

JavaScript

Lab 1: Introduction

J. Pieck, L. Van Houdt, S. Van Peborgh, T. Van Nuland

1 PROJECT SET-UP 3

1.1 Overview: The Games Library 3

1.2 GitHub classroom assignment 3

1.3 Clone repository 3

1.4 Design the homepage 5

1.5 Committing and Pushing Your Changes 6

1.6 Statistics page 6

2 VARIABLES, FUNCTIONS, OBJECTS & CONDITIONALS 8

2.1 Creating objects 8

2.2 Displaying all objects 9

2.3 Display favorite games 11

2.4 Show the average rating of all your games 12

2.5 Conditionals 13

2.6 Conditional ternary operator 14

3 SUBMITTING YOUR RESULT 15

1 Project set-up

1.1 Overview: The Games Library

In this series of labs, you will create a front-end for an existing back-end application. This application allows you to manage your games library: add and remove games, give them ratings, and mark them as favorites.

Up to this point, you've learned how to create static websites using HTML and CSS. These skills remain essential for building the foundation of your front-end, but you will now use JavaScript to make your website dynamic. In each lesson, you will build a front-end application that consumes, processes, and displays data from a back-end application.

1.2 GitHub classroom assignment

For this project, we will use GitHub Classroom to host your source code. This platform automates the distribution of starter code and introduces you to the same workflow used in the industry. It also provides version control for your code, which acts as a reliable back-up.

The first step is to accept the assignment by clicking the link you can find on Toledo (under “Assignments” > “Module 2: JavaScript”).

1.3 Clone repository

Once you've accepted the assignment, GitHub Classroom will create a personal repository for you. You will then see a URL for this repository (you may need to refresh the page to see it). Copy this URL.

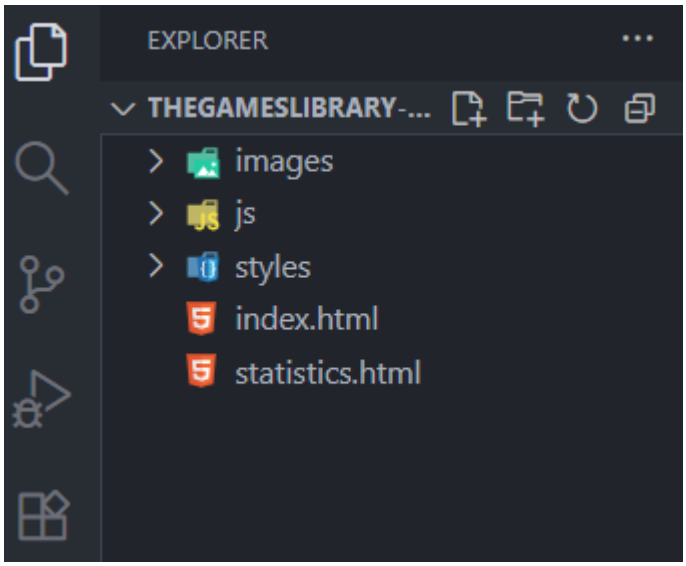
Next, open a terminal (e.g., Git Bash) in a local folder of your choice (e.g., C:\front-end) and run the following command:

```
git clone <URL_of_your_repo>
```

The URL of your repo looks like this:

```
https://github.com/UCLL-Frontend/the-games-library-{github_username}
```

