Web Programming
JavaScript Part II.

Leander Jehl | University of Stavanger

Outline

- So far
  - JavaScript syntax, control statements, variables, functions, objects
  - Built-in objects (Math, Array, etc.)
- Today
  - Event-driven programming
  - Manipulating the DOM

Events and event handling

- *Event-driven programming*: execution is triggered by user actions
- *Event* is a notification that something specific has occurred
- *Event handler* is a script that is executed in response to the appearance of an event
- HTML tags are used to connect events to handlers

```
<div class="green" ondblclick="myEvent('green double clicked');"></div>
```

event (double click)

event handler

Events

- Mouse events
- Keyboard events
- Frame/object events
- Form events
- … and more
  - Clipboard, print, media, animation, etc.
- See http://www.w3schools.com/jsref/dom_obj_event.asp for the full list

Mouse events

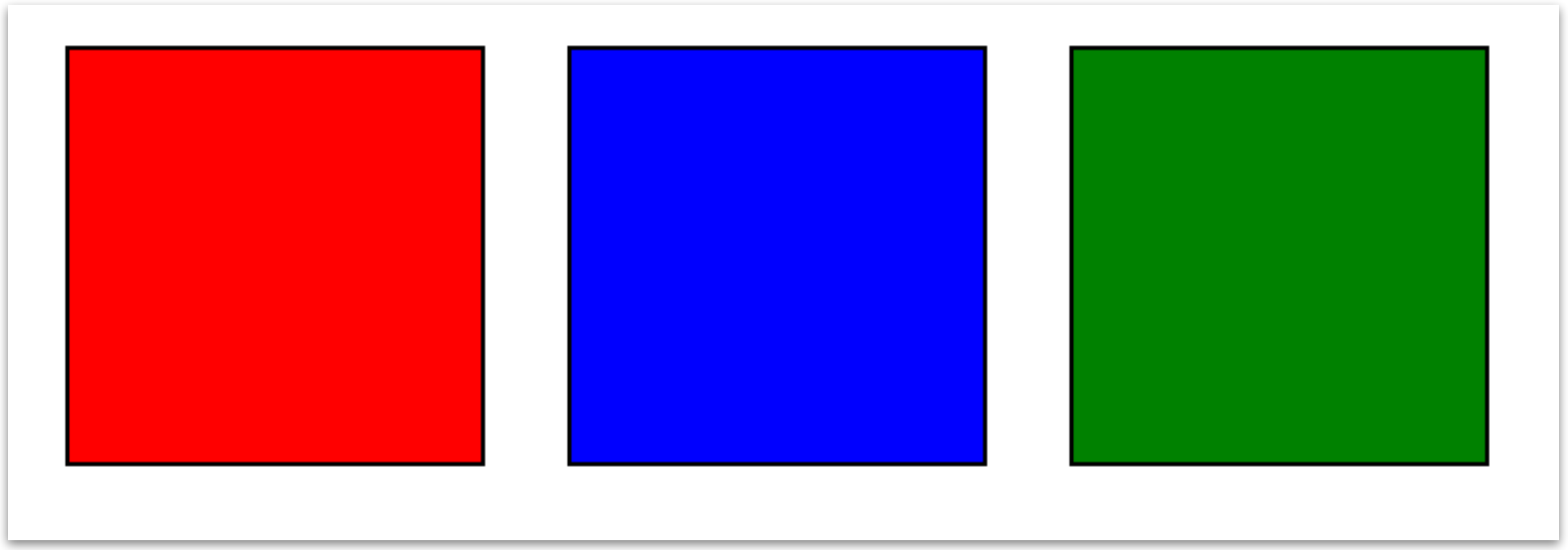
- **`onclick`** — click on an element
- **`ondblclick`** — double click on an element
- **`onmousedown`** — mouse button pressed over an element
- **`onmouseup`** — mouse button released over an element
- **`onmouseover`** — when the pointer is moved onto an element, or onto one of its children
- **`onmouseout`** — when a user moves the mouse pointer out of an element, or out of one of its children

Example

examples/js/events_dom/mouse_events.html

```
<script>
    function myEvent(message) {
        alert(message);
    }
</script>
```

```
<div class="red" onmouseover="alert('red alert');"></div>
<div class="blue" onclick="alert('blue clicked');" ></div>
<div class="green" ondblclick="myEvent('green double clicked');"></div>
```



