UNIT 1

JAVASCRIPT

Part 2 - Exercise

Client-side Web Development
2nd course – DAW
IES San Vicente 2025/2026
Author: Arturo Bernal Mayordomo

Índice

JavaScript Exercise 2 3
Part 1 3
Part 2 5

JavaScript Exercise 2

In this exercise, we'll create two different sections where we'll work primarily with objects, classes, and collections. Download the users.js file and add it to the folder where you'll be doing the exercise.

Part 1

Create a file called **part1.js** and import **users array** from file **users.js**. Let's split the code into 4 different sections (it displays text in the console to indicate the start of each section). It's important to note that the original user array **should not be modified** at no time for any of the operations we carry out.

Section 1

Generate a new array containing the email and password details of the three youngest administrators. Try to perform this operation without generating any intermediate variables or constants (only the final result). Except for sorting the array, use JavaScript's iterator operations (obtain an iterator from the array).

Example of final result:

```
[
  { email: 'bob@example.com', password: 'secureBob' },
  { email: 'sophia@example.com', password: 'sophiaSecure' },
  { email: 'diana@example.com', password: 'D1ana#Admin_25' }
]
```

Section 2

Using iterators, obtain the names of the users who have access to PC1. Separately, obtain the names of the users with access to PC9. Using set methods, obtain the names of the users who have access to both PCs.

Display the three lists formatted with the JavaScript Internationalization API (INTL) in English. The output should look like this:

```
PC1: Peter, Bob, Charlie, Henry, Liam, and Ryan
PC9: Charlie, Grace, Henry, Noah, and Ryan
PC1 y PC9: Charlie, Henry, and Ryan
```

Section 3

Generate an array of users with more or less secure passwords. To do so, they must meet the following requirements:

- Minimum length 5
- At least one lowercase
- At least one capital letter
- At least one number
- At least one non-alphanumeric character (letter or number)

Afterwards, keep only the username and password of the users who meet it.

Obtain an iterator from the users array and apply the operations described above to the iterator.

This is the content of the final array:

```
[
  'Alice -> P@ssw0rd123!',
  'Diana -> D1ana#Admin_25',
  'Henry -> H3nry!_Secur3',
  'Liam -> L!am_P@ss_2025',
  'Peter2 -> P3t3r2-S#cur3',
  'Ryan -> Ry@n_Is_Str0ng!'
]
```

Section 4

Using this function which returns a random integer from zero up to a maximum (excluding):

```
function getRandomInt(max) {
  return Math.floor(Math.random() * max);
}
```

Get a user from the array at random and generate a new object with the same data as the user and an extra field called **lastAccess** with the date in a format like this (09/18/2025, 11:05:20)

Do not modify the original user! Use the operator **spread**. Print the user with the generated field and then print the original user to verify that it has not been modified.

Part 2

Create the file **robot.class.js** with the class **Robot** which will have:

- Two private attributes (model and battery) with their corresponding getters. The setter property will have a setter that checks whether the charge is between 0 and 100, or it will display an error and do nothing. The constructor will always set the battery to 100 (the model receives this).
- A charge method that will bring the battery to 100%.

Create a file **mobile-robot.class.js** with the class **MobileRobot** which will inherit from Robot and will have:

- An extra private attribute with the speed it reaches (speed) and its corresponding getter.
- A method **move** which will display the message (replace values): Moving *model* at *speed* km/h and will decrease the battery by 20%.
- A toString method that will display all the characteristics of the robot (including the robot type)

Create a file **flying-robot.class.js** with the class **FlyingRobot** which will inherit from Robot and will have:

- An extra private attribute with the height it reaches (altitude) and its corresponding getter.
- A method **fly** which will display the message (replace values): Flying *model* to altitude meters and will decrease the battery by 50%.
- A toString method that will display all the characteristics of the robot (including the robot type)

Finally create the file **part2.js** where we will implement the functionality of this section. Create an initial **array of robots**.

We are going to execute this section with **node**, so we'll take advantage of the built-in ability to read data from the console to interact with the user. To create the object that will read user input, we'll need the following code at the beginning and end of the file:

```
import * as readline from "node:readline/promises";
import { stdin as input, stdout as output } from "node:process";

const r1 = readline.createInterface({ input, output });
// The entire program
r1.close(); // Finally we close the input/output stream
```

We'll show the user a menu until they choose the exit option (0). The menu will look something like this:

```
--------------------------------
 MENU
--------------------------------
1) Show Mobile robots
2) Show flying robots
3) Create a robot
4) Move robots
5) Fly robots
6) Show robot info
0) Exit
```

To ask the user something and collect the answer we will use the following instruction:

```
const resp = await r1.question("Something to ask for: ");
```

Important: Do not create functions where you read from the console since readline works with promises and managing them with functions is something we have not seen yet (I will explain what **await** means later).

Here's what each option will do:

1. It will only show the information (toString) of the mobile robots
2. It will only show information about flying robots.
3. It will create a robot by asking for the model, type of robot, and, depending on the type, the value of its distinctive characteristic. We'll add it to the array.
4. Call the method **move()** of the robots that have it (mobile). Don't filter or check anything. Use the **optional concatenation operator** (?.).
5. Same as the previous one but with the method **fly()**
6. It will ask for a position in the array to display. It displays information about the selected robot (toString) or the message "Robot not found at position x" if there is no robot at that position. **Don't** use any kind of checks such as **if..else**. Instead use the **optional concatenation operator** (?.) and the **null coalescence operator** (??).

It's not enough that the results are what's required. The exercise MUST be performed in compliance with the guidelines indicated in each section, or it may be considered invalid. At the very least, you should try to comply with them as much as possible.