T - Web Development Seminar

T-WEB-x00

Day 06

Javascript

1.3

{EPITECH.}

Day 06

repository name: web_seminar_day06_$ACADEMICYEAR
repository rights: ramassage-tek
language: javascript

- Your repository must contain the totality of your source files, but no useless files (binary, temp files, obj files,...).
- All the bonus files (including a potential specific Makefile) should be in a directory named *bonus*.

During the days to come, you will have a series of exercises to get familiar with the JavaScript language.

The exercises are independent; therefore, you can solve them in the order that best suits you, but we recommend you to do it in the proposed order.

We will use node to launch each of your JavaScript files.

EXERCISE 01 — 2PTS

Turn in: exercise01.js

Draw me a triangle

Write a function *drawTriangle* that takes a height as parameter and draws a triangle on the standard output.
Obviously the height corresponds to the triangle height (see below).
The function must be exportable, and in a *exercise01.js* file.

Prototype: drawTriangle(height)

Terminal

```
~/T-WEB-x00> cat exercise01.js
module.exports = {
    drawTriangle: function(limit)  {
        //
        // your code here
        //
    }
}
```

Your function will be tested the following way:

Terminal

```
~/T-WEB-x00> cat exercise01_tester.js
const exercise01 = require('./exercise01');

exercise01.drawTriangle(6);
~/T-WEB-x00> node exercise01_tester.js
$
$$
$$$
$$$$
$$$$$
$$$$$$
```

EXERCISE 02 — 2PTS

Turn in: exercise01.js

An array and another array

Write a function named *arraysAreEqual* that returns true if both arrays passed as parameters are equal, false otherwise.
The function must be exportable, and in a *exercise02.js* file.

Prototype: arraysAreEqual(arr1, arr2)

Terminal

```
~/T-WEB-x00> cat exercise02.js
module.exports = {
    arraysAreEqual: function(arr1, arr2)  {
        //
        // your code here
        //
    }
}
```

Your function will be tested the following way:

Terminal

```
~/T-WEB-x00> cat exercise02_tester.js
const exercise02 = require('./exercise02');

exercise02.arraysAreEqual([1, 2], [1, 4]);
~/T-WEB-x00> node exercise02_tester.js
False
```

EXERCISE 03 — 2PTS

Turn in: exercise03.js

Keep good Count

Write a *countGs* function that takes a string as parameter and returns the number of uppercase 'G' characters it contains.
The function must be exportable, and in a *exercise03.js* file.

Prototype: countGs(str)

EXERCISE 04 — 3PTS

Turn in: exercise04.js

Baby steps in the dojo arena

Write a *fizzBuzz* function that takes a number as parameter and prints all the numbers from 1 to this number.
Three requirements:

1. For numbers divisible by 3, print "Fizz" instead of the number.
2. For numbers divisible by 5 (and not 3), print "Buzz" instead of the number.
3. For numbers that are divisible by both 3 and 5 print "FizzBuzz".

The function must be exportable, and in a *exercise04.js* file.

Every ouutput (be it a number or a string) should be comma separated.

Prototype: fizzBuzz(num)

Terminal

```
~/T-WEB-x00> node exercise04_tester.js 20 | cat -e
1, 2, Fizz, 4, Buzz, Fizz, 7, 8, Fizz, Buzz, 11, Fizz, 13, 14, FizzBuzz, 16, 17,
Fizz, 19, Buzz$
```

EXERCISE 05 — 4PTS

Turn in: exercise05.js
Restriction: only ES5 is allowed

A good range

Write a *range* function that takes three arguments (start, end and step) and returns an array containing all the numbers from start up to (and including) end.
The third argument indicating the step value used to build up the array is optional. If not provided, the array elements go up by increments of one, corresponding to the classic stepping behavior.
The function must be exportable, and in a *exercise05.js* file.

Make sure it also works with negative step values so that range.

Prototype: range(start, end, step)

Terminal

```
~/T-WEB-x00> cat example.js
// ...
console.log(range(1, 10, 2));
console.log(range(19, 22));
console.log(range(5, 2, -1));
~/T-WEB-x00> node example.js
[1, 3, 5, 7, 9]
[19, 20, 21, 22]
[5, 4, 3, 2]
```

EXERCISE 06 — 3PTS

Turn in: exercise06.js

Equality is complex

Write an *objectsDeeplyEqual* function that takes two values and returns true only if they are the same value or are objects with the same properties whose values are also equal when compared with a recursive call to *objectsDeeplyEqual*.

The word is out: you will be using recursion.

Your function should find out whether to compare two things by identity or by looking at their properties.

'null' is also an "object".

Your function is not supposed to be too complex, keep in mind that this is only the first day.
The function must be exportable, and in a *exercise06.js* file.

Prototype: objectsDeeplyEqual(cmp1, cmp2)

Terminal

```
~/T-WEB-x00> cat example.js
// ...
var obj = {here: {is: "an"}, object: 2};
console.log(objectsDeeplyEqual(obj, obj));
console.log(objectsDeeplyEqual(obj, {here: 1, object: 2}));
console.log(objectsDeeplyEqual(obj, {here: {is: "an"}, object: 2}));
~/T-WEB-x00> node objectsDeeplyEqual.js
true
false
true
```

EXERCISE 07 — 4PTS

Turn in: exercise07.js

Iterating is key

Write a *arrayFiltering* function that takes two arguments : an array and a *test* function.
The argument named *test* is a function that returns a boolean.
You dont have to care about this function implementation.

This function must be called for each element contained in the array given as parameter.
The return values determine whether an element is included in the returned array or not (if the test succeed, it should be included in the returning array).

The *arrayFiltering* function returns a new array containing filtered values.
The function must be exportable, and in a *exercise07.js* file.

Prototype: arrayFiltering(array, test)

Terminal

```
~/T-WEB-x00> cat example.js
// ...
var toFilter = [1, 2, 3, 4, 5, 6, 7, 8, 9];
var res = arrayFiltering(toFilter, function (value) {
    return value % 3 === 0;
});
console.log(res);
~/T-WEB-x00> node example
[3,6,9]
```