WDD 330 Web Frontend Development II
Samuel Palacios
Week 02
May 7th, 2022

Readings Assignment Notes

1. Read Ch2: Programming basics

- ✓ I find it very interesting that JavaScript has **"Undefined"** as a data type. At first, I thought that this should be "Null" and not another data type. This data type indicates that the variable simply has no value assigned to it.

- ✓ Always talking about data types, another basic thing: if a variable does not belong to a primitive data type, by definition... it is an **"Object"**.

- ✓ Regarding the **scope**, in JavaScript, we have global and local variables, the latter of blocks such as functions or certain statements. I finally understood the difference between using **"var"** or **"let"** when declaring a variable.

- ✓ Regarding the handling of memory spaces, variables can be assigned **by reference** or **by value**, if objects or primitive values are handled respectively.

- ✓ All right with the methods of the **String class**, are essential for manipulating strings. The **"Template Literals"** thing is great, I didn't know JavaScript already had this, I used it a lot in Python and it's great to have it in JS as well.

- ✓ I was also not aware of the **Symbol** primitive type, where immutable distinct values are generated which are ideal for identifiers that are generated once and are not going to change.

- ✓ With the **Number** methods, all good, I found it interesting, and I didn't know about the "isInfinite" method, I find some cases where it can be very necessary.

- ✓ All very clear with operators, implicit conversions, casting, boolean data types, etc. Also, all very well with bitwise operators, comparison operators, etc.

2. Read all of Ch3: Arrays, Logic, and Loops

- ✓ I had never delved into the subject of **"Arrays"** in JS, I found it interesting that several examples used arrays declared with the word "constant", but it was about non-immutable data...as the only constant is the reference to the object of the arrangement.

- ✓ With the methods used with arrays, I hadn't used the "**Shift"** and "**Unshift"** methods which are very similar to others that languages like Python used for arrays. The **"join"** method seemed great to me; it allows to join all the elements of the array with a specific separator in the same chain.

- ✓ All is good with the other operations on arrays like slicing, splicing, reverse, and sort. I delved into the **sort** of method, and you can create a custom method to do the sorting of the items if you need something other than the default behavior.

- ✓ The classic method for finding an item in an array is still **"indexOf".** The "includes" method is more practical, it handles a boolean value that indicates whether the item is inside the array.

