Chap 3 - JS

In this chapter, you will find a series of exercises dedicated to JavaScript (JS) to explore the main features of this programming language.

> To solve the problems, it is advised - before coming to the practicals - to read the five first chapters of the JS guide from MDN.

All the documentation of JS is available in this reference from MDN.

1. Configuration

1.1. Skeleton of a HTML page

Create a file `index_chap3.html` and copy and paste this basic HTML page entitled `Playing with JS`..

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Playing with JS</title>
<script type="text/javascript" src="my_code.js">
</head>
<body>
<header>
<nav>
</nav>
</header>
<section>
<article>
</article>
</section>
</body>
</html>
```

Then, create a file `my_code.js` in the same folder as `index_chap3.html`.

2. First steps with JS

In this section, you will explore the main features of JS.

2.1. var, if, for, and while…

Exercise 2.1 [★☆☆☆☆]: : Write the function `hashtags(num)` displaying a triangle of hash symbols in the browser console. `num` is the maximum number of hash symbols in the last line.

```
hashtags(10);
/*
Result in the console:
#
##
###
####
#####
######
```

```
#######
########
#########
##########
*/
```

> **Note #1:** Use a loop `for` .
> **Note #2:** To print in the console, use `console.log()` method.

Exercise 2.2 [★☆☆☆☆]: : Write the function `min(a,b)` returning the minimum value between `a` and `b` .

```
var i = min(0,10); // ← 0
var j = min(0,-10); // ← -10
```

> **Note**: Use the conditional statement

Exercise 2.3 [★☆☆☆☆]: : Write the function `countBs(txt)` taking as argument a string and counting the number of uppercase characters (or letters) **B** .

```
var i = countBs('BARBAPAPA'); // ← 2
```

> **Note**: Rather than using the array notation, you can use the String method `charAt(N)` . See the Mozilla documentation.

Exercise 2.4 [★☆☆☆☆]: : Write the function `countChars(txt,char)` taking as arguments a string and the character that is to be counted. This function returns the number of characters (or letters) .

```
console.log(countChars('BARBAPAPA','B') ); // ← 2
console.log(countChars('BARBAPAPA','A') ); // ← 4
```

> **Note**: Rather than using the array notation, you can use the String method `charAt(N)` . See the Mozilla documentation.

Exercise 2.5 [★★☆☆☆]: : Write the function `twoPowerOf(exponent)` calculating the result of $2^{exponent}$. For this implementation, use a loop `while` .

```
var result = twoPowerOf(10); // ← 1024
```

Exercise 2.6 [★★☆☆☆]: : Write the function `buzz(num)` displaying with `console.log(..)` the numbers from 1 to `num` except for numbers divisible by 3 replaced by *Fizz* and those divisible by 5 (and not 3) by *Buzz*. The numbers divisible by 3 AND 5 must trigger the display of *FizzBuzz*.

```
buzz(10);
/*
1
2
Fizz
4
Buzz
Fizz
7
8
9
Buzz
*/
```

Exercise 2.7 [★★★☆☆]: : Write the function `chessboard()` displaying a chessboard where the dark and light squares are displayed as '▪' (unicode hexadecimal 25A0) and '▢' (unicode hexadecimal 25A2) characters, respectively.

> **Note**: The algorithm is: if the sum of the square coordinates is even then the square is light otherwise it is dark. The top left square has the coordinates (0,0) and the bottom right (7,7).

Here is an example:

```
chessboard();
/*
The display in the console is:
▢ ■ ▢ ■ ▢ ■ ▢ ■
■ ▢ ■ ▢ ■ ▢ ■ ▢
▢ ■ ▢ ■ ▢ ■ ▢ ■
■ ▢ ■ ▢ ■ ▢ ■ ▢
▢ ■ ▢ ■ ▢ ■ ▢ ■
■ ▢ ■ ▢ ■ ▢ ■ ▢
*/
```

> **Note:** Use nested loops `for` (inner and outer loops).

1.2. JS5 *vs* ES6

Exercise 2.8 [★★★☆☆]: : Write the function `rangeES6(start, end, step)` that returns an array containing numbers from `start` up to, and including `end`. The argument `step` is optional and its default value is 1. The first and second arguments `start` and `end` are mandatory.

> **Note**: Implement this function in ECMAScript 6.

```
console.log(rangeES6(1,10) ); // ← [1,2,3,4,5,6,7,8,9,10]
console.log(rangeES6(4,10,2) ); // ← [4,6,8,10]
```

You can improve your function to take into account negative `step` like this…

```
console.log(rangeES6(10,4,-1) ); // ← [10,9,8,7,6,5,4]
```

Exercise 2.9 [★★★☆☆]: : From the previous function `rangeES6(start, end, step)`, write the function `rangeJS5(start, end, step)`. See details above.

> **Note**: Implement this function in JavaScript 5. The only difference with the previous one is the processing of optional argument `step`.

```
console.log(rangeJS5(0,10) ); // ← [0,1,2,3,4,5,6,7,8,9,10]
console.log(rangeJS5(4,10,1) ); // ← [4,5,6,7,8,9,10]
```

1.3. Exotic operators

Exercise 2.10 [★★★☆☆]: : Write the function `fromRGB(red,green,blue)` converting a red, green, and blue color into the corresponding hexadecimal notation used in HTML.

```
var rgb = fromRGB(255, 142, 22);
console.log(rgb); // ← '#FF8E16'
```

> **Note**: The method `toString(..)` is useful.

Exercise 2.11 [★★★★☆]: : Write the function `toRGB(hexa_color)` converting a hexadecimal notation of RGB color used in HTML to an object with 3 properties: red, green, and blue.

```
var rgb = toRGB('#FF8E16');
console.log(rgb); // ← {red: 255, green: 142, blue: 22}
```

> **Note**: For this implementation, convert the string into a number (24bits = 8bits + 8bits + 8bits) and then, use the *unary operators* of JS to extract from this 24-bit number, the red, green, blue 8-bit numbers.

1.4. Advanced programming concepts

Exercise 2.12 [★★★★☆]: : Write the function `toChain(array)` converting an array into a series of chaining objects. Each object has two properties `value` and `next` pointing to the next object in the chain as shown in Fig. 1.

Fig: 1: Resulting chain of objects.

```
var tree = toChain([1,2,3]);
// ← {value:1, next:{value:2,next:{value:3,next: null}}}
```

Exercise 2.13 [★★★★★]: : Write a **recursive** function `isEven(n)` returning a boolean value depending if the number `n` is even of odd. Rather than using the remainder (or modulo) operator, this function must be implemented with the following algorithm:

- zero is even
- one is odd
- for any other number `n` , its evenness is the same as `n-2` .

> **Note:** Recursion is a programming technique to avoid the use of loops. In this case, the function is calling itself. Here is an example of recursion.

```
// Classical loop
for (var i=0; i < 10; i++) {
  console.log(i);
}

// Recursion
function print(i) {
  if (i >= 10) {
    return;
  }
  console.log(i);
  print(i+1); // Recursive call of print
}
// Run the recursive function call
print(0);
```

Exercise 2.14 [★★★★★]: : Write a **recursive** function `toChain2(array)` based on `toChain()` converting an array into a series of chaining objects. Each object has two properties `value` and `next`.

```
var tree = toChain2([15,25,35]);
// ← {value:15, next:{value:25,next:{value:35,next: null}}}
```

> **Note**: Use a private function `addNode(index)` declared within `toChain2(..)` and that takes an only argument corresponding to the array index. This private function returns an object `{value: ???, next : ???}`.