avionschool

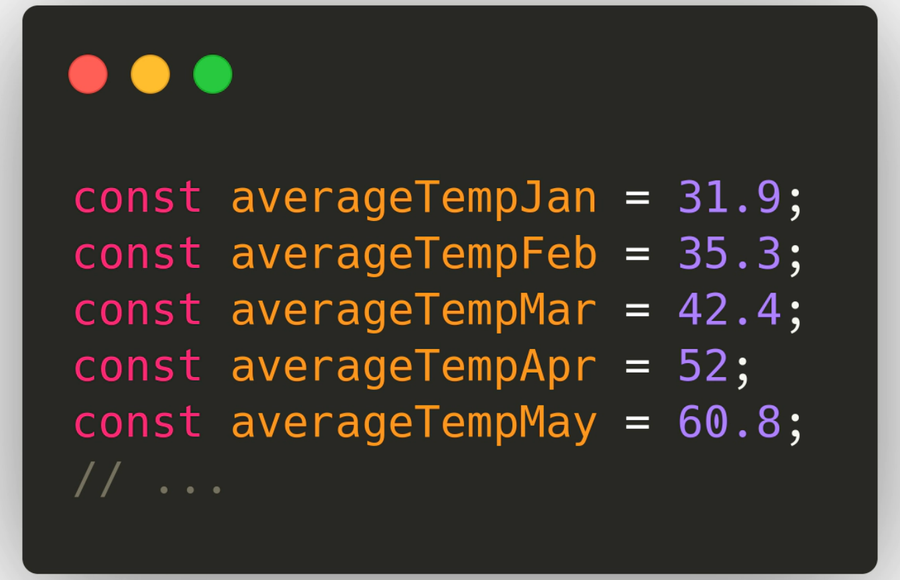
Lesson 9.0 Arrays

BATCH 4

DECEMBER 14, 2020

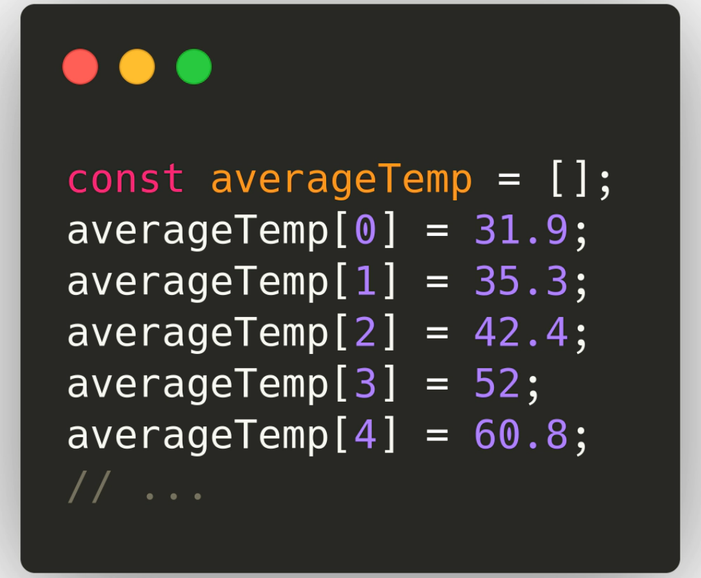
Why should we use arrays?

- Let's consider that we need to store the average temperature of each month of the year for a city.
- If we store the temperature for only one year, we can manage 12 variables.



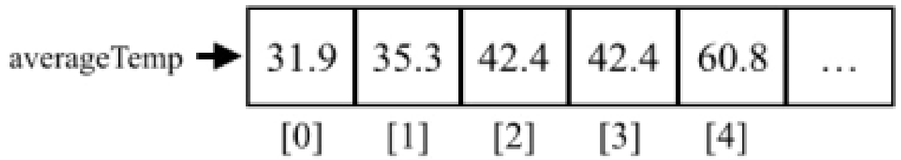
Why should we use arrays?

- However, this is not the best approach.
- What if we need to store the average temperature for more than one year?
- Fortunately, this is why arrays were created, and we can easily represent the same information mentioned earlier as follows:



```
const averageTemp = [];
averageTemp[0] = 31.9;
averageTemp[1] = 35.3;
averageTemp[2] = 42.4;
averageTemp[3] = 52;
averageTemp[4] = 60.8;
// ...
```

Why should we use arrays?