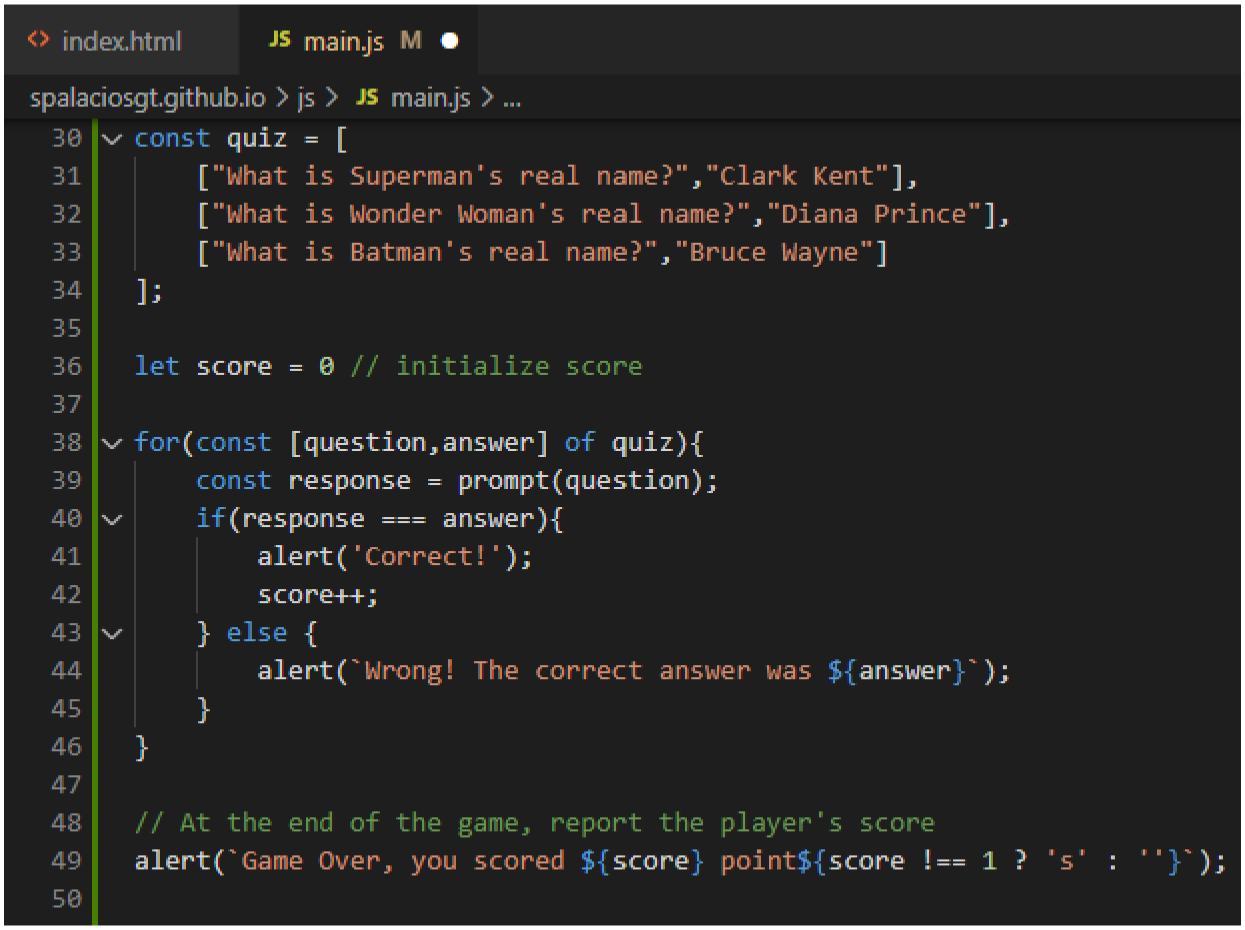
- ✓ With **multidimensional arrays**, more positions or indices are added to the arrays. At first, it is a bit difficult to understand, but as soon as they are registered and the values are explored, it is understood. With one dimension it is a vector, a one-column table; with two dimensions it is a table with as many rows and columns; with 3 dimensions it is a list of tables, and with 4 dimensions it is a table of tables, and so on.

- ✓ About the topic of **Sets**, very interesting... they are like arrays but only allow **unique values**, that is, they do not repeat and can be located unequivocally. Sets can be converted to arrays and vice versa.

- ✓ Reading the topic of **"Weak Sets"** I find out that in JavaScript there is also a kind of **Garbage Collector** as in the Java programming language to be able to free space from the heap of objects whose pointer is null.

- ✓ The **"Maps"**, here begins the interesting. It is a key-value-oriented data structure. A map can be formed by an array of keys or keys, and another array of values. These values can be primitive data or objects, other arrays, etc.

- ✓ With conditional or **logical** statements, everything is ok. With the **"loops"** the most interesting thing is to be able to go through data structures through lambda functions, which is the current trend than the typical "for".



```javascript
const quiz = [
    ["What is Superman's real name?","Clark Kent"],
    ["What is Wonder Woman's real name?","Diana Prince"],
    ["What is Batman's real name?","Bruce Wayne"]
];

let score = 0 // initialize score

for(const [question,answer] of quiz){
    const response = prompt(question);
    if(response === answer){
        alert('Correct!');
        score++;
    } else {
        alert(`Wrong! The correct answer was ${answer}`);
    }
}

// At the end of the game, report the player's score
alert(`Game Over, you scored ${score} point${score !== 1 ? 's' : ''}`);
```

3. **Read all of Ch4: Functions**

✓ The **"Function"** is the basic code block of JavaScript and many interpreted languages such as Python. It is the most common way of grouping logic before the definition of objects. In JavaScript, as in other languages, functions can be used as expressions. That is, they do not need to be defined previously but can be expressed in a declaration.