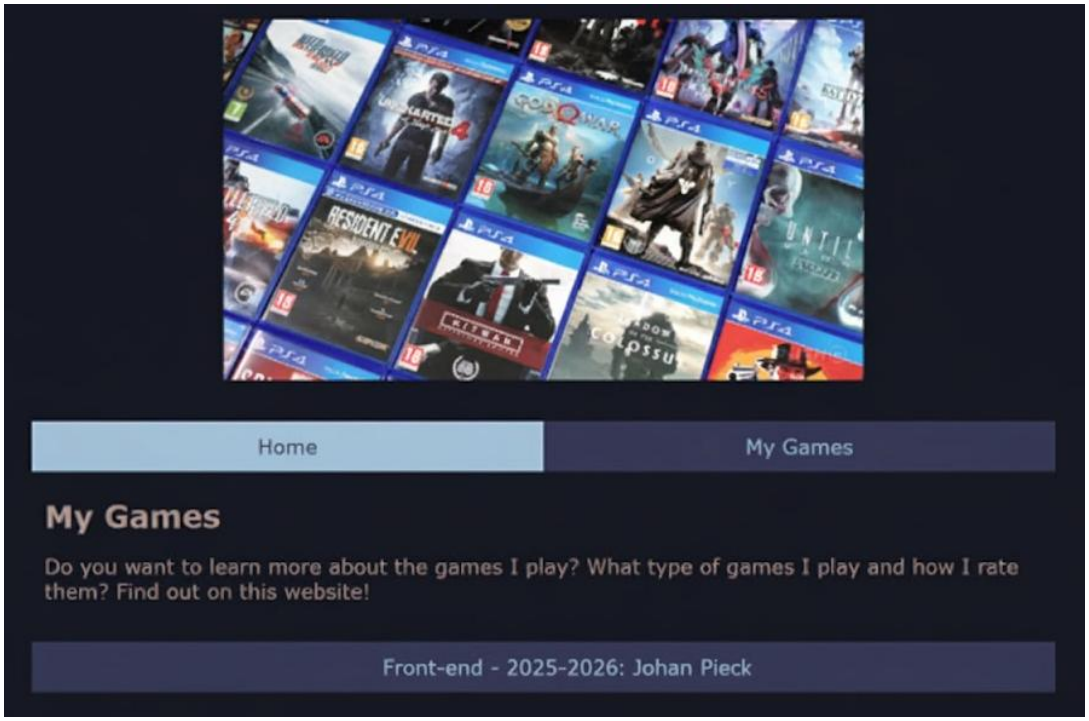
Open this folder in VSCode, a basic skeleton should already be available:



1.4 Design the homepage

Your first task is to build the static welcome page. We've already provided a basic HTML structure in `index.html`. When you preview this file in your browser, you'll notice it has a very basic layout. Open index.html in VSCode to inspect the code. We have included comments (between the `<!-- -->` tags) to guide you.

Use your creativity to build a polished and visually appealing homepage. Here is some inspiration:



Home page

- ❑ index.html is styled using CSS
- ❑ A logo is included in the header
- ❑ Your name appears in the footer

1.5 Committing and Pushing Your Changes

It's good practice to commit your work to GitHub regularly as you complete new parts of the assignment. This creates a reliable backup and a clear version history of your progress. To commit your changes, run the following commands from your working directory (e.g. C:/projects/the-games-library-johanpck):

```
git add .
git commit -m “Your commit message”
git push -u origin main
```

1.6 Statistics page

Open `statistics.html`. This page will serve as our practice area for the first few lessons. We'll use JavaScript to populate it with various game library statistics, like a complete collection overview, player averages, top-rated games, and more. Initially, our focus will be entirely on JavaScript functionality rather than a polished DOM structure. Later, once you are more comfortable with JavaScript, we will build a proper overview page that neatly organizes the games in a table.

Let's prepare `statistics.html` for this. First, create an empty div element with the id status and place it directly below the `h2` element. This div will be our target for displaying data from our script. Next, you need to include the JavaScript files so they load with the page. To do this, add the following `<script>` tag inside the `<head>` of your HTML:

```
<script src="js/dom.js" defer></script>
```



The defer attribute instructs the browser to execute the JavaScript file only after the entire HTML document has been fully loaded. This is crucial because it guarantees that all HTML elements are available in the DOM before your script attempts to access them.

You will write all your code in the js/games.js file. This script depends on helper functions provided in dom.js. For this reason, it is important that you always load dom.js before games.js.



Overview page

- ❑ Statistics.css is styled using CSS
- ❑ dom.js is included in the head
- ❑ games.js is included in the head, after dom.js
- ❑ A logo is included in the header
- ❑ Main element contains an empty div with id “status”
- ❑ Your name appears in the footer

2 Variables, Functions, Objects & Conditionals

In this lesson, you will learn how to declare and use variables, objects, functions, and conditionals. We will start by hardcoding data for several games and displaying this information, along with some basic statistics, on the `statistics.html` page. For now, we'll do this without grouping the games into an array.

If you're an experienced programmer, these initial exercises might seem rudimentary. However, we encourage you to complete them as a quick refresher. As we introduce more advanced JavaScript concepts in later lessons, we will build upon and refine this task.

Each exercise begins with a brief review of the relevant code from the “Farm” demo. This is followed by the exercise itself and a set of “Goals”. You should use these goals to verify that you have used the intended techniques, ensuring you are practicing the correct skills.

2.1 Creating objects

Objects are variables used to store a collection of related data. Think of an object as a single container where information is stored in `key-value` pairs, which we call `properties`. For example, an `animal` object could have properties like a `name`, a `type`, and an `age`.

