**Language syntax.**

**Variables**

To begin with, we will deal with declaring variables. The most typical way that we can find on the Internet to declare variables in Javascript is through the reserved word var, which allows you to declare variables of any type. For example:

var name = "Nacho";

var age = 40;

However, this way of declaring variables has some drawbacks, such as, and above all, the fact of declaring a variable locally to a scope, and that it can be used from outside that scope, because the variable is valid within of the function where it has been defined. So, for example, this code would work, and it would show "Nacho" as the name in both cases, despite the fact that, intuitively, the variable name should not exist outside the if:

if (2> 1)

{

var name = "Nacho";

console.log ("Name inside:", name);

}

console.log ("Name out:", name);

To avoid these vulnerabilities, we will use the let keyword, instead of var, to declare variables:

if (2> 1)

{

let name = "Nacho";

console.log ("Name inside:", name);

}

console.log ("Name out:", name);

In this way, the scope of each variable is restricted to the block where it is declared, and the previous code would cause an error.

<https://www.w3schools.com/js/js_let.asp>

Remember, too, that we can use the const word to define constants in code. This will be particularly useful both for defining conventional constants (such as a text or fixed number, for example) and for loading libraries, as we will see in later sessions.

const pi = 3.1416;

IMPORTANT: the inclusion of let and const took place with the ES6 specification, also known as ECMAScript2015, so during these first years it is possible that some browsers do not support it to develop frontends .. but it is supported in node.

**Conditional structure**

The if structure behaves as in most programming languages. What it does is evaluate a logical condition, returning a Boolean as the result and if it is true, it executes the code that is inside the if block. Optionally we can add the else if block, and an else block (the same as in other structured programming languages).

var price = 65;

if (price <50) {

console.log ("This is cheap!");

} else if (price <100) {

console.log ("This is not cheap ...");

} else {

console.log ("This is expensive!");

}

The switch structure has a behavior similar to that of other programming languages. As we know, a variable is evaluated and the block corresponding to the value it has is executed (it can be number, string, ..). Normally, you need to put the break instruction at the end of each block, since otherwise it would continue executing the instructions in the next block. An example where two values ​​will execute the same block of code is the following:

var userType = 1;

switch (userType) {

case 1:

case 2: // Types 1 and 2 enter here

console.log ("You can access this zone");

break;

case 3:

console.log ("You don't have permission to access here");

break;

default: // None of the above

console.error ("Bad user type!");

}

**Loops**

We have the typical while loop that evaluates a condition and repeats itself over and over until the condition is false (or if the condition is false from the beginning, it does not perform the block of instructions it contains).

var value = 1;

while (value <= 5) {// Print 1 2 3 4 5

console.log (value ++);

}

In addition to while, we can use do..while. The main difference is that the condition check is performed at the end of the instruction block, therefore the code will always be executed at least once.

var value = 1;

do {// Print 1 2 3 4 5

console.log (value ++);

} while (value <= 5);

The for loop works the same as in other programming languages. We initialize one or more values, we establish the completion condition and the third section is to establish the increment or decrement (or the instructions that will be executed at the end of each iteration).

var limit = 5;

for (var i = 1; i <= limit; i ++) {// Print 1 2 3 4 5

console.log (i);

}

As you know, you can initialize one or more variables and also execute several instructions in each iteration separating them with commas.

var limit = 5;

/ \* Print

fifteen

2 - 4

3 - 3

4 - 2

5 - 1

\* /

for (var i = 1, j = limit; i <= limit && j> 0; i ++, j--) {

console.log (i + "-" + j);

}

Another version of the for is the for..in loop. With this loop we can iterate the indexes of an array or the properties of an object (similar to the foreach loop in other languages, but going through the indexes instead of the values).

var ar = new Array (4, 21, 33, 24, 8);

for (var index in ar) {// Print 4 21 33 24 8

console.log (ar [index]);

}

**Arrays**

In JavaScript, arrays are a special type of object. We can create an array with the instance of an Array class object. These do not have a fixed size, therefore, we can initialize it with a size and then add more elements to it.