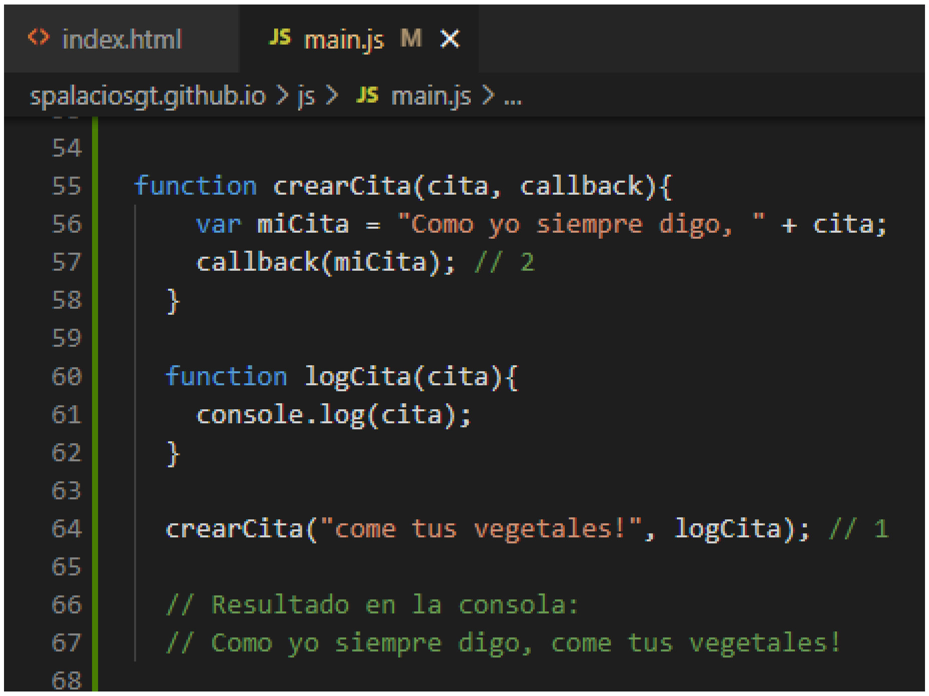
✓ Each function has its header or signature, and may or may not have parameters, even by default. Also, a function may or may not return a value, usually, a good function does a single task and **returns a result**.

✓ Since functions are the basic blocks of code, there are all kinds of variants to declare and define a function, such as having no parameters, having a fixed list of parameters, varying the parameter signature or header, default parameters, or having undefined parameters (**a list of parameters).**), etc.

✓ Callbacks, with this topic the first thing we must understand is that functions are objects. A **"Callback"** function is one that accepts another function as parameters (higher-order function), which will be called again later and contains the logic to know when that call will be made.

- ✓ One of the reasons to use "Callback" functions is to execute asynchronous blocks of code, for example, if we need to wait for the result of a call to an external API or that is in another resource,

- ✓ As "Array Iterators" we have some functions that allow us to modify one or more values of an array without necessarily using a "For Each". For example, map() allows you to modify the values of the array by applying an operation or function, reduce() allows you to summarize the values of an array in a row by applying some operation, and filter() allows you to reduce the data set by applying a criterion.



```
function crearCita(cita, callback){
    var miCita = "Como yo siempre digo, " + cita;
    callback(miCita); // 2
  }

  function logCita(cita){
    console.log(cita);
  }

  crearCita("come tus vegetales!", logCita); // 1

  // Resultado en la consola:
  // Como yo siempre digo, come tus vegetales!
```