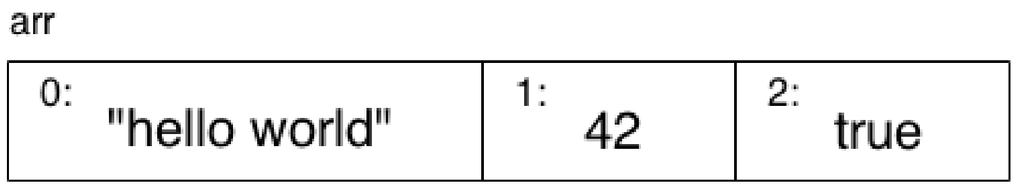
Arrays

- most common, simplest memory data structure in computer programming as every programming language includes some form of array.
- An array stores values that are all of the same datatype sequentially.
- Because arrays are built-in, they are usually very efficient and are considered good choices for many data storage purposes.
- Although JavaScript allows us to create arrays with values from different datatypes, we will follow best practices and assume that we cannot do this (most languages do not have this capability).

Arrays

- An array is an object that holds values (of any type) not particularly in named properties/keys, but rather in numerically indexed positions.
- Note: Languages that start counting at zero, like JS does, use 0 as the index of the first element in the array.

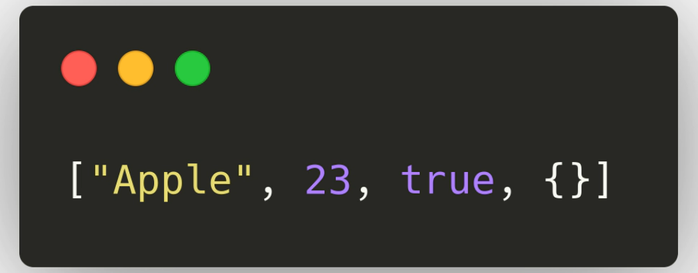
```
var arr = [
    "hello world",
    42,
    true
];

arr[0];          // "hello world"
arr[1];          // 42
arr[2];          // true
arr.length;      // 3

typeof arr;      // "object"
```



avion

Array Literal: Anatomy

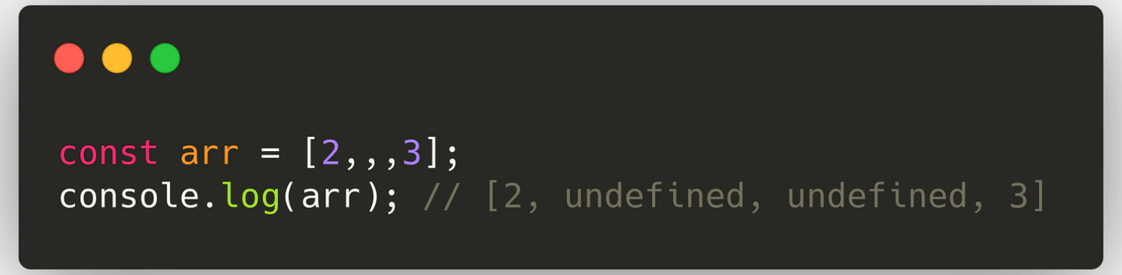
- a list of zero or more expressions, each of which represents an array element, wrapped in square brackets.



