Javascript Lab Seminar

MELL-JAVASCRIPT-00

Day 00

Recursivity

v1.61

Day 00

Recursivity

repository name: javascript_lab

branch name: day_00

Your repository must contain the totality of your source files.

You must have one file per Task. my_compute_factorial_it is the task one so you need to have a file named `my_compute_factorial_it.js`.

You are only allow to use **var let const if while** and **for**
Do not use any function of any kind that is not your. If you need to use concact() method then create it.

If one of your files prevents you from compiling and if we are not able to correct your work you will receive a 0.

All of the day's functions must produce an answer in under 2 seconds. Overflows must be handled (as errors).

Here's a complete list of the packages we'll use specifically for developing on the command line:

- chalk — colorizes the output
- clear — clears the terminal screen
- clui — draws command-line tables, gauges and spinners
- figlet — creates ASCII art from text
- inquirer — creates interactive command-line user interface
- minimist — parses argument options
- configstore — easily loads and saves config without you having to think about where and how.

Task 01

my_compute_factorial_it

Write an iterative function that returns the factorial of the number given as a parameter. It must be prototyped the following way:

```
int my_compute_factorial_it ( int nb) ;
```

In case of error, the function should return 0.

Delivery: ./my_compute_factorial_it.js 3

in that case, the exepected output is 6

```
0! = 1

if n < 0, n! = 0
```

Task 02

my_compute_factorial_rec

Write a recursive function that returns the factorial of the number given as a parameter. It must be prototyped the following way:

```
int my_compute_factorial_rec ( int nb)
```

In case of error, the function should return 0.

Delivery: ./my_compute_factorial_rec.js

Task 03

my_compute_power_it

Write an iterative function that returns the first argument raised to the power p, where p is the second argument. It must be prototyped the following way:

```
int my_compute_power_it( int nb, int p) ;
```

Delivery: ./my_compute_power_it.js

n_0 = 1 ifp<0,n_p =0

Task 04

my_compute_power_rec

Write an recursive function that returns the first argument raised to the power p, where p is the second argument. It must be prototyped the following way:

```
int my_compute_power_rec( int nb, int p) ;
```

Delivery: ./my_compute_power_rec.js

Task 05

my_compute_square_root

Write a function that returns the square root (if it is a whole number) of the number given as argument. If the square root is not a whole number, the function should return 0.

It must be prototyped the following way:

```
int my_compute_square_root( int nb)
```

Delivery: ./my_compute_square_root.js

Task 06

my_is_prime

Write a function that returns **1** if the number is prime and **0** if not. It must be prototyped the following way:

```
int my_is_prime( int nb)
```

Delivery: ./my_is_prime.js

As you know, 0 and 1 are not prime numbers.

Task 07

my_find_prime_sup

Write a function that returns the smallest prime number that is greater than, or equal to, the number given as a parameter.

It must be prototyped the following way:

```
int my_find_prime_sup( int nb)
```

Delivery: ./my_find_prime_sup.js

Task 08

The n queens

Write a function that returns the number of possible ways to place n queens on a nxn chessboard without them being able to run into each other in a single move.

It must be prototyped the following way:

```
int count_valid_queens_placements ( int n)
```

The output must be as follows:

```
$> ./count_valid_queens_placements 1
1

$> ./count_valid_queens_placements 2
0

$> ./count_valid_queens_placements 3
0

$> ./count_valid_queens_placements 4
2

$> ./count_valid_queens_placements 5
10
```

Delivery: ./count_valid_queens_placements.js

Google **the n queens problem**

Damn it, this is recursion day!