Mouse event properties

- Further properties of the event can be accessed
  - **`button`** — which mouse button was pressed
  - **`clientX, clientY`** — coordinates of the mouse pointer, relative to the current window
  - **`screenX, screenY`** — coordinates of the mouse pointer, relative to the screen
  - **`shiftKey, ctrlKey, altKey, metaKey`** — boolean properties, reflecting the state of corresponding key: Shift, Ctrl, Alt or Command (Mac only)

Example

examples/js/events_dom/mouse_event_logger.html

```html
<script>
    function mhandle(event) {
        let msg = event.type
                + " button=" + event.button
                + " clientCoord=(" + event.clientX + ", " + event.clientY + ")"
                + " screenCoord=(" + event.screenX + ", " + event.screenY + ")"
                + (event.shiftKey ? " +shift" : "")
                + (event.ctrlKey ? " +ctrl" : "")
                + (event.altKey ? " +alt" : "")
                + (event.metaKey ? " +meta" : "");
        document.getElementById("log").innerHTML += msg + "\n";
    }
</script>
```

```html
<div onclick="mhandle(event);"></div>
```

Keyboard events

- **`onkeydown`** — when the user is pressing a key
- **`onkeypress`** — when the user presses a key (triggers after keydown)
- **`onkeyup`** — when the user releases a key

Working with keyboard events

- Keydown/keyup are for any keys
- Keypress is for characters
- Key event properties
  - **`keyCode`** — the scan-code of the key (i.e., which key was pressed; it’s the same for "a" and "A")
  - **`charCode`** — the ASCII character code
  - **`shiftKey, ctrlKey, altKey, metaKey`** — boolean properties, reflecting the state of corresponding key: Shift, Ctrl, Alt or Command (Mac only)

Example

examples/js/events_dom/keyboard_event_logger.html

```
<script>
    function khandle(event) {
        let msg = event.type
                + " keyCode=" + event.keyCode
                + " charCode=" + event.charCode
                + (event.shiftKey ? " +shift" : "")
                + (event.ctrlKey ? " +ctrl" : "")
                + (event.altKey ? " +alt" : "")
                + (event.metaKey ? " +meta" : "");
        document.getElementById("log").innerHTML += msg + "\n";
    }
</script>
```

```
<input type="text" id="kinput" onkeydown="khandle(event);"
onkeyup="khandle(event);" onkeypress="khandle(event);"/><br/>
Log:<br/>
<textarea rows="18" id="log"></textarea>
```

Frame/object events

- **`onload`** — when an object has loaded
  - Most common usage: **`<body onload="…">`**
- **`onpageshow`** — when the user navigates to a webpage
- **`onpagehide`** — when the user navigates away from a webpage
- **`onresize`** — when the document view is resized
- **`onscroll`** — when an element's scrollbar is being scrolled

Example

examples/js/events_dom/frame_events.html

```
<body onload="alert('page loaded');"
      onpageshow="console.log('navigated to page');"
      onpagehide="console.log('navigated away from page');">
```

Exercises #0, #1

github.com/dat310-2024/info/tree/master/
exercises/js/events_dom

Form events

- `onfocus` — when an element gets focus
- `onblur` — when an element loses focus
- `onchange` — when the content/state of a form element has changed (for `<input>`, <select>, and <textarea>)
- `oninput` — when an element gets user input (for <input> and <textarea>)
- `onsubmit` — when a form is submitted
- `onreset` — when a form is reset

onchange vs. oninput

- **`oninput`** occurs immediately after the value of an element has changed
- **`onchange`** occurs when the element loses focus, after the content has been changed
- **`onchange`** also works for <select> (not just <input> and <textarea>)

Example

examples/js/events_dom/form_events.html

```
<script>
    function setfocus(element) {
        element.style.backgroundColor = "yellow";
    }
   function input(element) {
        console.log(element.name + " oninput: " + element.value);
    }
</script>
```