CREATE-ing Arrays

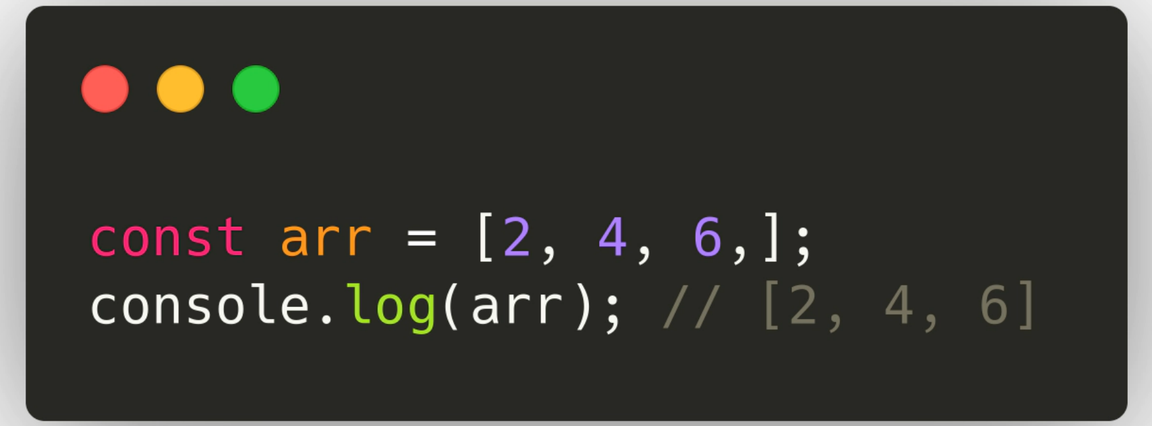
Array Literal: Skipping Elements

- While creating an array literal, elements can be skipped using commas. The skipped locations are filled by undefined.



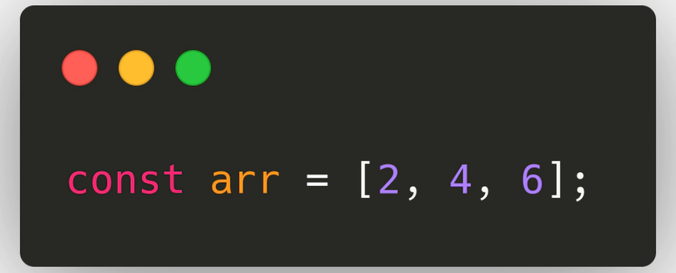
Array Literal: Trailing Comma

- Trailing comma in an array is ignored. It does NOT create an undefined element at the last.
- Trailing commas can create errors in older browsers. So it is better not to use it.



Array Declaration: Square Brackets

- An array in JavaScript can be declared using square brackets([]). All the list items are wrapped inside a pair of square bracket.



Array Declaration: Empty

- An array without any element is called an empty array. An empty array is declared using an empty pair of square bracket.

Array() Constructor: Another Declaration

- An array in JavaScript can also be declared using `Array()` constructor. The arguments passed as input to the constructor forms the array elements.

```
const arr = new Array("Apple", "Orange");
console.log(arr); // ["Apple", "Orange"];
```

Array() Constructor: Single number argument

- When we pass a single number as argument to Array(), it treats the number as array length.

```
const arr = new Array(2);
console.log(arr.length); // 2
console.log(arr); // [undefined, undefined]
```

Array() Constructor: Single number argument

- If the single argument passed to the Array() constructor is a non-number, that argument is taken as an array element.

```
const arr = new Array("Backbencher");
console.log(arr); // ["Backbencher"]
```

Array() Constructor: Empty

- An empty array is created using Array() constructor by not passing any arguments.

Array.of()

- Array.of() creates a new array from passed arguments.

Array.of(): Difference with Array()

- When passed a single number argument to Array(), it creates an empty array of length equal to the passed number. Where as, when passed a single number argument to Array.of(), it creates an array with only that number as its element.

```
console.log(Array(3)); //[undefined, undefined, undefined]
console.log(Array.of(3)); // [3]
```

READ-ing Arrays

Accessing Elements

- All elements in an array is stored as a list of items with numeric indices. The index starts from 0. An element of the array can be read by passing the index of the element.

Reading non-existent element

- When tried to read a non-existent element from an array, it returns undefined.

Index as string

- It is possible to read an element from array by passing the index as type string.

```
const arr = ["Mercedes", "BMW", "Audi"];
console.log(arr["1"]);  // "BMW"
```

Iterating Array Elements

- To access a specific position of the array, we can also use brackets, passing the index of the position we would like to access.
- But, what if we want to output all the elements?

```
let daysOfWeek = ['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday']
```

UPDATE-ing Arrays

Setting Array Elements

- Array elements can be set by initializing at the time of declaration itself. The square bracket([]) syntax or Array() syntax can be used for that.

