Introduction to DOM and Events

Bjarte Wang-Kileng

HVL

August 18, 2025

Western Norway University of Applied Sciences

Outline

JavaScript prerequiste

- JavaScript was introduced in DAT108.
- HTML and CSS was introduced the first semester.

JavaScript knowledge prerequisite

DAT152 requires basic knowledge of JavaScript and HTML, e.g. that of DAT108.

JavaScript actions on HTML elements

- Read, add, delete and modify HTML content.
- Also actions on style sheets.

Referencing HTML elements

Before actions on HTML elements, JavaScript must locate the elements.

Methods that locate web elements return JavaScript objects that represent the element.

HTML content

- Text content of HTML element.

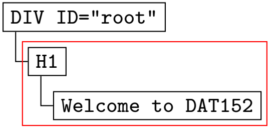
```
<!-- Text content of an H1 element -->
<div id="root">
    <h1>Welcome to DAT152</h1>
</div>
```

- HTML element attribute.

```
<!-- Attribute of a DIV element -->
<div id="root">
    <h1>Welcome to DAT152</h1>
</div>
```

- HTML tree structure.

```
<!-- Tree structure below a DIV -->
<div id="root">
    <h1>Welcome to DAT152</h1>
</div>
```



Working with HTML elements from JavaScript

- JavaScript can only work with HTML elements in browser memory:
  - Put the HTML script tag at the end of the HTML document, or
  - use the HTML attribute *defer* on the script tag, or
  - use JavaScript modules, or
  - run the code as an event handler on e.g. event **DOMContentLoaded**.
- Only the two last approaches make the code portable.

JavaScript modules

For most of DAT152, we will work with JavaScript through modules.

Loading of JavaScript modules implies *defer*.

Attribute *defer*

- To be be used on tag *script*.
- Only to be used on JS code loaded from a separate file:

```
<script src="jsfile.js" defer></script>
```

- Will allow browser to load the JS file in a separate I/O thread.
- The code will be run only after the document has finished loding.
- The order of the *script* tags determins the run sequence of the code.

Outline

Accessing HTML element

- Document method *getElementById*.
- Document and element method *querySelector*.
- Document and element method *querySelectorAll*.
- More methods will be introduced later.

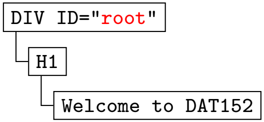
Method *getElementById*

- Returns element with a given HTML ID attribute.
- Returns **null** if no element with the given HTML ID.
- Observe, an HTML ID is unique within an HTML document.

Demo using *getElementById*

- HTML:

```
<div id="root">
    <h1>
        Welcome to DAT152
    </h1>
</div>
```



- JavaScript:

```
const rootElement = document.getElementById("root");
```

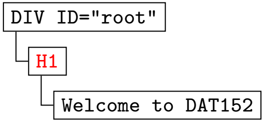
Method *querySelector*

- Returns element that matches a given CSS selector.
- If more elements match selector, only the first is returned.
- Returns **null** if there are no elements that match the selector.
- Throws error **DOMException** on invalid CSS-selector.
- Observe, you are supposed to have some knowledge on CSS selectors from other HVL courses.

Demo using *querySelector*

- HTML:

```
<div id="root">
    <h1>
        Welcome to DAT152
    </h1>
</div>
```



- JavaScript - *querySelector* as a method of *document*:

```
const element= document.querySelector("h1:first-child");
```

- JavaScript - *querySelector* as an element method:

```
const rootElement = document.getElementById("root");
const element= rootElement.querySelector("h1:first-child");
```

Method *querySelectorAll*

- Returns a list of all elements that match a given CSS selector.
- The list is empty if no elements match the CSS selector.
- Throws error **DOMException** on invalid CSS-selector.

Demo using *querySelectorAll*

- HTML:

```
<body id="root">
    <p>DAT152 is thought in the autumn of 2025.</p>
    <p>Demonstration of <var>querySelectorAll</var>.</p>
</body>
```

- JavaScript - *querySelectorAll* as method of *document*:

```
const elements= document.querySelectorAll("body *");

console.log(`BODY contains ${elements.length} HTML elements`);
```

- JavaScript - *querySelectorAll* as an element method:

```
const rootElement = document.getElementById("root");
const elements = rootElement.querySelectorAll("body *");
```

Outline

Modify HTML content

- Element property *textContent*.
- Element property *innerHTML*.
- Element method *insertAdjacentHTML*.
- More methods and properties will be introduced later.

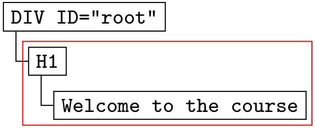
Element property *textContent*

- Returns or assigns text content of an HTML element.
- Any prior content is deleted on assignment.
- Returns or assigns pure text only.
- Safe to use on user data.