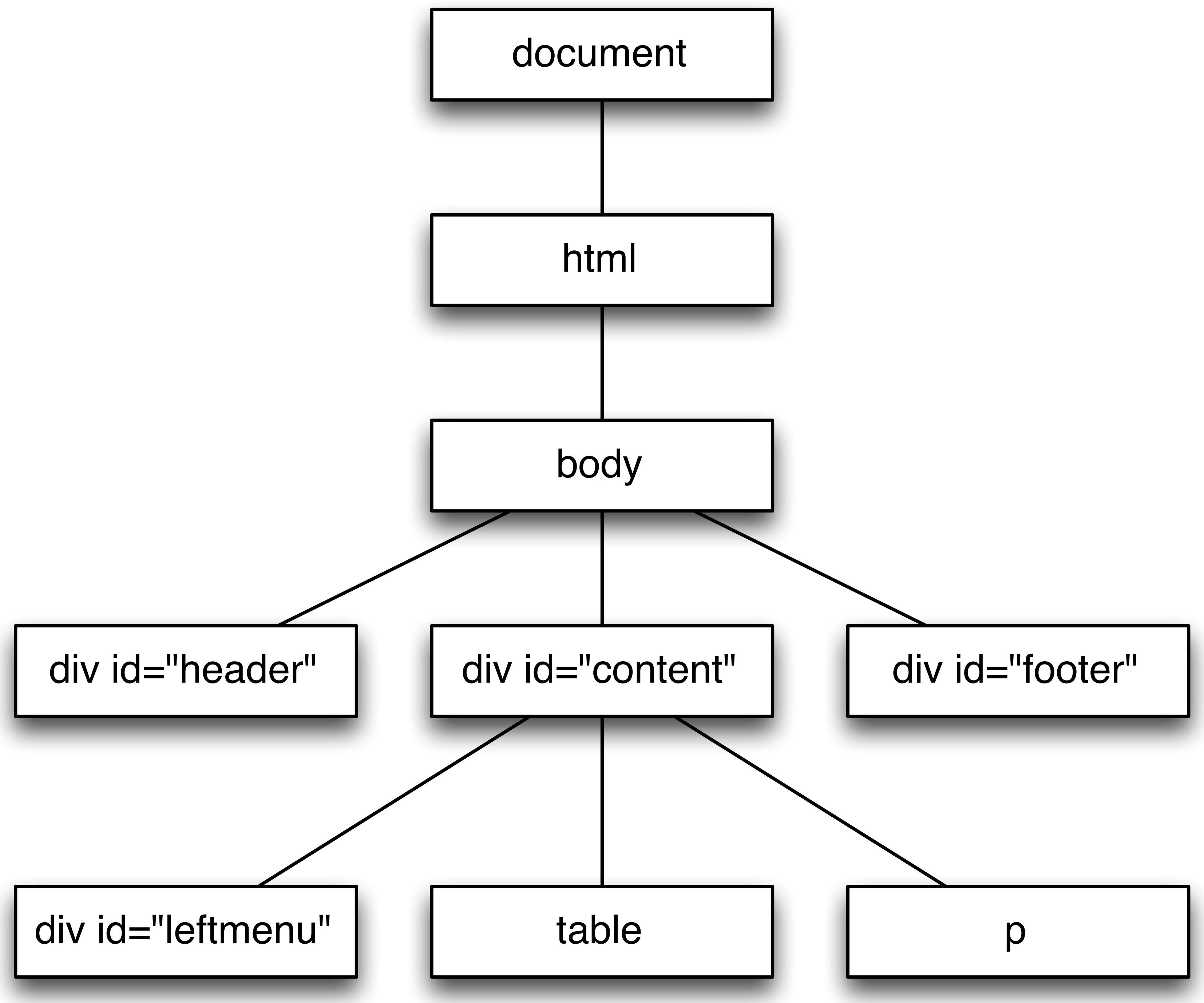
```
<form name="test" onsubmit="alert('form submitted');">

<input type="text" name="name" size="20" placeholder="Firstname, lastname"
       onfocus="setfocus(this);"
       onblur="losefocus(this);"
       oninput="input(this);"
       onchange="change(this);"/>
```

`this` refers to the this particular <input> element

Document Object Model (DOM)

- Internal model of the HTML page
- Consistent way (across all browsers) to gain access to the structure and content of HTML
- A tree of HTML elements
- **Object** model
  - Each HTML elements is an object (with methods and properties)
  - Plus two additional objects: document and window



Interacting with the DOM

- JavaScript can interact with the DOM to get access to the elements and the content in them
  - Getting and setting the attributes of elements
  - Creating or adding elements
  - Removing elements

Wait until the page has fully loaded!

