UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II
BACHELOR OF SCIENCE IN COMPUTER SCIENCE
WEB TECHNOLOGIES COURSE – SPRING 2025
TEACHER: Luigi Libero Lucio Starace, Ph.D.

DATE: March 27, 2025
BEST DONE BEFORE: April 3, 2025

ASSIGNMENT: #06 TOPIC: JAVASCRIPT IN A BROWSER ENV.

For this assignment, download and extract the provided `Assignment-06-materials.zip` archive. Serve the provided files in a HTTP server of your choice (you may use the one included with the Live Preview VS Code extension).

EXERCISE 1. TO-DO APP

In this exercise you will create a simple web app to help users keep track of a list of things to do. Open the provided **todo.html** document in your favourite text editor. As you'll see, the document includes an external **todo.js** script (currently empty). Your task in this exercise is to implement a basic To-do app with the following features:

- Users can add as many text to-dos as they want. They can add them by adding some text in the input field and by clicking on the dedicated "Add to the list" button. It must not be possible to add empty items (i.e., with no text) to the list.
- Users can mark an item as **done** by clicking on the item. By clicking again on an item that has been marked as **done**, users can mark them as **to-do** again. **Hint**: in the provided stylesheet, a `.done` class is defined for styling items that have been marked as completed. **Hint:** Once you get the to-do item on which the user clicked, you can use the toggle method on its `classList` property (i.e., `.classList.toggle("done")`) to easily toggle the done class and mark the item as completed.
- Users can delete items by double-clicking on them. **Hint**: the `dblclick` event is generated on double clicks.
- In addition to clicking on the dedicated "Add to the list" button, users can also add items by pressing the **Enter** key on their keyboard while typing. **Hint:** for this task, you can listen for the `keydown` event and check whether `event.key === "Enter"`. Since the input field is inside a form, and pressing Enter triggers form submission by default, you will also need to call `event.preventDefault()` to prevent a form submission which would reload the web page.

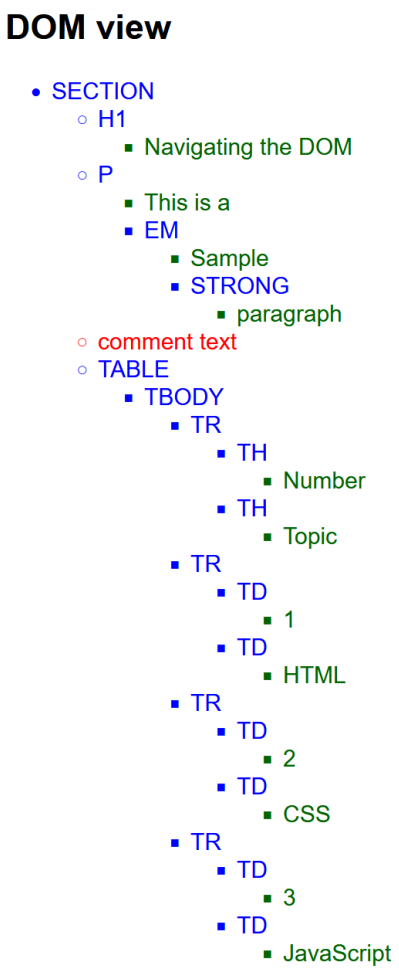
ASSIGNMENT: #06 TOPIC: JAVASCRIPT IN A BROWSER ENV.

EXERCISE 2. DOM NAVIGATION TRICKERY

Open the provided **dom.html** document in your favourite text editor. The document includes an external script named **dom.js** and calls a `renderDOM` function. This function represents part of the structure of the current document as a list, within the document itself! The function takes as input the **id** of an element whose DOM structure is to be represented in the current page (in the example, **`container`**), and the **id** of a list in which the DOM will be represented (in the example, **`dom`**).

Your task for this exercise is to implement the `renderDOM` function in the **dom.js** file. The list representation of the DOM your function will create should be as follows. HTMLElement nodes should be represented as `<li>` with the `.element` class. The text content of the `<li>` is the name of the tag itself. Child nodes are represented as nested lists in the `<li>`. Comment nodes are represented as `<li>` with the `.comment` class, and their text content is the commented text. Similarly, text nodes are represented as `<li>` with the `.text` class. Styles for these classes are defined in the HTML document. Last, text nodes containing only whitespaces and newlines should not be included in the representation. **Hint:** you can check if a `textNode` contains only whitespaces by checking whether `textNode.data.trim().length > 0`.