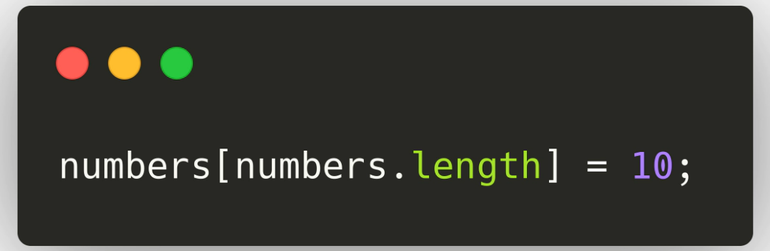
```
const arr1 = ["Apple", "Banana"];
const arr2 = new Array("Apple", "Banana");
const arr3 = Array("Apple", "Banana");
```

Setting Array Elements

- Once an array is declared, the elements in the array can be set using square brackets and element index.

```
const arr = [];
arr[0] = "Apple";
arr[1] = "Banana";
console.log(arr); // ["Apple", "Banana"]
```

Inserting an element at the end of the array

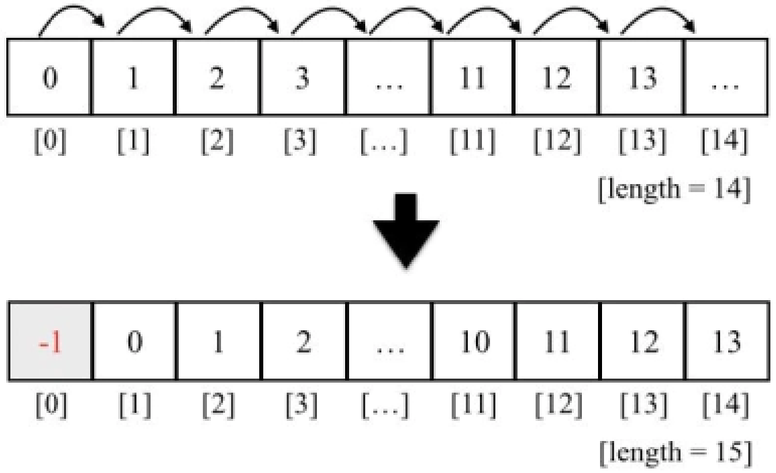


Inserting an element at the end of the array

```
numbers.push(11);
numbers.push(12, 13);
```

Inserting an element in the first position

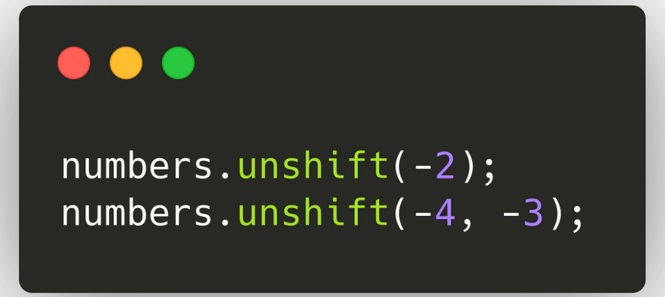
- Suppose we need to add a new element to the array (number -1) and would like to insert it in the first position, not the last one.
- To do so, first we need to free the first position by shifting all the elements to the right. We can loop all the elements of the array, starting from the last position (value of length will be the end of the array) and shifting the previous element (i-1) to the new position (i) to finally assign the new value we want to the first position (index 0).



Inserting an element in the first position

```
Array.prototype.insertFirstPosition = function(value) {
    for (let i = this.length; i >= 0; i--) {
        this[i] = this[i - 1];
    }
    this[0] = value;
};
numbers.insertFirstPosition(-1);
```

Using the unshift method



Memory Allocation

- Normally in a programming language, arrays allocate a contigous block of memory with fixed length. But in JavaScript, arrays are just Object types with special constructor and methods. So internally, JavaScript engine cannot go with contigous memory approach for arrays. It treats an array like an object(hash) internally.

Dynamic nature of arrays

- JavaScript approach of handling arrays as objects is required to handle dynamic nature of arrays. In JavaScript, we can keep on adding elements to an existing array using push() method.

```
const arr = [];
arr.push("Backbencher");
arr.push("JavaScript");
console.log(arr); // ["Backbencher", "JavaScript"]
```

Dynamic nature of arrays

- Different data types in JavaScript like Boolean, Number, String and so on, occupy different memory sizes. In JavaScript, it is possible to update an element of an array to different data type.

```
const arr = ["Backbencher", "JavaScript"];
arr[1] = 72;
console.log(arr); // ["Backbencher", 72]
```

Dynamic nature of arrays

- Different data types in JavaScript like Boolean, Number, String and so on, occupy different memory sizes. In JavaScript, it is possible to update an element of an array to different data type.
- Here, the second element of the array is changed from a string type to number type. If arrays are storing elements in contigous memory locations, then we need to shift elements in memory when the data type changes. But since arrays are internally treated as objects, it is easy to make data type changes in an array.

```
const arr = ["Backbencher", "JavaScript"];
arr[1] = 72;
console.log(arr); // ["Backbencher", 72]
```

One more thing on array declaration...

