HTML markup, for better readability and allows you to add event listeners even when you do not control the HTML markup.

You can easily remove an event listener by using the `removeEventListener()` method.

The addEventListener() method

Syntax:

```
element.addEventListener(event, function, useCapture);
```

The first parameter is the type of the event (like `"click"` or `"mousedown"` or any other HTML DOM Event.)

The second parameter is the `function` we want to call when the event occurs.

The third parameter is a `boolean` value specifying whether to use event bubbling or event capturing. *This parameter is optional*.

> Note that you don't use the "on" prefix for the event; use "click" instead of "onclick".

Add Many Event Handlers to the Same Element

The addEventListener() method allows you to add many events to the same element, without overwriting existing events:

```
element.addEventListener("click", myFunction);
element.addEventListener("click", mySecondFunction);
```

You can add events of different types to the same element:

```
element.addEventListener("mouseover", myFunction);
element.addEventListener("click", mySecondFunction);
element.addEventListener("mouseout", myThirdFunction);
```

Codepen example, same as earlier

Add an Event Handler to the window Object

The `addEventListener()` method allows you to add event listeners on any HTML DOM object such as HTML elements, the HTML document, the `window` object, or other objects that support events, like the `xmlHttpRequest` object.

Caturing the `resize` event:

```
<p>Try resizing this browser window
to trigger the "resize" event handler.</p>
<p id="win"> </p>
```

```
const winSize = document.getElementById("win");

window.addEventListener("resize", () => {
   winSize.innerHTML = `Your window is now
   ${window.innerWidth} by ${window.innerHeight} pixels`;
});
```

Codepen example

Add an Event Handler to the window Object

Tracking mouse clicks:

```
<p>Also, click anywhere inside the browser window</p>
<p id="mouse"> </p>
```

```
const mouseData = document.getElementById("mouse");

window.addEventListener("click", (e) => {
   mouseData.innerHTML = `Your last click was at
   the coordinates [${e.pageX}, ${e.pageY}] on the page`;
});
```

Codepen example, same as above

onkeyup Event

The `onkeyup` event occurs when the user releases a key (on the keyboard).

> Tip: The order of events related to the `onkeyup` event:

1. `onkeydown`
2. `onkeypress`
3. `onkeyup`

onkeyup Event

In HTML:

```
<element onkeyup="myScript">
```

In JavaScript:

```
object.onkeyup = function(){myScript};
```

In JavaScript, using the addEventListener() method:

```
object.addEventListener("keyup", myScript);
```

onkeyup Event, an anoying example

HTML:

```
<label for="fname">Enter your name:</label>
<input type="text" id="fname">
```

JavaScript

```
const input = document.querySelector("#fname");
input.addEventListener("keyup", () => {
    input.value = input.value.toUpperCase();
});
```

Codepen example

onkeyup Event, another example

HTML:

```
<label for="name">Enter your name:</label>
<input type="text" id="name">

<p>My name is: <span id="demo"></span></p>
```

JavaScript

```
const input = document.querySelector("#name");
const output = document.querySelector("#demo");

input.addEventListener("keyup", () => {
    output.innerText = input.value;
});
```

Codepen example

Document: scroll event

The scroll event fires when the document view or an element has been scrolled.

```
document.addEventListener ('scroll', () => {});
```

> **Note**: In iOS UIWebViews, scroll events are not fired while scrolling is taking place; they are only fired after the scrolling has completed.

Codepen example

JavaScript HTML DOM Node Lists

A NodeList object is a list (collection) of nodes extracted from a document.

Most browsers return a NodeList object for the method `querySelectorAll()`.

JavaScript HTML DOM Node Lists

Note: A `NodeList` object is almost the same as an `HTMLCollection` object:

- An `HTMLCollection` is a collection of HTML elements.
- A `NodeList` is a collection of document nodes.
- A NodeList and an HTML collection is very much the same thing, in practice.

Note: A `NodeList` may look like an `Array`, but it is not:

- *You can **loop** through the node list and refer to its nodes like an Array.*
- However, you **cannot** use Array Methods, like `valueOf()`, `push()`, `pop()`, or `join()` on a node list.

JavaScript HTML DOM Node Lists

```
<p>I'm a paragraph</p>
<p>I'm also a paragraph</p>
<p>I'm definitively a paragraph</p>
<p>Why am I always last?</p>
```

The following code selects all `<p>` nodes in a document:

```
const myNodeList = document.querySelectorAll("p");

// log the second <p> node:
console.log(myNodeList[1]);
// <p>I'm also a paragraph</p>

console.log(myNodeList.length);
// logs the number of nodes: 4
```

> Note: The index starts at 0.

Codepen example

Using Node list to set Event Listener

```
<p>I'm a paragraph</p>
<p>I'm also a paragraph</p>
<p>I'm definitively a paragraph</p>
<p>Why am I always last?</p>
```

```
const myNodeList = document.querySelectorAll("p");

colors = ["red", "green", "blue", "gold"];

for (let node of myNodeList) {
  node.addEventListener('click', () => {
    let r = Math.floor((Math.random() * colors.length));
    node.style.color = colors[r];
  });
}
```

Codepen example

Todos

Mollify

Read Handling DOM Events, no Lesson Task today.

Misc.

Read Introduction to events at mdn web docs. (Note: bubbling and delegation is Level 2, for now...)