- In most cases, we need to wait for the DOM to be fully created before start executing JavaScript code

```
<script>
    function init() {
        ...
    }

    window.onload = init;
</script>
```

The **`init()`** function is assigned to the onload event of the (browser) window.

Wait until the page has fully loaded!

- In most cases, we need to wait for the DOM to be fully created before start executing JavaScript code

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

Scripts with **`defer`** will be executed after DOM is loaded.
Only for external files!

Finding HTML elements

- Finding elements by ID
  - Typically saved to a variable so that we can refer to the element throughout the code

```
let element = document.getElementById("someid");
```

- Finding elements by tag/class name
  - E.g., listing names and values of all input elements

```
let x = document.getElementsByTagName("input");
for (let i = 0; i < x.length; i++) {
    console.log(x[i].name + ": " + x[i].value);
}
```

Getting properties of HTML elements

- **`id`** — the value of the id attribute
- **`innerHTML`** — the HTML content (between the opening and closing tags)

```
let mydiv = document.getElementById("mydiv");
console.log("HTML content: " + mydiv.innerHTML);
```

- **`tagName`** — the name of the HTML tag (in uppercase, e.g., P, DIV, H1, etc.)
- **`getAttribute()`** — a specific attribute's value
- See a full list of properties and methods of the element object http://www.w3schools.com/jsref/dom_obj_all.asp

Changing HTML elements

- Change the inner HTML

```
document.getElementById("mydiv").innerHTML = "new content";
```

```
document.getElementById("mydiv").innerHTML = "<p>new content</p>";
```

- Change the text inside the element

```
document.getElementById(“mydiv").textContent = "new content";
```

  - cannot add new HTML elements

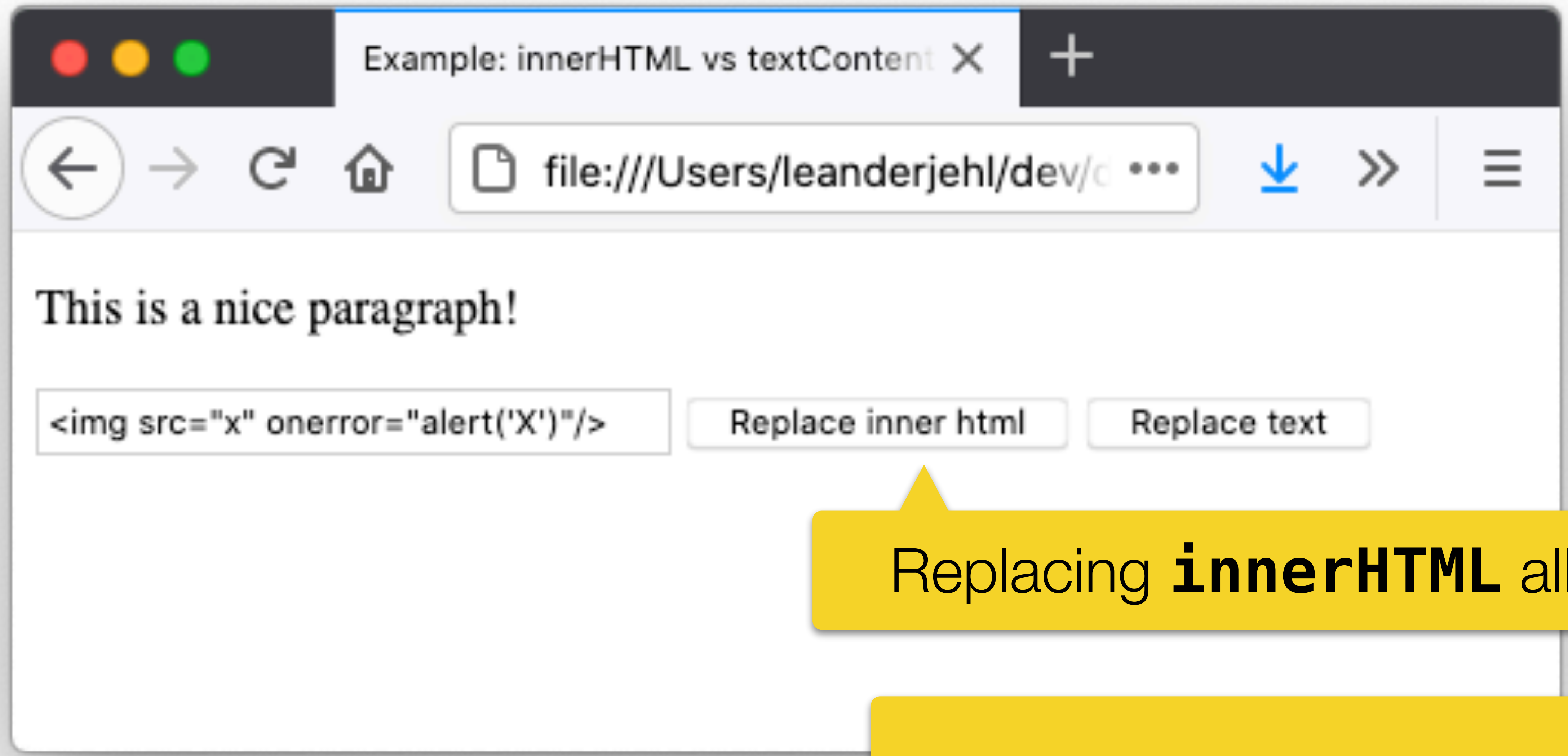
- Change the value of a specific attribute

```
document.getElementById("myImage").src = "landscape.jpg";
```

```
document.getElementById("myImage").setAttribute("src", "landscape.jpg");
```