The constructor can receive 0 parameters (empty array), 1 number (the size of the array), or in any other case, an array will be created with the received elements. We must bear in mind that in JavaScript an array can contain different types of data at the same time: number, string, boolean, object, etc.

var a = new Array (); // Create an empty array

a [0] = 13;

console.log (a.length); // Print 1

console.log (to [0]); // Print 13

console.log (to [1]); // Print undefined

Note that when you access an array position that has not been defined, it returns undefined. The length of an array depends on the positions that have been assigned. Let's see an example of what happens when you assign a position greater than the length and that is not consecutive to the last assigned value.

var a = new Array (12); // Create an array of size 12

console.log (a.length); // Print 12

a [20] = "Hello";

console.log (a.length); // Now print 21 (0-20). Positions 0-19 will have the value undefined

We can reduce the length of the array by directly modifying the length property of the array (length). If we reduce the length of an array, the positions greater than the new length will be considered undefined.

var a = new Array ("a", "b", "c", "d", "e"); // Array with 5 values

console.log (to [3]); // Print "d"

a.length = 2; // Positions 2-4 will be destroyed

console.log (to [3]); // Print undefined

You can create an array using square brackets instead of using new Array (). The elements that we put inside, separated by commas, will be the elements that the array will initially have.

var a = ["a", "b", "c", "d", "e"]; // Array of size 5, with 5 values ​​initially

console.log (typeof a); // Print object

console.log (a instanceof Array); // Print true. a is an instance of array

a [a.length] = "f"; // We insert a new element at the end

console.log (a); // Print ["a", "b", "c", "d", "e", "f"]

**Methods for arrays**

Let's see how to insert values ​​at the beginning of an array (unshift) and at the end (push). And we will see two different ways to display their values ​​by console.

var a = [];

a.push ("a"); // Insert the value at the end of the array

a.push ("b", "c", "d"); // Insert these new values ​​at the end

console.log (a.valueOf ()); // Print ["a", "b", "c", "d"]. You can omit valueOf (), it will still be called

a.unshift ("A", "B", "C"); // Insert new values ​​at the beginning of the array

console.log (a.toString ()); // Print A, B, C, a, b, c, d. toString () is a bit different from valueOf ()

Now, let's see the opposite operation. We are going to remove from the beginning (shift) and also from the end (pop) of the array. These operations will return the value that has been removed.

console.log (a.pop ()); // Print and delete the last position → "d"

console.log (a.shift ()); // Print and delete the first position → "A"

console.log (a); // Print ["B", "C", "a", "b", "c"]

We can print the elements of an array using join () instead of toString (). By default, it returns a string with all elements separated by commas. However, we can specify the separator to print.

var a = [3,21,15,61,9];

console.log (a.join ()); // Print "3,21,15,61,9" "

console.log (a.join ("- # -")); // Print "3 - # - 21 - # - 15 - # - 61 - # - 9"

How do we concatenate two arrays? Using concat.

var a = ["a", "b", "c"];

var b = ["d", "e", "f"];

var c = a.concat (b);

console.log (c); // Print ["a", "b", "c", "d", "e", "f"]

console.log (a); // Print ["a", "b", "c"]. Array a has not been modified

The slice method returns a new array from the intermediate positions of another.

var a = ["a", "b", "c", "d", "e", "f"];

var b = a.slice (1, 3); // (start position → included, end position → excluded)

console.log (b); // Print ["b", "c"]

console.log (a); // Print ["a", "b", "c", "d", "e", "f"]. The original array is not modified

console.log (a.slice (3)); // A parameter. Returns from position 3 to the end → ["d", "e", "f"]

**splice** removes elements from the original array and returns the removed elements. It also allows inserting new values

var a = ["a", "b", "c", "d", "e", "f"];

a.splice (1, 3); // Remove 3 elements from position 1 ("b", "c", "d")

console.log (a); // Print ["a", "e", "f"]

a.splice (1,1, "g", "h"); // Delete 1 element in position 1 ("e"), and insert "g", "h" in that position

console.log (a); // Print ["a", "g", "h", "f"]

a.splice (3, 0, "i"); // At position 3, delete nothing, and insert "i"

console.log (a); // Print ["a", "g", "h", "i", "f"]

We can reverse the order of the array using the reverse method.