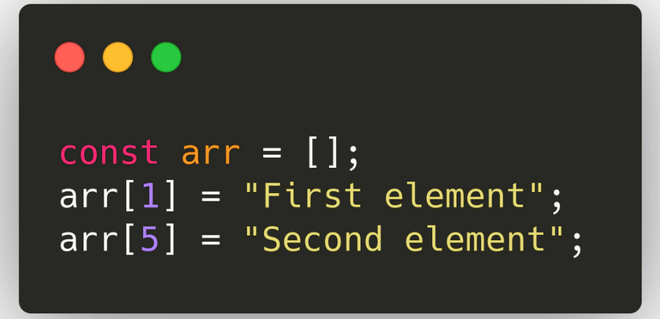
- When declaring an array using Array() constructor function, we can specify the size of the array.
- When JavaScript engine executes above line of code, is it allocating 1000 memory locations?

One more thing on array declaration...

- No. JavaScript cannot do that because it does not know what type of values are going to be in the array. JavaScript is actually creating an Array object and setting the length property of that object to 1000.
- Every array(instance of Array()) has a property length which indicates the size of the array.
- One of the easy way to delete all elements of an array is to set the length property of the array to 0.

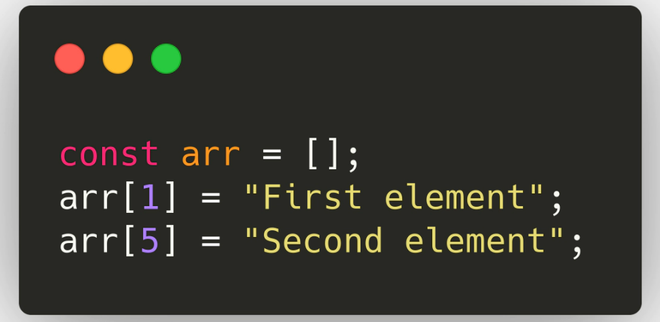
Sparse arrays

- An array with at least one "empty slot" in it is often called a "sparse array."
- Arrays in JavaScript are sparse. That means, we can create an array with non-contigous elements.



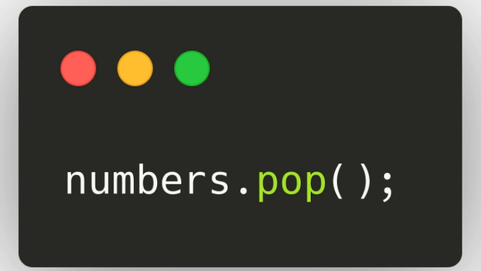
Sparse arrays

- When JavaScript engine executes above code, it creates an array object with two keys, 1 and 5 and corresponding values are stored.
- Sparse array approach in JavaScript saves memory usage, since it allocates memory only for elements that have data.
- What if we try to print the value of arr[3]? It prints undefined. That is not because JavaScript is filling all gaps with undefined. It is because JavaScript is designed to return undefined when we access a non-existent property of an object.
- In the above code snippet, when we assign a value to index 5, JavaScript automatically sets the value of arr.length to 6. That means, JavaScript always sets the length property value greater than the largest index value of the array.