Example

Examples/js/events_dom/innerHTML.html


Example: innerHTML vs textContent
file:///Users/leanderjehl/dev/

This is a nice paragraph!

```
<img src="x" onerror="alert('X')"/>
```

Replace inner html
Replace text

Replacing **`innerHTML`** allows to inject JS code.

Do not insert user input using **`innerHTML`**!

Changing CSS properties

- `style.x` — the value of a style property **x**
  - See http://www.w3schools.com/jsref/dom_obj_style.asp
- Change the style property of an HTML element

```
document.getElementById("mydiv").style.height = "200px";
```

```
document.getElementById("mydiv").style.backgroundColor = "blue";
```

camelCase: backgroundColor in JS
background-color in CSS

- Add/remove classes assigned to a HTML element

```
let div = document.getElementById("mydiv");

if (!div.classList.contains("border")) {
    div.classList.add("border");
}
else {
    div.classList.remove("border");
}
```

Assigning events to elements (1)

- Setting the element's **on…** attribute in HTML

```
<script>
    function dosomething() {
        …
    }
</script>
```

```
<div id="mydiv" onclick="dosomething()"></div>
```

Assigning events to elements (2)

- Modifying the element’s **on…** property

```
<script>
    function dosomething() {
        …
    }
    function init() {
        document.getElementById("mydiv").onclick = dosomething;
    }
    window.onload = init;
</script>
```

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

Assigning events to elements (3)

- Using event listeners
  - Attaches an event handler to an element without overwriting existing event handlers
  - Multiple event handlers might be added to one element

```
document.getElementById("myBtn").addEventListener("click", showAlert);
document.getElementById("myBtn").addEventListener("click", log);
```

  - Event listeners can be removed too

```
document.getElementById("myBtn").removeEventListener("click", showAlert);
```

  - See http://www.w3schools.com/js/js_htmldom_eventlistener.asp

Passing parameters to event handlers

- Functions assigned to events from JS cannot take arguments
  - Otherwise the function is immediately executed

```
function changeColor(element) {
    …
}
function init() {
    let mydiv = document.getElementById("mydiv");
    mydiv.style.backgroundColor = "blue";
    mydiv.onclick = changeColor(mydiv);
}
```

Wrong! changeColor() executes immediately

- Solution: use an "anonymous function" that calls the specified function with the parameters

```
mydiv.onclick = function() {changeColor(mydiv);}
```

Example

examples/js/events_dom/event_listeners.html

```
function init() {
    // assign showAlert() and log() to all divs
    let x = document.getElementsByTagName("div");
    for (let i = 0; i < x.length; i++) {
        x[i].addEventListener("click", showAlert);
        x[i].addEventListener("click", log);
    }

    // remove log() from elements that have the nolog class
    x = document.getElementsByClassName("nolog");
    for (let i = 0; i < x.length; i++) {
        x[i].removeEventListener("click", log);
    }
}
```

