Sintaxis del lenguaje

Contenido

1. Variables

Para empezar, nos ocuparemos de declarar variables. La forma más típica que podemos encontrar en Internet para declarar variables en Javascript es a través de la palabra reservada “var”, que permite declarar variables de cualquier tipo. Por ejemplo:

```
var name = "Nacho";

var age = 40;
```

Sin embargo, esta forma de declarar variables presenta algunos inconvenientes. Uno de ellos es que, al usar **var**, una variable declarada dentro de un bloque (por ejemplo, dentro de un if) sigue siendo accesible desde fuera de ese bloque. Esto ocurre porque la validez de var se limita al ámbito de la función y no al del bloque en el que se define. Por ejemplo, el siguiente código funcionaría y mostraría *"Nacho"* en ambos casos, aunque de forma intuitiva cabría esperar que la variable no existiera fuera del if:

```
if (2> 1)

{

        var name = "Nacho";

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

}

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

Para evitar esas vulnerabilidades, usaremos la palabra “let” en lugar de “var”, para declarar variables:

```
if (2> 1)

{

        let name = "Nacho";

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

}

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

De esta forma, el alcance de cada variable se restringe al bloque donde se declara, y el

código anterior daría error.
https://www.w3schools.com/js/js_let.asp

Recuerda que también podemos usar la palabra const para definir constantes en el código. Esto será especialmente útil tanto para definir constantes convencionales (como un texto o un número fijo, por ejemplo) como para cargar bibliotecas, como veremos en sesiones posteriores.

```
const pi = 3.1416;
```

2. Estructura de los condicionales

La estructura if se comporta como en la mayoría de los lenguajes de programación. Lo que hace es evaluar una condición lógica, devolviendo como resultado un booleano y si es verdadero, ejecuta el código que está dentro del bloque if. Opcionalmente podemos agregar el bloque else if y un bloque else (igual que en otros lenguajes de programación estructurados).

```
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!");
}
```

La estructura del “case” tiene un comportamiento similar al de otros lenguajes de programación. Como sabemos, se evalúa una variable y se ejecuta el bloque correspondiente al valor (puede ser número, cadena...). Normalmente, debe colocar la instrucción de interrupción al final de cada bloque, ya que de lo contrario continuaría ejecutando las instrucciones en el siguiente bloque. Un ejemplo donde dos valores ejecutarán el mismo bloque de código es el siguiente:

```
var userType = 1;
switch (userType) {
```

```
case 1:
case 2: // Tipos 1 and 2 entran aquí
console.log ("Puedes entrar a esta zona");
break;
case 3:
console.log ("No tienes premise aquí");
break;
default: // Ninguno de los anteriores
console.error ("Tipo de usuario incorrecto!");
}
```

3. Bucles

Tenemos el típico ciclo while que evalúa una condición y se repite una y otra vez hasta que la condición es falsa (o si la condición es falsa desde el principio, no realiza el bloque de instrucciones que contiene).

```
var value = 1;
while (value <= 5) {// Print 1 2 3 4 5
        console.log (value ++);
}
```

Además de while, podemos usar do...while. La principal diferencia es que la verificación de la condición se realiza al final del bloque de instrucciones, por lo que el código siempre se ejecutará al menos una vez.

```
var value = 1;
do {// Print 1 2 3 4 5
        console.log (value ++);
} while (value <= 5);
```

El bucle for funciona igual que en otros lenguajes de programación. Inicializamos uno o más valores, establecemos la condición de finalización y el tercer apartado es establecer el incremento o decremento (o las instrucciones que se ejecutarán al final de cada

iteración).

```
var limit = 5;
for (var i = 1; i <= limit; i ++) {// Print 1 2 3 4 5
	console.log (i);
}
```

Como sabes, puedes inicializar una o más variables y también ejecutar varias instrucciones en cada iteración separándolas con comas.

```
var limit = 5;
/ * Imprime
1 - 5
2 - 4
3 - 3
4 - 2
5 - 1
* /
for (var i = 1, j = limit; i <= limit && j> 0; i ++, j--) {
	console.log (i + "-" + j);
}
```

Otra versión de for es el bucle for..in. Con este bucle podemos iterar los índices de una matriz o las propiedades de un objeto (similar al bucle foreach en otros lenguajes, pero pasando por los índices en lugar de los valores).

```
var ar = new Array (4, 21, 33, 24, 8);
for (var index in ar) {// Print 4 21 33 24 8
	console.log (ar [index]);
}
```

4. Arrays

En JavaScript, los arrays son un tipo especial de objeto. Podemos crear un array con una instancia de la clase Array. Estos no tienen un tamaño fijo, por lo tanto, podemos inicializarlos con un tamaño y luego añadir más elementos.