var a = ["a", "b", "c", "d", "e", "f"];

a.reverse (); // Reverse the original array

console.log (a); // Print ["f", "e", "d", "c", "b", "a"]

Also, we can sort the elements of an array using the sort method.

var a = ["Peter", "Anne", "Thomas", "Jen", "Rob", "Alison"];

a.sort (); // Sort the original array

console.log (a); // Print ["Alison", "Anne", "Jen", "Peter", "Rob", "Thomas"]

But what happens if we try to sort elements that are not strings? By default, it will sort it by its value as a string (taking into account that if they are objects, it will try to call the toString () method to sort it). To do this, we will have to pass a function (of ordering), which will compare 2 values ​​of the array and return a numerical value indicating which is less (negative if the first is less, 0 if they are equal and positive if the first is greater).

var a = [20, 6, 100, 51, 28, 9];

a.sort (); // Sort the original array

console.log (a); // Print [100, 20, 28, 51, 6, 9]

a.sort (function (n1, n2) {

return n1 - n2;

});

console.log (a); // Print [6, 9, 20, 28, 51, 100]

**New methods ES6**

Using indexOf, we can know if the value we pass to it is in the array or not. If it finds it, it returns the first position where it is, and if not, it returns -1. Using the lastIndexOf method it returns the first occurrence found starting from the end.

var a = [3, 21, 15, 61, 9, 15];

console.log (a.indexOf (15)); // Print 2

console.log (a.indexOf (56)); // Print -1. Not found

console.log (a.lastIndexOf (15)); // Print 5

The every method will return a boolean indicating if all the elements of the array meet a certain condition. This function will receive any element, test it, and return true or false depending on whether it meets the condition or not.

var a = [3, 21, 15, 61, 9, 54];

console.log (a.every (function (num) {// Check if each number is less than 100

return num <100;

})); // Print true

console.log (a.every (function (num) {// Check if each number is even

return num% 2 == 0;

})); // Print false

On the other hand, the some method is similar to every, but it returns true when one of the elements of the array meets the condition.

var a = [3, 21, 15, 61, 9, 54];

console.log (a.some (function (num) {// Check if any element of the array is even

return num% 2 == 0;

})); // Print true

We can iterate through the elements of an array using the forEach method. Optionally, we can keep track of the index that is being accessed at all times, and even receive the array as a third parameter.

var a = [3, 21, 15, 61, 9, 54];

var sum = 0;

a.forEach (function (num) {//

sum + = num;

});

console.log (sum); // Print 163

a.forEach (function (num, index, array) {// index and array are optional parameters

console.log ("Index" + index + "in [" + array + "] is" + num);

}); // Print -> Index 0 in [3,21,15,61,9,54] is 3, Index 1 in [3,21,15,61,9,54] is 21, ...

To modify all the elements of an array, the map method receives a function that transforms each element and returns it. This method will return at the end a new array of the same size containing all the resulting elements.

var a = [4, 21, 33, 12, 9, 54];

console.log (a.map (function (num) {

return num \* 2;

})); // Print [8, 42, 66, 24, 18, 108]

To filter the elements of an array, and obtain an array that contains only the elements that meet a certain condition, we use the filter method.

var a = [4, 21, 33, 12, 9, 54];

console.log (a.filter (function (num) {

return num% 2 == 0; // If it returns true, the element stays in the returned array

})); // Print [4, 12, 54]

The reduce method uses a function that accumulates a value, processing each element (second parameter) with the accumulated value (first parameter). As the second parameter of reduce, you should pass an initial value. If you don't pass an initial value, the first element of an array will be used as such (if the array is empty it would return undefined).

var a = [4, 21, 33, 12, 9, 54];

console.log (a.reduce (function (total, num) {// Sum all elements of the array

return total + num;

}, 0)); // Print 133

console.log (a.reduce (function (max, num) {// Get the maximum number of the array

return num> max? num: max;

}, 0)); // Print 54

If you prefer to do the same thing that reduce does but in reverse, we'll use reduceRight.

var a = [4, 21, 33, 12, 9, 154];

console.log (a.reduceRight (function (total, num) {// Start with the last number and subtract all others

numbers=null;

return total - num;

})); // Print 75 (If we don't want to send it an initial value, it will start with the value of the last position of the array