Exercises #2 (#2b)

github.com/dat310-2024/info/tree/master/
exercises/js/events_dom

Working with forms

- Different element properties, depending on the type of input
- Common
  - **`name`** — name attribute
  - **`type`** — which type of form element it is
  - **`disabled`** — whether the element is disabled or not
  - **`form`** — reference to the form that contains the element
  - **`required`** — whether the input must be filled out before submitting the form

Input text object

- <input> and <textarea> elements
  - **`value`** — get or set the value of the element
- See
  - http://www.w3schools.com/jsref/dom_obj_text.asp
  - http://www.w3schools.com/jsref/dom_obj_textarea.asp

```
<script>
    let name = document.getElementById("name");
    console.log("Name: " + name.value);
</script>
```

```
<input type="text" name="name" id="name"/>
```

Select list

- Properties
    - `length` — number of options in the list
    - `value` — value of the selected option
    - `selectedIndex` — index of the selected option
    - `options[index].value` — value of the option at a given index pos.
    - `options[index].text` — text corresponding to the option at a given index position
- See
    - http://www.w3schools.com/jsref/dom_obj_select.asp

Select list example

examples/js/events_dom/form_elements.html

```
<script>
    function processForm() {
        let name = document.getElementById("name");
        console.log("Name: " + name.value);

        let country = document.getElementById("country");
        for (let i = 0; i < country.length; i++) {
            console.log("[" + country[i].value + "] " + country[i].text
                        + (country[i].selected ? " selected" : ""));
        }
        console.log("Selected: " + country.options[country.selectedIndex].text);
    }
</script>
```

```
<select name="country" id="country" onchange="processForm();">
    <option value="--">Select</option>
    <option value="NO">Norway</option>
    <option value="SE">Sweden</option>
    <option value="DK">Denmark</option>
</select>
```

Exercises #3

github.com/dat310-2024/info/tree/master/
exercises/js/events_dom

Input checkbox and radio

- Properties
  - **`checked`** — sets or returns the checked state
- See
  - http://www.w3schools.com/jsref/dom_obj_checkbox.asp
  - http://www.w3schools.com/jsref/dom_obj_radio.asp

Checkbox example

examples/js/events_dom/form_events.html

```
<script>
    function processForm() {

        let delivery = document.getElementsByName("delivery");
        for (let i = 0; i < delivery.length; i++) {
            console.log("[" + (delivery[i].checked ? "X" : " ") + "] "
                        + delivery[i].value);
        }
    }
</script>
```

```
<label>Delivery
    <input type="radio" name="delivery" value="normal">Normal
    <input type="radio" name="delivery" value="extra">Extra
    <input type="radio" name="delivery" value="hyper">Hyper
</label>
```

Form validation using JavaScript

```
<script>
    function checkForm() {
        let valid = true;

        // perform input check
        // set valid to false if it fails

        return valid;
    }
</script>
```

```
<form name="test" action="…" onsubmit="return checkForm();">
…
</form>
```

If the **`checkForm()`** function returns **`true`** the form will submit. If **`false`**, the form does nothing.

Form validation using JavaScript

```
<script>
    function checkForm() {
        let valid = true;

        // perform input check
        // set valid to false if it fails

        return valid;
    }
</script>
```

```
<form name="test" action="…" onsubmit="return checkForm();">
…
</form>
```

If the **`checkForm()`** function returns **`true`** the form will submit. If **`false`**, the form does nothing.

Process form only in Javascript

- When submitting a form with `action=“”`, the page is reloaded

```
<form action="">
```

- To avoid this you can
  - Specify the form is submitted to Javascript:

```
<form action=“javasript:handle()“>
```

  - Use a button instead of a submit element:

```
<input type=“button“ value=“button“>
```

Exercises #4

github.com/dat310-2024/info/tree/master/
exercises/js/events_dom

Source of truth

- Can be in the DOM or in JS

```
// This function uses the DOM (html) as source of truth
function add(){
    counter = document.getElementById("count");
    count = parseInt(counter.innerText) + 1;
    counter.innerText = count;
}

// a global variable
let count = 0;
// increment uses the global count
function increment(){
    count += 1;
    document.getElementById("count_2").innerText = count;
}
```