The expected output for the exercise is shown as follows.



UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II
BACHELOR OF SCIENCE IN COMPUTER SCIENCE
WEB TECHNOLOGIES COURSE – SPRING 2025
TEACHER: Luigi Libero Lucio Starace, Ph.D.

DATE: March 27, 2025
BEST DONE BEFORE: April 3, 2025

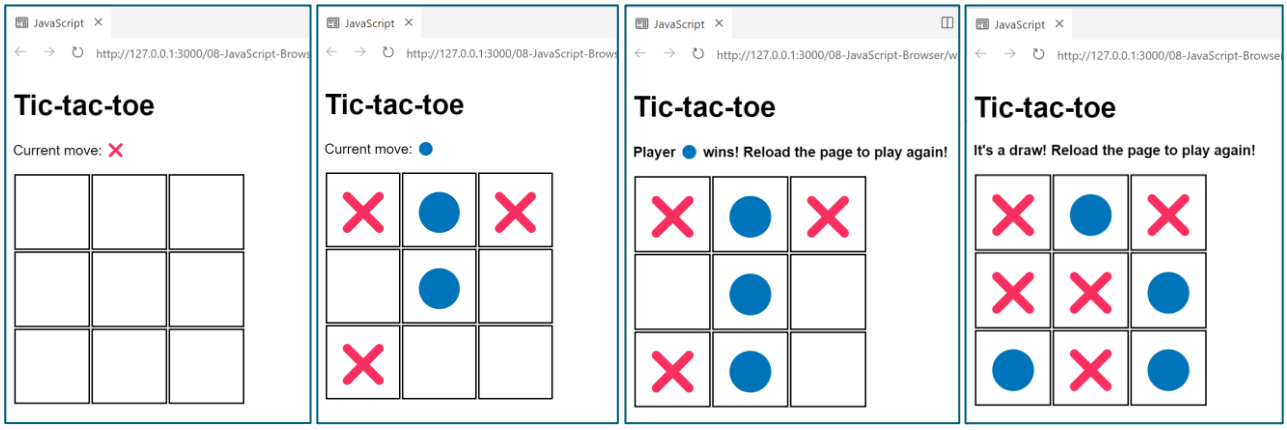
ASSIGNMENT: #06 TOPIC: JAVASCRIPT IN A BROWSER ENV.

EXERCISE 3. TIC–TAC–TOE

Open the provided **tic-tac-toe.html** file in your favourite text editor. As you'll see, the document includes an external **tic-tac-toe.js** script (currently empty). Your task in this exercise is to implement a fully functional tic-tac-toe game by writing the appropriate code in the **tic-tac-toe.js** script. For this task, you are not allowed to change the HTML code of the document.
Here are some notes and specifications:

- The `setupGame` function takes as input a string representing the **id** of the container element in which the game board will be rendered. You should start by implementing this function in the **tic-tac-toe.js** scripts.
- The board must be represented as a table containing three rows, each with three table cells. Such table should be dynamically created and inserted in the DOM, in the container specified to the `setupGame` function.
- For the sake of simplicity, you can use two emojis (❌ and 🏐) as the symbols of the two players.
- The first player to make a move is the one with the ❌ symbol. Subsequently, the players take turns at making moves. A move is made by clicking on an empty table cell. When a player makes a move, its symbol is placed in the selected table cell.
- During a match, the game shows an indication the player that must make the next move.
- A match can result either in victory by one of the two players (if they manage to put three of their symbols in a row, in a column, or in one of the diagonals), or in a draw (if there remain no more moves to make, and none of the player won). The game should show an indication as soon as one of the player wins, or the match ends in a draw.
- After one of the player wins, no player should be able to make additional moves.
- T here is no need to write more CSS in addition to the styles provided in the HTML document, but feel free to do so if want to further improve the looks of the game!

As follows, some screenshots show a possible implementation of the game.



ASSIGNMENT: #06 TOPIC: JAVASCRIPT IN A BROWSER ENV.