El constructor puede recibir:

- 0 parámetros → crea un array vacío.
- 1 número → define el tamaño del array.
- Cualquier otro caso → creará un array con los elementos recibidos.

Debemos tener en cuenta que en JavaScript un array puede contener distintos tipos de datos al mismo tiempo: números, cadenas, booleanos, objetos, etc.

```
var a = new Array(); // Crea un array vacío
a[0] = 13;
console.log(a.length); // Imprime 1
console.log(a[0]); // Imprime 13
console.log(a[1]); // Imprime undefined
```

Al acceder a una posición no definida de un array, se devuelve undefined. La propiedad length depende de las posiciones que hayan sido asignadas.

Ejemplo con salto de posiciones:

```
var a = new Array(12); // Crea un array de tamaño 12
console.log(a.length); // Imprime 12
a[20] = "Hello";
console.log(a.length); // Ahora imprime 21 (0–20). Las posiciones 0–19 tendrán valor undefined
```

Podemos **reducir la longitud** de un array modificando directamente su propiedad length. Al hacerlo, las posiciones mayores que el nuevo tamaño se destruyen:

```
var a = new Array("a", "b", "c", "d", "e"); // Array con 5 valores
console.log(a[3]); // Imprime "d"
a.length = 2; // Se destruyen las posiciones 2–4
console.log(a[3]); // Imprime undefined
```

También se pueden crear arrays con corchetes:

```
var a = ["a", "b", "c", "d", "e"]; // Array de tamaño 5, con 5 valores iniciales
console.log(typeof a); // Imprime object
console.log(a instanceof Array); // Imprime true
a[a.length] = "f"; // Insertamos un nuevo elemento al final
console.log(a); // Imprime ["a", "b", "c", "d", "e", "f"]
```

5. Métodos para arrays

5.1. unshift y push

Veamos cómo insertar valores al principio de un array (**unshift**) y al final (**push**). También veremos dos formas diferentes de mostrar sus valores por consola.

- **push()** → inserta al final.
- **unshift()** → inserta al principio.

```
var a = [];
a.push("a");
a.push("b", "c", "d");
console.log(a.valueOf()); // ["a", "b", "c", "d"]

a.unshift("A", "B", "C");
console.log(a.toString()); // "A,B,C,a,b,c,d"
```

5.2. pop y shift

Ahora, veamos la operación contraria. Vamos a eliminar elementos desde el principio (**shift**) y también desde el final (**pop**) del array. Estas operaciones devolverán el valor que ha sido eliminado.

- **pop()** → elimina y devuelve el último.
- **shift()** → elimina y devuelve el primero.

```
console.log(a.pop()); // "d"

console.log(a.shift()); // "A"

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

5.3. join() y toString()

Podemos imprimir los elementos de un array usando **join()** en lugar de **toString()**. Por defecto, devuelve una cadena con todos los elementos separados por comas. Sin embargo, podemos especificar el separador que queramos usar al imprimir.

- **join()** → devuelve una cadena con separadores personalizados.

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

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

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

5.4. concat

¿Cómo concatenamos 2 arrays? Usando **concat.**

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

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

var c = a.concat(b);

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

5.5. slice

El método **slice** devuelve un nuevo arrays de las posiciones intermedias de otro.

```
var a = ["a", "b", "c", "d", "e", "f"];
console.log(a.slice(1, 3)); // ["b", "c"]
console.log(a.slice(3)); // ["d", "e", "f"]
```

5.6. splice

El método **splice** elimina elementos del array original y devuelve los elementos borrados. Tambien permite insertar nuevos valores.

