Assignment: guess the number game

Instructions (general)

The logic of the game is the following:

- generate a random number between 1 and 100 and store it in a variable.
- ask the user for a number.
- compare this number to the random number you generated.
- inform the user whether the number to guess is smaller or bigger than the input.
- if the user correctly guesses the number, inform the user and stop the game.

Step-by-step instructions (basic version)

- Create a new folder for the project.
- Create a new HTML document.
- The contents of the HTML document can be empty for the moment, but make sure the structure is complete and correct.
- Create a new Javascript file. You can use the filename of your choice, but make sure you remember it for next step.
- Add the script to the HTML document. You have two options:
  - add it to the `<head>`
  - add it at the bottom of the `<body>`
  - if your file is called `script.js`, you can use `<script src="script.js"></script>`

In the script:

- Generate a random number between 0 and 100.
  - You can use `Math.random()` to generate a pseudo-random floating point number between 0 and 1.
  - Multiply this number by 100, and round it using `Math.floor()` to obtain an integer.
  - Store this number in a variable (use `let` to declare the variable).
- Ask the user for input using the prompt function: `userGuess = prompt("What is your guess?")`.
- Write the logic to compare this value with the random number.
  - Be careful! User inputs are provided as **strings**, when your generated number is a **number**.
  - If `text` is a string that is a number, you can use `parseInt(text)` to transform it to an integer.
  - If the text is not a number, `parseInt` returns `NaN` (**N**ot **A** **N**umber).
- Depending on the comparison, provide feedback to the user (the number is bigger / smaller / correct).
- You can use `alert` to generate a dialog box.
- Wrap the whole logic block above in an infinite loop.
  - You can run an infinite loop with `while(true) { ... }`.
  - Make sure you `break` out of the loop when the user guessed the correct number.

Make sure the game works as expected.

Add more features

1. Ask the user first for the maximum number that can be generated randomly.
2. When the player wins, use `confirm()` to ask whether to play another round. Try two versions:

- one where 1. happens at each round
- one where 1. only happens when the page is loaded first

3. Limit the number of attempts a player has in a round. You can use a fixed value (for example 10).

Improve the code

- Write a function that plays **A SINGLE TURN**:
  - `function playOne(correctNumber)`
  - `correctNumber` is the number to guess
  - the function asks the user for their guess and provides feedback **ONCE**
  - it returns `true` if the correct number was guessed, and `false` otherwise
  - refactor your code to use this function instead
- Write a function that plays the whole game:
  - `function play()`
  - this function generates the random number, and calls `playOne` as required
  - refactor your code to use this function instead

Practice with event listeners and the DOM

- Create a new HTML document, create a new empty `<div>`.
- Create three CSS rules / classes:
  - one class that sizes the element to 400px by 400px
  - one class that makes the background color red
  - one class that makes the background color green
- Provide your div with the two classes required to adjust its size and make its background color red
- Make sure your div has a distinct ID (we will use `box` in this example)
- Create a new Javascript file, and load it from your HTML document.
- **MAKE SURE YOUR SCRIPT FILE IS LOADED AT THE END OF THE `<body>`**

We are now going to use DOM traversal and event management to change the background color of the div when we click on it.

- `document.getElementById("box")` will provide you with the HTML node with id `box`.
- You can make changes to this element using different methods. For example:

```
let myBox = document.getElementById("box");
myBox.classList.add("red")
myBox.classList.remove("green")
myBox.innerHTML = "<p>Contents</p>"
myBox.textContent = "Hello world!"
```

- Write the logic that "toggles" between the two classes, red and green

💡 `myBox.classList.contains("red")` will return `true` if `red` is one of the classes of the element

- Add a new event listener that changes the background color of the box when you click on it.

💡 `document.getElementById("box").addEventListener("click", functionToCallOnClick)` will call the function `functionToCallOnClick` whenever the div is clicked.

- Wrap the toggling logic above in a function, and add it to the event listener.
- Check the behaviour of your script.

Improve the script to run the game only when the page is loaded

- Our script only works because it is loaded at the end of the `<body>`. At that stage, the div is already loaded in the DOM and the script can operate on it.
- If you move the script to the `<head>`, it will start executing immediately - at that stage, the div has not been parsed / loaded yet and the script cannot add an event listener to it.
- The solution is to run the script only when the DOM tree has loaded and is ready.
- There is an event for that, which is handled by the `document` itself.
- `document.addEventListener("DOMContentLoaded", functionToCallWhenReady)` will call the function when the document has finished loading in the browser.
- Move your script to the `<head>`, and make sure it still works as expected.

Anonymous functions

Instead of writing specific functions for our event handlers, in many cases we are going to use anonymous functions (= functions that are declared on the spot). For instance: `document.getElementById("box").addEventListener("click", function() { console.log("Click!"); })` will log a message to the console every time the element is clicked.

Guess the number: using the DOM tree

- Create a new HTML document.
- Create a new form, and add two input elements:
  - an input text box
  - a submit button
- Make sure your form, input and submit elements have a distinct ID.
- Below the form, create an empty `<div>` element. Make sure to give it an ID.

The elements above are the layout of the game:

- the user inputs their guess in the input box
- the user clicks the button or types Enter
- the web page provides feedback about the guess in the `<div>` below

Reuse the code from the first exercise and refactor it to use event listeners and DOM elements.

Notes

- If your text input has the ID `userGuess`, then `document.getElementById("userGuess").value` contains the current text entered into the box. This is going to be a **STRING** - you may have to convert it to a number with `parseInt` for further processing.
- Pressing Enter in the input box or clicking the Submit button will trigger a `submit` event on the associated **FORM ELEMENT**.
- The default behaviour when submitting a form is to send the form data to the server, and to load the result. We don't have a server, and we don't want to have anything else happen when we submit our form.
- The callback function for event listeners takes an optional argument: the event that triggered the call.
- You can call `.preventDefault()` on that event to prevent it from bubbling up and prevent the page from refreshing.

```
document.getElementById("form").addEventListener("submit", function(event) {
  event.preventDefault(); // will prevent the event from bubbling up and making the
browser refresh the page
});
```

- You only need this for events that have an unwanted default effect (most of the time when submitting forms).

- You can create new DOM nodes and insert them into your HTML document. For example:

```
// Create a new node <p>
let myParagraph = document.createElement("p")
// Give it the class "text-blue"
myParagraph.classList.add("text-blue")
// and set its text
myParagraph.textContent = "Nice to meet you!"
// Now add this element at the very end of the element with id results
document.getElementById("results").appendChild(myParagraph)
```

Improve the game

- add a new input text: its value is the upper limit of the randomly generated number
- add a new input text: its value is the number of attempts a player has before losing the game
- make the two input above `disabled` when the user starts guessing
    - you can use `elem.setAttribute("disabled", true)`
    - and `elem.removeAttribute("disabled")`
- add a new button to "Reset" the gameplay / allow the user to restart from scratch (changing the values for max number and number of tries)

- add a "counter": an HTML element that displays how many tries you have left in the current round