In JavaScript, you define an object as follows:

```
const duck = { name: “Albert”, type: “duck”, age: 2 }
```

This code creates an object named `animal1`. This object contains three properties: `name`, `type`, and `age`, where each property is a key-value pair. Here, the key `name` has the value “Albert”, the key `type` has the value “duck”, and the key `age` has the value 2.



Create 4 different game objects in the file “games.js” in the folder “js”. A game has a name, is of a particular type and has a rating assigned.
If you are not familiar with gaming, here are some examples of popular games:

- Elden Ring, a fantasy game
- Horizon Forbidden West, an adventure game
- Pokemon Legends: Arceus, an RPG game
- GTA V, an open world game



In games.js you should have:

- ❑ Four different constants, where each constant holds a single game object.
- ❑ Each game object must contain the following properties:
  - name: string
  - type: string
  - rating: number

2.2 Displaying all objects

The `addStatus` function, provided in `dom.js`, lets you display text on the page. It takes a single argument (the text you want to show) and automatically creates a new paragraph with that text inside the HTML element with `the id="status".`

```
const addStatus = (status) => { … }
```

To display the sentence “Name: Albert – Type: duck – Age: 2” on the screen, you can use:

```
addStatus(`Name: ${animal1.name} - Type: ${animal1.type} - Age:
${animal1.age}`)
```

Note the use of the backtick (`) instead of double (“) or single (‘) quotes.

You can also use the `addStatus` function to print headings on the page:

```
addStatus("<h1>My games</h1>");
```

This adds a new h1 element to the DOM.

Your goal is to display the four game objects you created earlier on the statistics.html page. You will accomplish this by creating two separate functions.

1. Create a formatting function: Write a function `gameToString` that takes a single game object as an argument. This function's job is to format the game's details into a single, readable string and then return that string.
2. Create a printing function: Write a second function named `printAllGames`. This function will call `gameToString` for each of your four game objects. For each result it gets back, it will then use the `addStatus()` function to display the information to `div#status`.

The result should look like this:

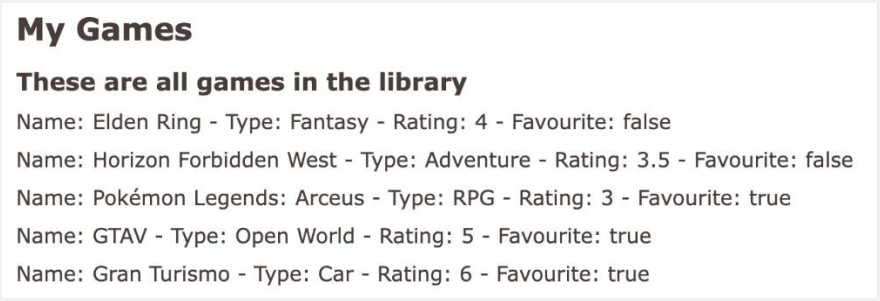


Goals: In games.js, you should have defined:

- ❑ a function printAllGames
- ❑ a function gameToString
- ❑ printAllGames should call addStatus and gameToString 4 times
- ❑ at the bottom you should execute the function printAllGames

2.3 Display favorite games

In this next step, we will add the ability to mark a game as a favorite. You will learn how to add a new boolean property to your objects and modify your existing functions to dynamically display this new status:



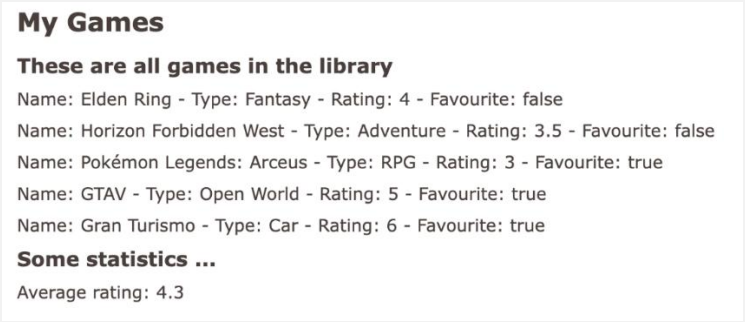
Goals: You should change the following in games.js

- ❑ Add a new property, isFavorite, to each of your existing game objects. This property should hold a boolean value (true or false).
- ❑ Create a new const for the game "Gran Turismo," complete with all its specific values (name, type, rating, and isFavorite).
- ❑ In printAllGames, you should only need to add one single line of code to ensure the new Gran Turismo game is also displayed.
- ❑ Update gameToString, so it also includes the isFavorite status in the output string, indicating whether or not a game is a favorite.

