PRÁCTICA 2.05 Jugando con la sintaxis

Normas de entrega

- En cuanto al **código**:
  - en la **presentación interna**, importan los **comentarios**, la claridad del código, la significación de los nombres elegidos; todo esto debe permitir considerar al programa como **autodocumentado**. No será necesario explicar que es un `if` un `for` pero sí su funcionalidad. Hay que comentar las cosas destacadas y, sobre todo, las **funciones** y **clases** empleadas. La ausencia de comentarios será penalizada,
  - en la **presentación externa**, importan las leyendas aclaratorias, información en pantalla y avisos de ejecución que favorezcan el uso de la aplicación,
  - si no se especifica lo contrario, la información resultante del programa debe aparecer en la consola del navegador `console.log(información)`,
  - los ejercicios deben realizarse usando ***JavaScript* ES6** y usar el modo estricto (`use strict`) No se podrá utilizar *jQuery* ni cualquier otra biblioteca (si no se especifica lo contrario en el enunciado),
  - para el nombre de **variables**, **constantes** y **funciones** se utilizará *lowerCamelCase,*
- En cuanto a la **entrega** de los archivos que componen los ejercicios:
  - todos los ejercicios en **una carpeta** (creando las **subcarpetas** necesarias para documentación anexa como imágenes o estilos) cuyo nombre queda a discreción del discente,
  - el nombre de los ficheros necesarios para resolver el ejercicio será el número de ejercicio que contenga,
  - el código contendrá ejemplos de ejecución, si procede, y
  - la carpeta será comprimida en formato `ZIP` y será subida a **Aules** de forma puntual.

Ejercicio 1 - Descomponiendo el problema I: el inicio

Crea dos funciones que realicen la siguiente tarea:

- la **primera** debe llenar un *array* de forma aleatoria con nueve números comprendidos entre el 1 y el 9 y que no se repitan. Devolverá ese array.
- la **segunda** debe recibir un *array* con nueve números y comprobar si alguno se repite. Devolverá `true` si se repite o `false` si no se repite.

Ejercicio 2 - Descomponiendo el problema II: el avance

Crea otras dos funciones con idéntico comportamiento que las anteriores, pero en esta ocasión los *arrays* con los que trabajará serán bidimensionales:

- la **primera** creará un array bidimensional de tres por tres en el que ningún número debe repetirse. Devolverá ese array.
- la **segunda** recibe un array de tres por tres y tiene alguno de sus nueve números repetido. Esta función devolverá true si existe alguna repetición y false si no la tiene.

Para crear un *array* bidimensional en JavaScript debes usar un *array* de *arrays* de este modo:

```
var tablero=[[0, 0, 0], [0 , 0, 0], [0 ,0, 0]];
```

Ejercicio 3 - Descomponiendo el problema III: el sudoku

Con todas las funciones elaboradas en los dos ejercicios anteriores, haz un programa que sea capaz de comprobar que la solución a un sudoku es correcta. Puedes generar funciones nuevas si así lo estimas oportuno. Esta función recibirá por parámetro el *array* con los valores del sudoku (uno de nueve por nueve) y devolverá **true** o **false** en función de si es correcto o no.

Ten presente que un sudoku es un array bidimensional de nueve filas por nueve columnas que a su vez se puede dividir en nueve *arrays* bidimensionales de tres por tres.

A continuación, dispones de un sudoku válido e invalido para hacer las pruebas. La explicación de las reglas del Sudoku se puede consultar **aquí**. Deberás comprobar que cumple la regla de las filas, de las columnas y opcionalmente los bloques de 3.

```
//Ejemplo de sudoku correcto
sudokuCorrecto = [];
sudokuCorrecto[0] = [1, 2, 3, 4, 5, 6, 7, 8, 9];
sudokuCorrecto[1] = [7, 8, 9, 1, 2, 3, 4, 5, 6];
sudokuCorrecto[2] = [4, 5, 6, 7, 8, 9, 1, 2, 3];
sudokuCorrecto[3] = [3, 1, 2, 6, 4, 5, 9, 7, 8];
sudokuCorrecto[4] = [9, 7, 8, 3, 1, 2, 6, 4, 5];
sudokuCorrecto[5] = [6, 4, 5, 9, 7, 8, 3, 1, 2];
sudokuCorrecto[6] = [2, 3, 1, 5, 6, 4, 8, 9, 7];
sudokuCorrecto[7] = [8, 9, 7, 2, 3, 1, 5, 6, 4];
sudokuCorrecto[8] = [5, 6, 4, 8, 9, 7, 2, 3, 1];
//Ejemplo de sudoku incorrecto
sudokuIncorrecto = [];
```

```
sudokuIncorrecto[0] = [2, 2, 3, 4, 5, 6, 7, 8, 9];
sudokuIncorrecto[1] = [7, 8, 9, 1, 2, 3, 4, 5, 6];
sudokuIncorrecto[2] = [4, 5, 6, 7, 8, 9, 1, 2, 3];
sudokuIncorrecto[3] = [3, 1, 2, 6, 4, 5, 9, 7, 8];
sudokuIncorrecto[4] = [9, 7, 8, 3, 1, 2, 6, 4, 5];
sudokuIncorrecto[5] = [6, 4, 5, 9, 7, 8, 3, 1, 2];
sudokuIncorrecto[6] = [2, 3, 1, 5, 6, 4, 8, 9, 7];
sudokuIncorrecto[7] = [8, 9, 7, 2, 3, 1, 5, 6, 4];
sudokuIncorrecto[8] = [5, 6, 4, 8, 9, 7, 2, 3, 1];
```

Ejercicio 4 - Buscaminas

Escribe una función que calcule la cantidad de minas adyacentes para el juego del Buscaminas. Puedes ver cómo funciona el juego **aquí**, y si no te queda claro puedes echar una partida **aquí**.

Esta función recibe un *array* bidimensional con la posición de las minas (valor **0** donde **NO** hay mina y valor **-1** donde **SÍ** hay mina). Con él, genera otro *array* bidimensional con la cantidad de minas que hay en las celdas adyacentes. En las celdas que hay una mina se guarda un valor **-1** (igual que en el *array* de entrada) y, en las que no, el número que indica las minas adyacentes.

Por ejemplo, recibiendo esta entrada:

```
[-1, 0, 0, 0]
[ 0, 0, 0, 0]
[ 0,-1, 0, 0]
[ 0, 0, 0, 0]
```

se produce esta salida por pantalla:

```
[-1, 1, 0, 0]
[ 2, 2, 1, 0]
[ 1,-1, 1, 0]
[ 1, 1, 1, 0]
```