EXERCISE 4. SNAKES ON A TABLE (★)

*Enough is **enough**! I have **had** it with these MF'ing **snakes** on this MF'ing **table**!*

Samuel L. Jackson on this Web Technologies assignment

In this exercise, you are going to implement your own clone of the classic Snake game, using nothing but an HTML table, a few CSS rules, and a little JavaScript. Here are some requirements:

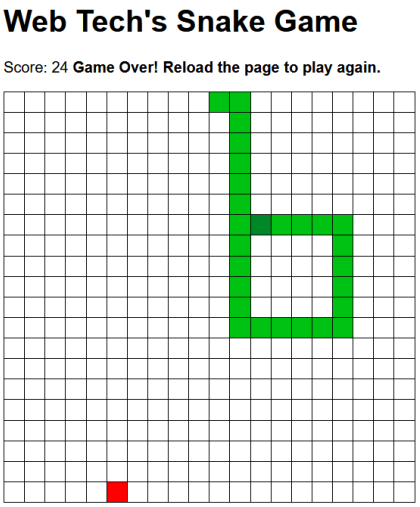
1. You should start from the provided **snake.html** file.
2. The game should be displayed within a **<table>** element (think of the table as a grid of pixels). Each table cell can be coloured using the CSS classes already provided in the HTML file (.snake-head, .snake-body, .food).
3. At the beginning of a game, the Snake starts at the center of the grid, and has a length of three cells (one snake-head, and two snake-body). Initially, the snake moves to the right.
4. During a game, food is spawned on the map in random positions. There can be at most one food item on the map at any given time. The food cannot spawn in cells which are already occupied by the Snake.
5. A new step in the game is computed every 100 milliseconds. Take a look at the **setInterval** global function for a simple way to run some code at specific time intervals.
6. Players can control the Snake using the directional arrows on their keyboard.
7. When the Snake eats a piece of food, its length increases by one (and the game also increases a score counter, which starts at 0).
8. If the Snake hits the borders of the map, or itself, during the game, the game ends and the player loses. If the Snake occupies all available space on the grid, the player wins.
9. This exercise may also be a nice way to practice with object-oriented JavaScript. You may define a "SnakeGame" class, including all methods to setup and manage a game. By providing different parameters to the SnakeGame constructor, it should be possible to create game maps of different sizes.

Here are some (optional) tips:

- Start by dynamically creating the game map (i.e., the table).
- Then, work on adding the necessary data structures (also in terms of auxiliary objects) to represent the state of the game at a given time step. You may use a dedicated Snake class to represent the Snake and its position on the map. The Snake class may also be responsible for updating the position of the Snake over time. You also need to keep track of where the food is.
- Next, create a rendering method to draw the map according to the current state of the game.
- Once you are done, you need to keep track of the current direction the snake is moving in, and implement a method that updates the current game state, computing the next state.
- Do not forget to check for lose/win conditions!

ASSIGNMENT: #06 TOPIC: JAVASCRIPT IN A BROWSER ENV.

When you are done, the game should look something like the picture provided as follows. If you want to, feel free to add additional features (e.g.: the possibility of pausing the game by pressing a specific key, or the possibility of specifying map sizes using dedicated inputs in the HTML document).



ASSIGNMENT: #06 TOPIC: JAVASCRIPT IN A BROWSER ENV.

EXERCISE 5. SIERPINSKI STRIKES BACK (★)

The Sierpiński triangle is a fascinating fractal pattern named after the Polish mathematician Wacław Sierpiński. It is a self-similar structure that emerges from a simple set of rules, creating an infinitely complex shape. The triangle is constructed by recursively subdividing a larger triangle into smaller congruent triangles, removing the central one, and repeating the process indefinitely. Despite its complexity, the Sierpiński triangle can be generated using a surprisingly simple algorithm called the chaos game.

In this assignment, you will use JavaScript and the HTML `<canvas>` element to draw a Sierpiński triangle one point at a time. By implementing the chaos game algorithm, you will witness how randomness and simple rules can create intricate and beautiful patterns.

What's a <canvas>?

The `<canvas>` element is an HTML component that allows you to draw dynamic and interactive graphics directly within a web page. Using JavaScript, you can draw shapes, lines, text, images, and even animations. The `<canvas>` is particularly useful for creating games, data visualizations, graphical effects, and much more. You can learn more about canvas elements in the dedicated MDN Canvas Documentation.

Instructions

Your task is to write a JavaScript program that draws a Sierpiński triangle on a **<canvas>** element. You will use the chaos game algorithm to generate the triangle one point at a time. Follow these steps:

1. **Check out the provided starter files (triangle.html and triangle.js)**. The HTML document contains a `<canvas>` element, with its width and height set to 600px each. The `<canvas>` element is also given an id, so we can easily refer to in our JavaScript code. The document also contains a heading with a title and a button which will be used to start drawing our fractal. The document also includes a `triangle.js` file, currently empty.

