Assignment 4

Recommended readings:

- Lecture slides as starting literature
- MDN Links within slides for details
- Understanding Closures

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

Exercise 1 – Scope, Closures, Currying

Answer following questions:

a) What does the JavaScript `Window` Object represent and how can it be used?
b) What is the notion of scope in JavaScript?
c) What is the difference between declaring variables with `var`, `let`, `const` or no keyword at all? What happens when `'use strict'` is invoked for a script?

Given following two JavaScript code snippets, discuss and reason about their outputs. What are the functional concepts behind them?

```
function createCounter() {
    let counter = 0;
    let incFunction = function() {
        counter += 1;
        return counter;
    }
    return incFunction;
}
const increment = createCounter();
const c1 = increment();
const c2 = increment();
const c3 = increment();
console.log('result:', c1, c2, c3);
```

```
let greet =
    gender =>
        name =>
            'Dear ' + (gender === 'female' ? 'Mrs. ' : 'Mr. ') + name;
let women = ['Susan', 'Karen', 'Amanda'];
let greetWoman = greet('female');
for(let w of women)
    console.log(greetWoman(w));
```

Exercise 2 – JavaScripts Objects

a) Create an object for an image that sets width, height, and bitdepth (number of bits used to save color information for every pixel) by using the JSON syntax. Extend the object by a function `computeSize()`, which computes the raw size needed for the image (in Mebibytes) and stores it in the instance variable `rawsize`. Furthermore, the function should also compute the number of MegaPixels and save it in the instance variable `megapixels`. Call the function for the object and print out (to the console) the values of both instance variables.

b) Define a constructor function *Image* that allows to create objects like above (with `width`, `height`, and `bitdepth`). Instead of a `computeSize()` function, set `rawsize` and megapixels already in the constructor function. Additionally, define a function `print()` within *Image*, which prints out all properties and values of the object (using a for loop) to the console. Create a few objects and test the print function with them.

c) Extend the prototype of Image by another function `printMore()`, which prints everything `print()` does, but also the ratio between width and height. Test your implementation with the objects created in Exercise 2b. What is the difference between `print()` and `printMore()`? Can you alter any of those functions during runtime?

d) Create another constructor function *Video*, which requires width, height, bitdepth, duration (in seconds), and framerate (in frames per second; a 'frame' is one image in the video). Video inherits from Image (i.e. you should call the constructor function of Image). In addition to Image it also has a function `totalFrames()`, which returns (based on duration and framerate) the total number of frames in that video. Test your implementation by creating a few objects, calling `print()`, and checking the return value of `totalFrames()`. Can you also call `printMore()` on these objects?

Exercise 3 – JavaScript Classes

Use the ECMA Script 6 extensions to define classes for *Image* (including a constructor) and *Video* (which inherits from Image), with the same instance variables and methods as used in Exercise 2. Test both classes by creating a few objects.

Exercise 4 – JSON Image Gallery

Extend the picture gallery from Assignment 3 by implementing the functionality of retrieving picture URLs from a server-side JSON file. An altered version of the gallery can be found in the *ex4* folder. Your task is to complete the object `ImageLoader`:

- Fully implement the `load` function of `ImageLoader`.
- Retrieve `gallery.json` via `XMLHttpRequest`.
- Parse the retrieved results and create appropriate Image elements for every contained image.
- Dynamically add the elements to `<div class="gallery">`.
- Initialize the gallery afterwards using the function `initGallery`.

Hint: Keep in mind that you need to host your files via an HTTP server, e.g. XAMPP on `localhost`, else HTTP requests will not work.

Exercise 5 – JSON Gallery Library

Extend the picture gallery from Assignment 3 even further by creating a Gallery library class `JsonGallery` (see *ex5* folder), which constructs a gallery within a `div` element passed to its constructor. Complete `JsonGallery.js` by adapting your previous code to fit the JS class paradigm and additionally take the following into account:

- Only edit the JavaScript files for this exercise – DOM elements should be created dynamically, e.g. use `document.createElement`/`element.innerHTML`.
- Again, image URLs must be retrieved from the server but this time use the `fetch` API for client-server communication.
- Also, be sure to use *Promises* when requesting resources.
- The gallery is now partitioned into two pages (`gallery_pg1.json`, `gallery_pg2.json`), which should be requestable one at a time in order to replace the images currently in the gallery. For this, you should create additional gallery page navigation buttons (e.g.: 'previous page'/'next page') on the page displaying the small images. Pressing one of these buttons should start a request for a new JSON file and setup the gallery according to the retrieved results.

Hint: Keep in mind that you need to host your files via an HTTP server, e.g. XAMPP on `localhost`, else HTTP requests will not work.