Demo using *textContent*

- Original HTML:

```
<div id="root">
    <h1>Welcome to the course</h1>
</div>
```

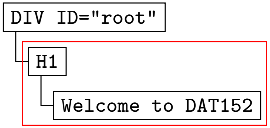


- JavaScript:

```
const element = document.querySelector("h1:first-child");
element.textContent = "Welcome to DAT152";
```

- Modified HTML:

```
<div id="root">
    <h1>Welcome to DAT152</h1>
</div>
```



Element property *innerHTML*

- Returns or assigns HTML content of an HTML element.
- Any prior content is deleted on assignment.
- Never use this property to assign user supplied data or data from other external sources!
- **Important:** If adding pure text, use a pure text approach, e.g. *textContent*, *innerText*, *insertAdjacentText*.

XSS attacks

Converting text to HTML, e.g. using *insertAdjacentHTML*, *innerHTML*, and *outerHTML* makes application vulnerable to XSS attacks.

Never use such metods if any part of the data originates from outside the JavaScript file itself!

Demo using *innerHTML*

- Original HTML:

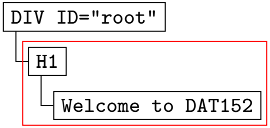
```
<div id="root"></div>
```

- JavaScript:

```
const rootElement = document.getElementById("root");
rootElement.innerHTML = "<h1>Welcome to DAT152</h1>";
```

- Modified HTML:

```
<div id="root">
    <h1>
        Welcome to DAT152
    </h1>
</div>
```



Element method *insertAdjacentHTML*

- Modifies the HTML structure.
- Never use this method on user supplied data or data from other external sources!
- **Important:** If adding pure text, use a pure text approach, e.g. *textContent*, *innerText*, *insertAdjacentText*.
- Using *insertAdjacentHTML* to modify DOM structure can be much more efficient than using property *innerHTML*.

Demo using *insertAdjacentHTML*

- Original HTML:

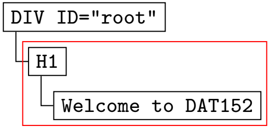
```html
<div id="root"></div>
```

- JavaScript:

```javascript
const rootElement = document.getElementById("root");
rootElement.insertAdjacentHTML(
    "beforeend",
    "<h1>Welcome to DAT152</h1>"
);
```

- Modified HTML:

```html
<div id="root">
    <h1>
        Welcome to DAT152
    </h1>
</div>
```



Adding content to ShadowRoot

- Later lectures will introduce ShadowRoot for GUI components.
- **ShadowRoot** content is added using property innerHTML, or using the methods setHTML or setHTMLUnsafe.
  - I.e. no *textContent*, *innerText*, *outerText*, *insertAdjacentHTML* or *outerHTML*.
- **ShadowRoot** method *setHTML* is XSS-safe.
- Method *setHTML* is only supported by Firefox, in nightly build.
  - Also defined for HTML elements, but not supported by any browser

HTML attributes reflected in JavaScript

- JavaScript objects properties can reflect HTML attributes.
- Observe that not all attributes are reflected.
- HTML:

```html
<A href="https://www.hvl.no/">Visit HVL</A>
```

- JavaScript:

```javascript
const anchor = document.querySelector("A");
anchor.href = "https://www.hvl.no/";
```

Element methods for accessing HTML attributes

- Element method getAttribute to get attribute.

```
const anchor = document.querySelector("A");
console.log(`Anchor has href ${anchor.getAttribute("href")}`);
```

- Element method setAttribute to set attribute.

```
const anchor = document.querySelector("A");
anchor.setAttribute("href","https://www.hvl.no/");
```

- Element method hasAttribute to check if element has attribute.

```
if (anchor.hasAttribute("href")) {
    console.log("Anchor has href attribute");
}
```

Custom data attributes

- User defined HTML attributes must begin with “data-”

```
<P data-course='DAT152'>Welcome to the course.</P>
```

- Attributes on the form “data-” are named custom data attributes.
- Property dataset provides read and write access to “data-” attributes.

```
const pelm = document.querySelector("P");
if (pelm.dataset.course !== undefined) {
    comnsole.log(`Course name is ${pelm.dataset.course}`);
}

const newcourse = 'DAT151';
pelm.dataset.course = newcourse;
```

- Observe that the “data-” part is removed from the attribute name.
  - For more rules on the property names, see e.g. dataset property.

Events and event handlers

- A DOM event is a signal in the browser that something has occured.
  - E.g. user clicked on a button.
- An event handler is JavaScript code that is run on an event.
  - The handler must be registered to run on the specific event signal.
- Events and event handlers will be covered in more details later.