2.4 Show the average rating of all your games

Your next task is to compute and display the average rating for all the games in your collection. First, create a function - let's call it getAverageRating - that sums the rating property of all your game objects and divides it by the total number of games. This function should return the final average value.

Next, call this function and use its return value with the existing addStatus() helper to display the result on the page. The result should look like this:



Goals: Your games.js file should meet these criteria:

- ❑ You must have a function named getAverageRating
- ❑ You need to make two calls to the addStatus() function:
  - One call to display a title for this section
  - A second call to display the average rating

2.5 Conditionals

An if-else statement is a conditional that controls the flow of your program. It executes one block of code if a specific condition is true, and a different block if that condition is false. This gives your code the ability to make decisions and react differently to various situations.

```
let result = null
if (first.age > second.age) {
  result = first
} else {
  result = second
}
return result;}
```

Add the game with the highest rating to the statistics. Create a function getHighestRating() that returns the name and rating of that game.

The result should look like this:

My Games

These are all games in the library

Name: Elden Ring - Type: Fantasy - Rating: 4 - Favourite: false
Name: Horizon Forbidden West - Type: Adventure - Rating: 3.5 - Favourite: false
Name: Pokémon Legends: Arceus - Type: RPG - Rating: 3 - Favourite: true
Name: GTAV - Type: Open World - Rating: 5 - Favourite: true
Name: Gran Turismo - Type: Car - Rating: 6 - Favourite: true

Some statistics ...

Average rating: 4.3
Gran Turismo is game with the highest rating: 6

Goals: Your games.js file should meet these criteria:

- ❑ You must have a function named getHighestRating
- ❑ Inside this function, you must use if statements to compare the ratings of each game to determine which is the highest.
- ❑ The function must return the entire game object that has the highest rating, not just the rating value itself.
- ❑ The final result should be displayed on the page with a single call to your getHighestRating function.

2.6 Conditional ternary operator

The ternary conditional operator provides a compact, one-line alternative to a standard if-else statement. Its syntax is:

```
condition ? valueIfTrue : valueIfFalse
```

For example, the Animals demo uses it to decide what to display. The code checks if a variable holding the oldest animal actually contains an object. If it does, it displays the animal's name; otherwise, it displays the message “Animals are equally old.”

```
const oldest = getOldestAnimal(animal1, animal2);
addStatus(`Oldest animal: ${oldest !== null ?
oldest.name : "Animals are equally old."}`);
```

Your task is to display a list of only your favorite games by name.

To achieve this, you will create two functions:

- A helper function, isFavorite(game), that takes a game object as an argument and returns a boolean.
- A function printFavoriteGames(), that checks each game using your isFavorite() function. If a game is a favorite, this function should display only its name on the page.

Add two new game objects to the very beginning of your games.js file.

The final result should look like this:

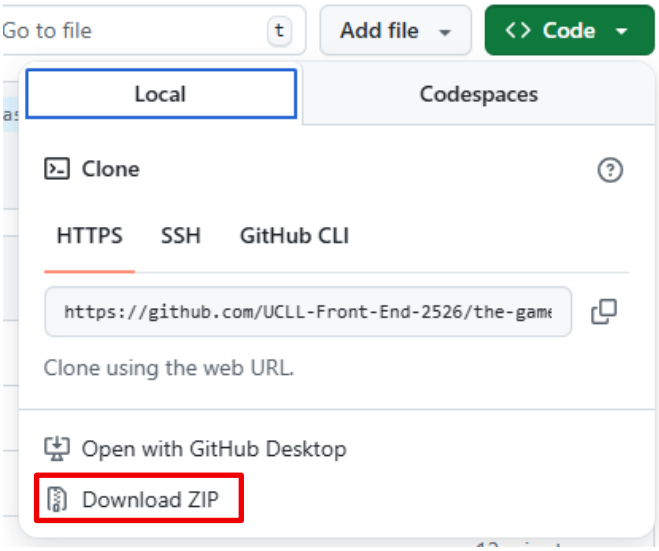


3 Submitting your result

To complete the lab, you must submit a ZIP file of your project via the assignment on Toledo. Important: Do not submit a link to your GitHub repository. Since you will continue to build on this same codebase in future labs, we need a "snapshot" of your code at this specific stage for grading.

The easiest way to create this snapshot is directly from GitHub:

- Navigate to your GitHub repository in your browser (e.g., https://github.com/UCLL-Front-End/the-games-library-jhnpck).
- Click the green "<> Code" button.
- Select Download ZIP from the dropdown menu.



Submit the downloaded ZIP file on Toledo to finalize your submission.