Assignment 3

Recommended readings:

- Lecture slides as starting literature
- MDN Links within slides for details, especially:
  - setInterval
  - clearInterval
  - replace
  - alert

Note: All exercises must be solved with plain JavaScript and CSS, using additional libraries or frameworks is not allowed.

The workload of this assignment corresponds to 2+ weeks of work (due to two public holidays), so it is strongly advised to start early and split up your effort over the available timespan (e.g. by solving Exercises 1-3 in the first week and 4-5 in the second week).

Exercise 1 – DOM Operations

Given the file *DOM_example.html*. Create a JavaScript file with the following code and include it in the page. Open the page in your browser, and explain each line of the output in detail:

```
var link = document.getElementsByTagName("a")[1];
console.log(link.attributes["href"].nodeValue);
console.log(link.getAttributeNode("href").nodeValue);
console.log(link.href);

console.log(link.attributes.length);
console.log(link.attributes[0].nodeType);
console.log(link.firstChild.nodeType);
console.log(link.parentNode.nodeType);
console.log(link.parentNode.nodeName);

var div = document.getElementById("wiki_en");
console.log(div.children[1].lastElementChild.previousElementSibling.src);
console.log(div.innerHTML);

console.log(document.querySelector("#wiki_ja p").innerText);
console.log(document.querySelectorAll(".europe h2")[0].firstChild.nodeValue);
```

Moreover, explain the consequences of executing the following JavaScript code:

```
var link = document.getElementsByTagName("a").item(2);
link.removeAttribute("target");
```

Experiment with the Chrome/Firefox/Safari Developer Tools (Console Tab), which also allows you to execute JavaScript code. You can also use it to interactively explore the result of a DOM selection command. For example, enter *document.getElementsByTagName("div")* and explain the result.

Exercise 2 – Calculating Glass Prize

This exercise will extend your solution of the glazier shop page from the Assignment 2.2. First, extend the form with a read-only input element with the *price* label. Then implement the following functionality in the file *calculate.js*:

- Implement a function *calculate(width, height, thickness)* that calculates and returns the price based on the following formula:
  - prize = width * height * thickness / 64 if the glass is coloured, or
  - prize = width * height * thickness / 100 if the glass is in a shade of grey
- Implement a function *update()* that updates the price input field accordingly, given that all required inputs are given and have valid values. The price should be rounded to two decimal places. If no valid values are given, the input field should show a hyphen (-).
- Attach the update function as an event handler for the relevant input elements such that update() is called whenever a value is changed.

Hints:

- A shade of grey is characterized by the identical R, G, and B values.
- All variables of the formula are assumed to be in cm. You may need to convert the values accordingly.

Exercise 3 – Making an HTML Page Editable

Given an HTML document containing a sequence of sections, where every section has one h1 element as a child (for example, the document of the Exercise 2.4). Create a JavaScript file *editable.js* dealing with the above document which fulfills the following requirements:

- Your code adds a form containing a select box, a text input, a textarea and an initially disabled submit button after the last section.
- The select box contains the headings of the document sections and allows the user to select a section by selecting its heading. After selection, the text input element shows the heading of the selected section and the textarea shows the innerHTML of the selected section. and the submit button is activated.
- When the submit button is clicked, the content of the selected section is updated with the data of the input element and the textarea. If the heading is changed in the textarea and in the input field, only the value of the input field applies. Be sure to correctly handle cases where the user removes the heading element in the HTML source. Also be sure to update the content of the select box after a heading is changed.

Do not change the HTML document except adding the reference to your JavaScript file! Your code should not only work for this specific file, but also for any other HTML document with the described structure.

Exercise 4 – Image Gallery

The folder *gallery* contains the basic picture gallery from the lecture slides. Extend the gallery such that when the user enters full window mode by clicking a picture, several control buttons appear (and disappear when returning to overview mode):

- "*previous*" showing the previous picture, or if it is not available, the last picture.
- "*next*" showing the next picture, or if it is not available, the first picture.
- "*play/pause*" implementing an automatic slideshow showing the pictures in an infinite loop until *pause* or *exit* is clicked. The duration for showing each picture should be specified using an input field (or alternatively a range slider) next to the buttons (default value: 3 seconds).
- "*exit*" for stopping the running slideshow and closing the full window mode.

Hint: you can either statically include the buttons in the HTML and show/hide them using CSS attributes, or dynamically add/remove them programmatically. In any case, you should use CSS to apply some style to the buttons.

Important: Your script should work with any HTML content following the format of the HTML document of the lecture.

Exercise 5 – Puzzle Game in JavaScript

The folder *puzzle* contains the HTML document *puzzle.html*. The page shows a grid with 4 cells and a list of 4 images. The goal of the game is to move each image of the list to the correct grid cell for solving the puzzle.

- Moving images should be implemented like in the lecture slides 3-40 and 3-41.
- If the user moves an image onto a cell in the grid and clicks (the mouse position is the relevant position), the background of the specified grid cell is set accordingly, and the original image element is removed.
- If there is already another picture in the grid cell, the previous picture is "moved back" to the list of pictures.
- When all pictures are in the correct position of the grid, the user wins and a corresponding message is displayed (e.g., by using alert(), but more sophisticated custom solutions are welcome!)
- The "grid" is defined by the position of the four div elements. The correct solution is encoded in the data attribute *"data-result"* of each div.
- While you should apply the general principle of the lecture for moving the images to the grid you should ensure that the mouse is not positioned above the currently moved image during moving. This makes is easier to interact with the grid.
- Bonus exercise (optional): The appearance of the mouse cursor should change "intuitively" when hovering over an image and during dragging (see CSS attribute *cursor*).