Some examples of DOM events

- User clicked a button
- Mouse is moved in or out from a web element.
- Text is inserted into an input element.
- The web document has finished loading.
- A web element got or lost focus.

Method *addEventListener*

- Method *addEventListener* lets us attach an handler to an event.
- Attach an event handler to a click on an HTML button element:

```
const button = document.querySelector("button");

button.addEventListener("click",
    ()=>{console.log("Welcome to DAT152")}
);
```

- Attach an event handler to an event on the web document:

```
document.addEventListener("DOMContentLoaded",
    ()=>{console.log("Document is now in browser memory")}
);
```

Using *addEventListener*

```
const button = document.querySelector("button");

button.addEventListener("click",
     (event)=>{
          console.log(`You clicked on a ${event.target.tagName} tag`)
     }
);
```

- To attach code to be run on an event signal, the browser must know:
  1. The element that should react on the event signal.
  2. The event type to react on.
  3. The JavaScript code to run, i.e. the event handler.
- Can also specify the event phase and other properties – details later.
- Information on the event signal is give as parameter to event handler.

Event handlers and callbacks

- The event handler argument of *addEventListener* is a *callback*.
- A *callback* is a function given as parameter in a function call:

```
function f() { ... }
function g(f) { ... }
g(f);
```

- Above code will run *g* with function *f* as argument.
  - Function *f* is the callback.
  - Function *g* is given *f* itself as an argument, **not** the result of running *f*.
- Function code of *g* can run function *f*:

```
function g(f) {
  const result = f(22);
  // More code of function g
}
```

Callback and function call

- The argument is the function itself.
- The argument is not not the result of running the callback.
- There are no parenthesises “()” behind a callback parameter.
- Through callbacks, a parent object can run methods on an event signal managed by a child object.
    - Parent uses child API to register methods to be run on the event.

Callbacks, *this* and *bind*

- In event handler, keyword *this* is the HTML element of event handler.

```
function eventhhandler(){
    console.log(`'this' is the HTML button: ${this}`);
}

button.addEventListener("click",eventhandler);
```

  - True also if event handler belongs to a class or object.
- Function method *bind* can specify the value of *this*.

```
const newfunction = oldfunction.bind(newthis);
```

- Can also use arrow syntax for the event handler.
  - Uses the *this* of the surrounding context.

Manage the value of *this*

- Using the function property method *bind*:

```
class MyClass {
    controller (root) {
        const button = root.querySelector("button");
        button.addEventListener("click",this.method.bind(this));
    }

    method() { ... }
}

const occurence = new MyClass(document.getElementById("rootid"));
```

- Using an arrow function envelope:

```
class MyClass {
    controller (root) {
        const button = root.querySelector("button");
        button.addEventListener("click",(event) => {this.method()});
    }

    method() { ... }
}

const occurence = new MyClass(document.getElementById("rootid"));
```

FORM elements

- HTML FORM elements are targeted user input and user actions.
- Input elements allow user to supply data to application.
- The button element is targeted mouse clicks.
- User data of an input element is avaliable as element property *value*.
  - The HTML attribute *value* is the initial data of the element.
  - The JS property *value* is the current data of the element.

Demo with FORM elements

- HTML:

```html
<h1>Welcome to <span>course</span></h1>

<form>
    <fieldset>
        <legend>Fill in name of course</legend>
        <input type="text" placeholder="Name of course" />
        <button type="button">Register course name</button>
    </fieldset>
</form>
```

- JavaScript:

```javascript
function setcourse() {
    const inputElement = document.querySelector("input");
    const course = inputElement.value.trim();
    if (course === "") return;

    const spanElement = document.querySelector("h1").querySelector("span");
    spanElement.textContent = course;
}

const button = document.querySelector("button");
button.addEventListener("click", setcourse);
```

JavaScript element property *classList*

- Assign CSS class names **surname** and **student** to HTML element:

```
<SPAN class="surname student">Ole</SPAN>
```

- Property *classList* gives access to CSS class names of HTML element.
- Each class name will correspond to a class name object in *classList*.
- Property *classList* is a live NodeList of all class name objects.
- Property *classList* includes methods to work with the class name list.

Modify the display of an HTML element

The recommended approach to modify the display of an HTML element in JavaScript uses *classList* to add and remove class names.

Working with *classList*

- Property *length* is the count of class names of the HTML element.

```
const element = document.querySelector("p");
console.log(`Element has ${element.classList.length} class names`);
```

- Method *toggle* alternates in adding and removing a class name.
  - Returns **true** if class name was added to HTML element.
- Method *contains* checks if HTML element has a given class name.
- Method *remove* removes a class name object from HTML element.
- Method *add* adds a class name to HTML element.
- For all methods, see Element: classList property and DOMTokenList.