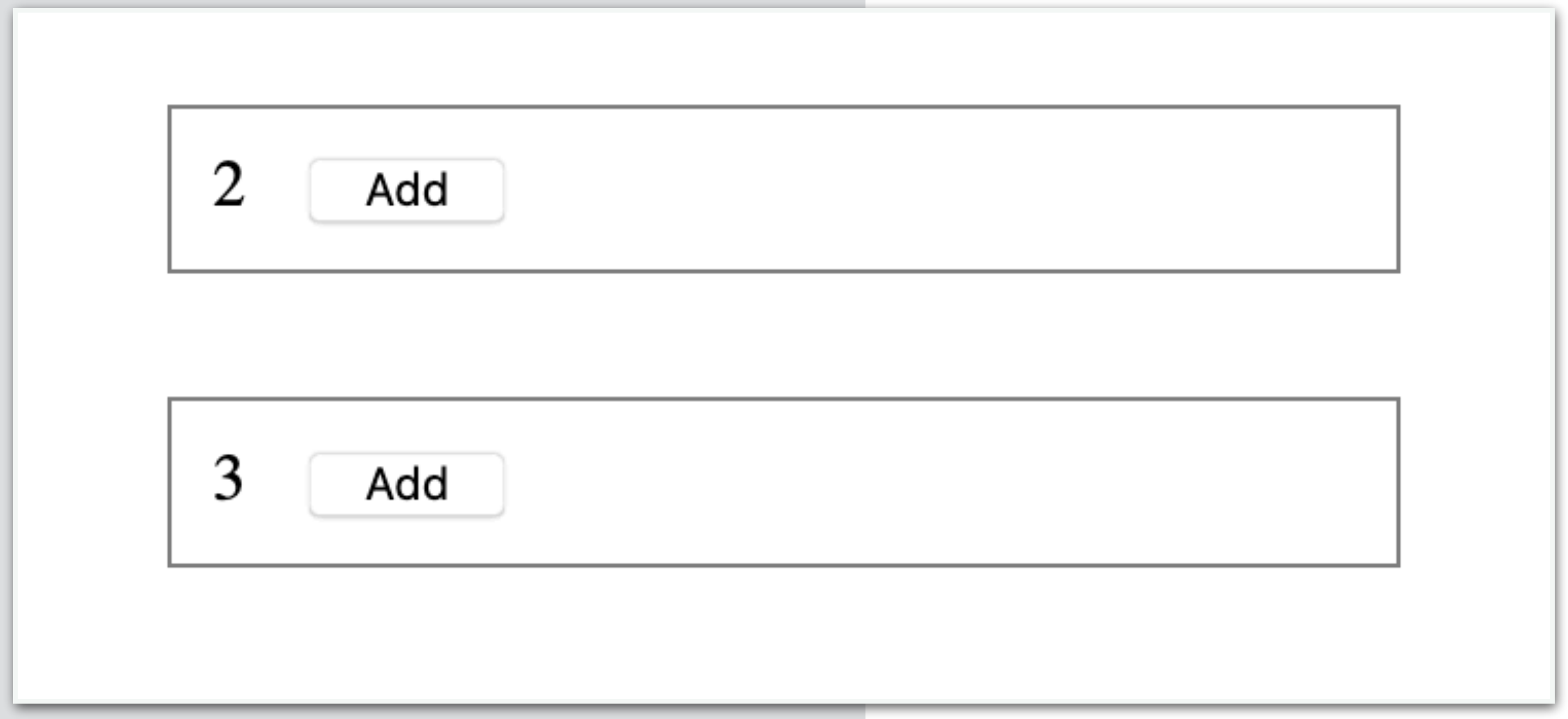
Example

Examples/js/more/source_of_truth.html

Use JS as source of truth in Assignment 4!

```
// This function uses the DOM (html) as source of truth
function add(){
    counter = document.getElementById("count");
    count = parseInt(counter.innerText) + 1;
    counter.innerText = count;
}

// a global variable
let count = 0;
// increment uses the global count
function increment(){
    count += 1;
    document.getElementById("count_2").innerText = count;
}
```



References

- W3C JavaScript and HTML DOM reference
  http://www.w3schools.com/jsref/default.asp
- W3C JS School
  http://www.w3schools.com/js/default.asp
- Mozilla JavaScript reference
  https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference