**Lab: JavaScript basics**

The purpose of this lab is to study the concepts discussed during [INF203 on Web technologies](http://perso.enst.fr/dufourd/cours/inf203/) and in particular JavaScript. The goal of this lab is to program in JavaScript without using the browser. Another lab session will concern JavaScript in the browsers.

This lab session is graded automatically and the grading tool is available so that you can measure where you are. See at the end.

**Additional useful information**

Additional useful information can be found in: [Labs Overview](https://perso.telecom-paristech.fr/dufourd/cours/inf203/labs-overview.html)

**Assignment**

For this lab, you will need to upload the different JavaScript files products. Use the “strict” mode of JavaScript, indentation and comments.

***Thank you for using zip (and not tar and gzip or bzip2). Zip the files directly, do not zip the folder.***

***No spaces or accented characters or special characters in the name of the zip file nor in folder names.***

***This assignment is to be done individually.***

**Preamble - Using Node JS**

JavaScript is a “complete” programming language. You can program in JavaScript everything you could do with any other programming language (Python, Java, C, …).

[NodeJS](https://nodejs.org/en/) is a runtime environment of JavaScript code. NodeJS uses the same JavaScript engine as Google Chromium. It can be used on the server side to generate web pages. It can also be used on the command line (like python for example) to run a program. That’s what we will do.

On machines in the TPT lab rooms, you can use it as follows: the > symbol at the beginning of the line refers to the NodeJS prompt, i.e. where NodeJS waits until you input code and press Enter, and the $ symbol refers to the Unix command prompt; the other lines are the results of the execution.

$ node

> console.log ('Hello World');

Hello World

undefined

>

The first line displays the desired result. The second displays the return value of the console.log function which is undefined.

This way of using NodeJS is very convenient for testing short programs, one line. This becomes more complicated for several lines. For this purpose, NodeJS accepts the name of a file as parameter. For example, if the hello.js file contains:

console.log ('Hello World');

The use of this file is simply:

$ node hello.js

Hello World

$

NodeJS also offers the possibility to debug your code, set breakpoints, inspect variables and stack, etc. To use the debugger, run node with the command:

$ node debug hello.js

The debug mode documentation is located [here](https://nodejs.org/api/debugger.html).

At the top of each file, put "use strict"; as a first line

If you have messages like “SyntaxError: Block-scoped declarations (let, const, function, class) not yet supported outside strict mode” then the “use strict”; at the top of your file is missing.

**Using node.js on the Télécom computers**

On the Télécom computers, there is a very old version of node.js which does not treat modules well.

The easiest solution to use a modern node is to install nvm.

Copy this and paste it in a terminal on one of the Télécom computers:

curl https://raw.githubusercontent.com/creationix/nvm/master/install.sh | bash

Possibly, execute what is asked at the end of the previous process. Then run:

nvm install node

This will install the latest version, compatible with modules (\*.mjs files).

Doing this once on one of the Télécom computers will be enough while you are using the same account.

**Additional useful information**

Additional useful information can be found in: [Labs Overview](https://perso.telecom-paristech.fr/dufourd/cours/inf203/labs-overview.html)

**Exercise 1 - Numbers, loops, arrays, functions and recursion**

Save the results of this exercise to a file named exercise1.mjs.

Here is a template of exercise1.mjs:

"use strict";

// no recursion

export function fibo\_it(n) {}

// recursive function

export function fibonaRec(n) {}

// no map function

export function fibo\_arr(t) {}

// use of map

export function fibMap(t) {}

To test the above, use another file like this:

"use strict";

import {fibo\_it,fibonaRec,fibo\_arr,fibMap} from "./exercise1.mjs";

console.log(fibo\_it(7)); // do more that one test per function

console.log(fibonaRec(8));

console.log(fibo\_arr([3,5]));

//...

Question 1a: Write a function named fibo\_it which calculates the nth number of the Fibonacci sequence iteratively (with a for-loop or while). fib(0) = 0, fib(1) = 1, fib(n) = fib(n-1) + fib(n-2)

Question 1b: Write a function named fibonaRec which calculates the Fibonacci sequence recursively.

Question 1c: Write a function fibo\_arr that takes an array of numbers and returns the array of results of fibonaRec called on the numbers without using the JS function map. You may reuse functions from previous questions.

Question 1d: Write a function named fibMap equivalent to the function from Question 1c using the JS function map.

In front of each of the functions of exercise1.mjs add the word export to allow automatic grading.

**Exercise 2 - Strings, Objects, Anonymous Functions**

Save the results of this exercise to a file named exercise2.mjs.

Question 2a: Write a wordc function which, for each word within a string, counts the number of occurrences of this word in this string. The function shall return an object whose properties are the words and the values of these properties are the occurrence numbers. Make sure this function works on a string of at least 500 words. The input string is assumed to contain no punctuation and only small caps.

Example: on the string “fish bowl fish bowl fish”, the result is {fish: 3, bowl:2}

Question 2b: Create a class WordL with a constructor which takes as input a string and that returns an object with the following methods:

* getWords() which returns an array of words present in the original text, lexicographically sorted and without duplicates,
* maxCountWord() which returns the word with the most occurrences, and if there are several words with the same number of occurences, returns the first of them in the lexicographically sorted list from getWords()
* minCountWord() which returns the word with the least number of occurrences, and if there are several words with the same number of occurences, returns the first of them in the lexicographically sorted list from getWords()
* and getCount(word) which gives the number of occurrences for a given word.

Question 2c: Add to WordL an applyWordFunc(f) method to apply any function to each word in lexicographic order and to return an array of results, for example:

function f (word) {return word.length;}

wordList.applyWordFunc(f) then returns the array of lengths of words

In front of the function wordc and class WordL, add the word export to allow automatic grading.

**Exercise 3 - Object-Oriented Programming in JavaScript**

Save the results of this exercise to a file named exercise3.mjs

Question 3a: Create a Stud class with attributes named lastName, firstName and id in that order, so that one can write the following code:

var student = new Stud("Dupond", "John", 1835);

Question 3b: Add a toString method that does not take a parameter, and which returns a string of characters built from the properties of the object in the form:

"student: Dupond, Jean, 1835"

Question 3c: Create a class ForStudent extending Stud that also allows to give a nationality to a student, so the arguments of the constructor are lastName, firstName, id and nationality. Add a toString method that takes the result of the basic class and adds the nationality, with a field of name nationality as follows:

"student: Doe, John, 432, American"

In front of each of the classes, add the word export to allow automatic grading.

**Exercise 4 - Module in JavaScript**

Save the results of this exercise to a file named exercise4.mjs.

Reuse the classes of Exercise 3 by writing:

import {Stud, ForStudent} from "./exercise3.mjs";

In the console, execute the command: npm install fs

This installs the standard module fs to read and write files.

Look for the fs module documentation to get an idea of the content of this module and how you will be able to use it in the next exercise.

In this fs module, there are many asynchronous functions. This lab is not about learning asynchronous programming so I do not recommend the use of asynchronous functions in fs. Using functions ending in “Sync” (readFileSync, writeFileSync…) will simplify your code.

Question 4: Create a module for a Prmtn class. An object of the Prmtn class may contain any number of Stud or ForStudent objects and have following methods:

* add(student) which adds a student to the promotion,
* size() which returns the number of students in the promotion,
* get(i) who returns the i-th Stud in the promotion in the order where you did add. Note: the index of the first student is 0
* print() that prints all students to the console, one per line, and returns the printed string
* write() which serializes the promotion to JSON, in other words transforms a promotion object in a string of characters,
* read(str) that reads a JSON object and rebuilds a promotion, WARNING: going through JSON.stringify then JSON.parse looses the fact that the object was a new Stud…
* saveToFile(fileName) that writes a promotion to a text file as a JSON object (reuse the above function write)
* readFromFile(fileName) which recreates a promotion from what has been saved to a file (reuse the above function read)

The functions write and read serialize and deserialize the object. Please use JSON.stringify to serialize and JSON.parse to deserialize.

***Note: The grader actually assumes the Prmtn is an array of student objects.***

In front of the class, add the word export to allow automatic grading.