Assignment 4

Recommended readings:

- Lecture slides as starting literature
- MDN Links within slides for details
- Understanding closures
- JavaScript's event loop explained
- Working with objects

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

Exercise 1 – Scope, Closures

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?
d) Given the following two JavaScript code snippets, discuss and compare their outputs.

```
function checkScope() {
  "use strict";
  var i = "function scope";
  if (true) {
    var i = "block scope";
    console.log("BlockScope i = ", i);
  };
  console.log("FunctionScope i = ", i);
  return i;
};

const check = checkScope();
console.log(check);
```

```
function checkScope() {
  "use strict";
  let i = "function scope";
  if (true) {
    let i = "block scope";
    console.log ("BlockScope i = ", i);
  };
  console.log("Function scope i = ", i);
  return i;
};

const check = checkScope();
console.log(check);
```

e) Given the following two JavaScript code snippets, discuss and reason about their different outputs.

```
for (var i=0; i<3; i++){
    setTimeout(() => {
        console.log(i);
    }, 1000);
}
console.log('After the Loop');
```

```
for (let i=0; i<3; i++){
    setTimeout(() => {
        console.log(i);
    }, 1000);
}
console.log('After the Loop');
```

Exercise 2 – Closures, Currying, Arrow Functions

a) 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));
```

b) Write an arrow function that takes an array as the input and filters out the numbers inside the array which are less than zero and are not integers. The function should return the resulting array.

c) Change the previous code so as to return the **square** of integer and positive values.

d) Why do we **need** closures? Inspect the following code that increments the counter by one, every time that we click on the button '`Click Me`'.

`Click.html`

```
<html>
     <head><script src=”click.js” defer></script></head>
     <body>
          <button id="myBtn">Click Me</button>
     </body>
</html>
```

`click.js`

```
var clickCount = 0
document.getElementById("myBtn").addEventListener("click",
  function() {
    alert(++clickCount);
  });
```

The problem with the aforementioned approach is that "clickCount" is a global variable. In other words, any script in this page can access this variable and change its value accidently. We aim to protect this variable. To protect a variable, we could make it a local variable within a function. This way, the variable can only be changed by calling the function. Implement the following code and reason about the problem preventing the counter from incrementing.

click.js

```
function incrementClickCount(){
  var clickCount = 0;
  return ++clickCount;
}

document.getElementById("myBtn").addEventListener("click",
  function() {
    alert(incrementClickCount());
  });
```

Use a closure to fix the problem!

Exercise 3 – Plain JavaScript Objects

a) Taking an object variable containing the first name, last name, and university name of one person, create a welcome message as follows: "Dear first-name last-name from university-name, welcome to the Web Technologies course."

b) An object containing the names of five universities is provided as follows:

```
const UNI = {
  names:["University of Vienna", "Medical University of Graz",
  "Medical University of Vienna", "Medical University of
  Innsbruck", "University of Klagenfurt"]};
```

Write a JavaScript function to return the following array of HTML list element.

```
["<li class="Name">University of Vienna</li>",
 "<li class="Name">Medical University of Graz</li>",
 "<li class="Name">Medical University of Vienna</li>",
 "<li class="Name">Medical University of Innsbruck</li>",
 "<li class="Name">University of Klagenfurt</li>"]
```

c) An object containing the information associated with one student exists as follows:

```
var STUDENTS = {
    firstNames:["Andy"],
    lastNames:["Lb"],
    ages: [32],
    fieldOfStudy: ["Information Technology"]};
```

Write a JavaScript function called "`addNew`" to add the information of new students to the object using the following code:

```
STUDENTS.addNew(["Marco", "Fr", 34, "Electrical Engineering"]);
```

Exercise 4 – JavaScript Prototypes

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 5 – 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.