2. **Access the Canvas in JavaScript.** In the `triangle.js` file, use `document.getElementById` to get a reference to the `<canvas>` element. Use the `getContext('2d')` method to get the 2D rendering context. This is what you'll use to draw on the canvas. **Hint:** Store the canvas element in a variable (e.g., `canvas`). Then, use `getContext('2d')` to get the 2D drawing context and store it in another variable (e.g., `ctx`). To learn more about `getContext`, check out MDN getContext Documentation.

3. **Define the Triangle's Vertices**: The Sierpiński triangle is defined by three points (vertices). Create an array of objects to store these points. You may use the following coordinates for the vertices: A: (300, 100); B: (100, 500); C: (500, 500). **Hint**: You may represent vertices using an array with three objects, each containing x and y properties for the coordinates.

ASSIGNMENT: #06 TOPIC: JAVASCRIPT IN A BROWSER ENV.

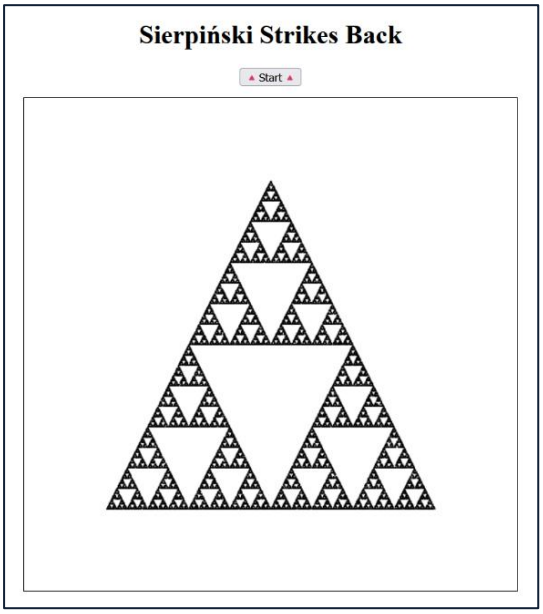
4. **Write a Function to Select a Random Vertex.** Create a function that randomly selects one of the three vertices. Use `Math.random()` and `Math.floor()` to pick a random index from the vertices array. **Hint**: Use `Math.random()` to generate a random number between 0 and 1, multiply it by the length of the vertices array, and use `Math.floor()` to round it down to the nearest integer. Learn more about `Math.random` on the MDN Math.random Documentation.

5. **Implement the Chaos Game Algorithm.** Create a function to implement the chaos game algorithm. Start with a random point within the canvas as the current point. You may get it using `Math.random()`, or you can statically select a point (e.g.: the point with coordinates (300, 300)). Then, use a loop (e.g., 50.000 iterations) to:

    - Select a random vertex using the function you defined earlier.
    - Calculate the midpoint between the current point and the selected vertex. **Hint**: the midpoint can be computed by averaging the x and y coordinates of the current point and the randomly selected vertex.
    - Draw a 1x1 pixel at the midpoint using `ctx.fillRect`. Learn more about `fillRect` at MDN fillRect Documentation.
    - Update the current point to the midpoint and go to next iteration.

6. **Implement logic to clear the canvas and start the drawing when the start button is pressed.** Use `document.getElementById` to get a reference to the button. Add an event listener to the button so that when it's clicked, the canvas is cleared and the function that implements the chaos game algorithm is called. **Hint**: You may use `ctx.clearRect` (MDN clearRect Documentation) to clear the canvas before calling the chaos game algorithm.

7. **Test Your Code.** Visit the triangle.html page you created and click the start button. If you've done everything correctly, you will see the Sierpiński triangle emerge from the canvas (as shown in the Figure on the right)!



8. **Experiment and Customize (Optional).** Change the number of iterations to see how it affects the triangle. Modify the color of the pixels using `ctx.fillStyle` (MDN fillStyle Documentation). Try changing the position of the vertices, or even adding more vertices to create different fractals!

9. **Introduce delays (Optional).** For a nicer effect, you may consider introducing a small delay when drawing each point, so that the fractal emerges more gradually. To this end, you may need to do some refactoring and use the `setInterval` function you used in **EXERCISE 4.**