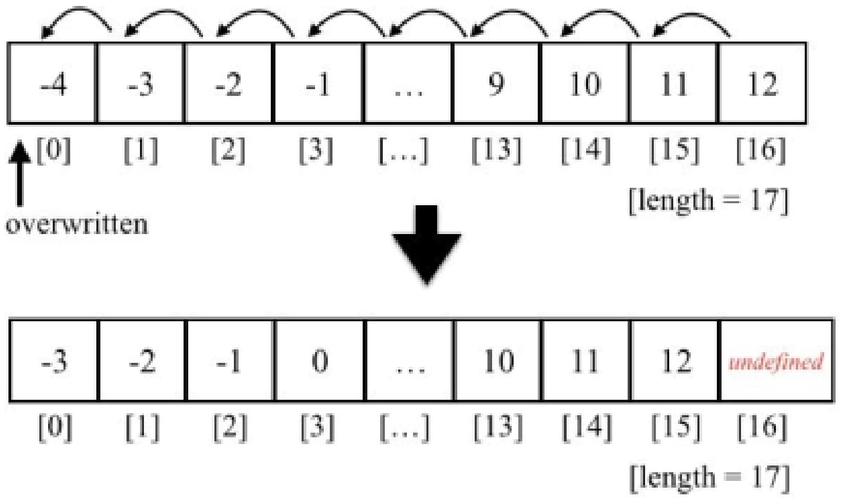


DELETE-ing(from) Arrays

Removing an element from the end of the array



Removing an element from the first position



Removing an element from the first position

- We have only overwritten the array's original values, and we did not really remove the value (as the length of the array is still the same and we have this extra undefined element).

```
for (let i = 0; i < numbers.length; i++) {
    numbers[i] = numbers[i + 1];
}
```

Removing an element from the first position

- To really remove the element from the array, we need to create a new array and copy all values other than undefined values from the original array to the new one and assign the new array to our array.

```
Array.prototype.reIndex = function(myArray) {
    const newArray = [];
    for(let i = 0; i < myArray.length; i++ ) {
        if (myArray[i] !== undefined) {
            // console.log(myArray[i]);
            newArray.push(myArray[i]);
        }
    }
    return newArray;
}

// remove first position manually and reIndex
Array.prototype.removeFirstPosition = function() {
    for (let i = 0; i < this.length; i++) {
        this[i] = this[i + 1];
    }
    return this.reIndex(this);
};
numbers = numbers.removeFirstPosition();
```

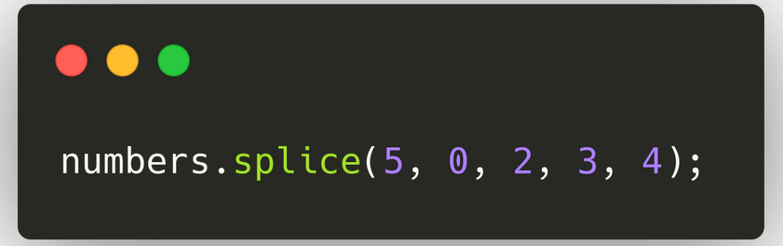
Using the shift method

Adding and removing elements from a specific position

- This code will remove three elements, starting from index 5 of our array

Adding and removing elements from a specific position

- The first argument of the method is the index we want to remove elements from or insert elements into. The second argument is the number of elements we want to remove (in this case, we do not want to remove any, so we will pass the value 0 (zero)). And from the third argument onward we have the values we would like to insert into the array (the elements 2 , 3 , and 4).



Adding and removing elements from a specific position

- Use control flow and create another array instance!

Array methods

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

Coding Challenge

- Create a game of tic tac toe.
- The design and representation and responsiveness of the tic tac toe board in the DOM is up to your creativity and imagination.
- The rule of whose turn it is (X or O) must be observed. You can choose whether X or O will play first.
- The "state" of the board should strictly come from a two-dimensional array (In JS, that's array within an array).
- You may use array **built-in methods.**

```
// an example of 2d array on JS
let board = [
  ['X', 'O', 'X'],
  ['X', 'O', ''],
  ['O', 'O', ''],
]
```

Coding Challenge

EXTRA CHALLENGE (REQUIRED)

- When a player wins, you must show the **history** of the game.
- To do that, every move should be "saved".
- Implement two **buttons, "Previous" and "Next",** that will show up when a game is done / finished.
- When clicking previous or next, the board should show the current move at that moment / turn.
- If there's no "next move", the next button must be disabled. And if there's no "previous move", the previous button must be disabled.
- Also, implement a **reset button**, that:
  1. restarts the game
  2. hides the next and prev buttons
  3. clears move history
- Yes. Refreshing the page clears the history of the game.
- No. You cannot use any library like React etc. (You can check them as references though.)
- No. You cannot use class(objects). Let's be "functional" for now!

```
// an example of how we can save the moves of the game
let moves = [
  [ // board before winning
    ['X', '', 'X'],
    ['X', 'O', ''],
    ['O', 'O', ''],
  ],
  [  // board when O wins
    ['X', 'O', 'X'],
    ['X', 'O', ''],
    ['O', 'O', ''],
  ],
]
```