Università della Svizzera italiana

Facoltà di scienze informatiche

**Web Atelier**
Prof. Cesare Pautasso
Masiar Babazadeh
Vincenzo Ferme
Andrea Gallidabino
Vasileios Triglianos

# JavaScript Exercises – 18.09.2015 / due 23.09.2015

The goal of these exercises is to test and improve your understanding of JavaScript as a functional scripting language for the Web browser.

First get and extract the exercise skeleton from Moodle. All the code should be written directly into the template ("script.js") provided in the skeleton package. After completing the exercise, repack the skeleton folder and upload it to Moodle.

**Exercise 1. Functional JavaScript (50 points)**

a) **Scalar Product** (10 points)

Implement a function `scalar_product(a,c)` which returns an array computed by multiplying each element of its input array a with the input scalar value c. (+5 Extra points if you use the map function to implement the scalar product instead of a for loop).

b) **Inner Product** (10 points)

Implement a function `inner_product(a,b)` which returns a value computed by summing the products of each pair of elements of its input arrays a, b in the same position

c) **Itemize** (5 points)

Implement a function `itemize(s)` which given a string s (which could also be empty) returns the string surrounded by the HTML `<li>` element tags. (e.g., `itemize("X")` -> `"<li>X</li>"`).

d) **MapReduce** (15 points)

Implement a function `mapReduce (f, a, seed)` which maps all elements of the array a through the function f and then returns a single value computed by adding all mapped array elements together with the provided seed. The seed is an optional parameter, which defaults to the empty string.

e) **getNumberSequence** (5 points)

Implement a function `getNumberSequence(c)` which given a positive integer will return a string containing all numbers between 1 and the number c. The numbers should be separated by `", "` (comma space).

f) **Count** (5 points)

Implement a function `count(s)` which inverts the `getNumberSequence` function.
`count(getNumberSequence(x)) == x`, for all integers x > 0.

Make sure that all tests are passed without failed assertions.

**Exercise 2. Letter frequency analysis (30 points)**

Hint: You should use Array only for numeric indexes that are continuous. Use objects when you want string indexes like: `var myObj = {}; myObj['A'] = 3;`

In this exercise you will create a set of JavaScript functions that perform letter frequency analysis on a given string and display the results to the users.

a) Write a `function letter_frequency(s)` that calculates the number of occurrences of letters in a given string, it returns an array indexed by the letter characters found in the string.

*Usage:*

```
var a = letter_frequency("Hello");
a["H"] == 1; a["E"] == 1; a["L"] == 2; a["A"] == undefined;
```

b) Write a function `display_letter_frequency(a,dom)` which display the output of the letter frequency analysis in an HTML table generated within the `dom` element passed as parameter. The output contains the string entered by the user and the letter frequency analysis results next to each letter. For instance, Below is a sample output for the string **"Hello"**:

| | |
|---|---|
| H | 1 |
| E | 1 |
| L | 2 |
| O | 1 |

It should be noted that white spaces or numbers are treated as letters and letters are counted without regard to their case (case-insensitive analysis).

Hint: use the `for (var x in a)` construct to iterate over the array with non-numeric indexes.

c) Implement the `online_frequency_analysis(dom)` function to link the provided input text field in the test page with the table so that as a user types a text in the input text field the table is updated to reflect the newly entered text. The function is already set up as an event handler for the input text field, whose DOM object is passed in its input parameter `dom` whenever the `keyup` event is triggered by the user.

**Exercise 3. Play/Pause function without globals (20 points)**

In this exercise you will implement the toggle between play and pause for the music player of exercise 1. Write a function `player(initial_state)` that when executed returns a function which toggles the player state between `"play "` and `"pause "`. More specifically:

- if the `initial_state` equals `"pause"` then the first time the returned function is invoked it should return the String `"play"`.
- if the `initial_state` equals `"play"` then the first time the returned function is invoked it should return the String `"pause "`.
- if there is no `initial_state` argument passed to the player function, the original state is set to `"pause"`, so the first time the returned function is invoked it should return the String `"play "`.

The `player` and the returned function **must not** use global variables.

**Exercise 4. Clock (Extra 30 points)**

In this exercise you will implement a clock that can display: (a) the current time; (b) the time elapsed since the clock was last clicked. This information will be rendered inside the `DOM` element with id `"clock"`. When the clock is clicked, the information should toggle between the aforementioned states.

1. Implement the body of the `clock()` function. This function has two responsibilities:
   a) to return a function `toggleClock(evt)` which you will attach to a click event handler for the element `"clock"`.
   b) to update the `"clock"` `DOM` element with the current time or the time elapsed in the HH:MM:SS format.

The behavior of the clock is the following:

- When the page loads, the clock displays the current time and keeps it tick.
- When the clock is clicked, it switches to 'timer' mode displaying the time elapsed since the clock element was last clicked (starting at `00:00:00`) and keeps it tick

Do not use global variables. All required click event handlers are already set up for you on the test page. It should not be necessary to change them to solve the exercise.

Hints:
1) Use the Date Object to get the current date.
2) The format should be `HH:MM:SS`. if the time is 3 hours 12 minutes and 2 seconds, your clock should display `03:12:02`.
3) Closures will come in handy.