```
var a = ["a", "b", "c", "d", "e", "f"];
a.splice(1, 3); // Borra 3 desde posición 1
console.log(a); // ["a", "e", "f"]

a.splice(1, 1, "g", "h");
console.log(a); // ["a", "g", "h", "f"]

a.splice(3, 0, "i");
console.log(a); // ["a", "g", "h", "i", "f"]
```

5.7. reverse

Podemos usar el método **reverse** para invertir el orden

```
var a = ["a", "b", "c", "d", "e", "f"];
a.reverse();
console.log(a); // ["f", "e", "d", "c", "b", "a"]
```

5.8. sort

También podemos usar **sort** para ordenar los elementos:

```
var b = ["Peter", "Anne", "Thomas", "Jen", "Rob", "Alison"];
b.sort();
console.log(b); // ["Alison", "Anne", "Jen", "Peter", "Rob", "Thomas"]
```

Pero, ¿qué ocurre si intentamos ordenar elementos que no son cadenas? Por defecto, se ordenarán según su valor convertido a cadena (teniendo en cuenta que, si son objetos, intentará llamar al método **toString()** para ordenarlos). Para solucionarlo, debemos pasar una **función de ordenación**, que compare 2 valores del array y devuelva un valor numérico indicando cuál es menor (negativo si el primero es menor, 0 si son iguales y positivo si el primero es mayor).

```
var a = [20, 6, 100, 51, 28, 9];
a.sort();
console.log(a); // ["100", "20", "28", "51", "6", "9"]

a.sort(function (n1, n2) {
  return n1 - n2;
});
console.log(a); // [6, 9, 20, 28, 51, 100]
```

6. Nuevos métodos ES6

6.1. indexOf

Usando **indexOf**, podemos saber si el valor que le pasamos está en el array o no. Si lo encuentra, devuelve la primera posición en la que está, y si no, devuelve -1. Usando el método **lastIndexOf** devuelve la primera ocurrencia encontrada comenzando desde el final.

```
var a = [3, 21, 15, 61, 9, 15];
console.log(a.indexOf(15)); // Imprime 2
console.log(a.indexOf(56)); // Imprime -1. No encontrado
console.log(a.lastIndexOf(15)); // Imprime 5
```

6.2. every

El método **every** devolverá un booleano indicando si todos los elementos del array

cumplen una determinada condición. Esta función recibirá cada elemento, lo probará y devolverá true o false dependiendo de si cumple la condición o no.

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

console.log(a.every(function (num) { // Comprueba si cada número es menor que 100

  return num < 100;

})); // Imprime true


console.log(a.every(function (num) { // Comprueba si cada número es par

  return num % 2 == 0;

})); // Imprime false
```

6.3. some

Por otro lado, el método **some** es similar a **every**, pero devuelve true cuando uno de los elementos del array cumple la condición.

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

console.log(a.some(function (num) { // Comprueba si algún elemento del array es par

  return num % 2 == 0;

})); // Imprime true
```

6.4. forEach

Podemos iterar a través de los elementos de un array usando el método **forEach**. Opcionalmente, podemos llevar el control del índice que se está accediendo en todo momento, e incluso recibir el array como un tercer parámetro.

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

var sum = 0;

a.forEach(function (num) { //

  sum += num;

});

console.log(sum); // Imprime 163
```

```
a.forEach(function (num, index, array) { // index y array son parámetros opcionales
  console.log("Index" + index + " in [" + array + "] is " + num);
});
// Imprime -> Index 0 in [3,21,15,61,9,54] is 3, Index 1 in [3,21,15,61,9,54] is 21, ...
```

6.5. map

Para modificar todos los elementos de un array, el método **map** recibe una función que transforma cada elemento y lo devuelve. Este método devolverá al final un nuevo array del mismo tamaño que contiene todos los elementos resultantes.

```
var a = [4, 21, 33, 12, 9, 54];
console.log(a.map(function (num) {
  return num * 2;
})); // Imprime [8, 42, 66, 24, 18, 108]
```

6.6. filter

Para filtrar los elementos de un array, y obtener un array que contenga solo los elementos que cumplen una determinada condición, usamos el método **filter**.

```
var a = [4, 21, 33, 12, 9, 54];
console.log(a.filter(function (num) {
  return num % 2 == 0; // Si devuelve true, el elemento permanece en el array devuelto
})); // Imprime [4, 12, 54]
```

6.7. reduce

El método **reduce** usa una función que acumula un valor, procesando cada elemento (segundo parámetro) con el valor acumulado (primer parámetro). Como segundo parámetro de reduce, se debe pasar un valor inicial. Si no se pasa, se usará como tal el primer elemento del array (si el array está vacío devolvería undefined).

```
var a = [4, 21, 33, 12, 9, 54];
console.log(a.reduce(function (total, num) { // Suma todos los elementos del array
```

```
  return total + num;
}, 0)); // Imprime 133


console.log(a.reduce(function (max, num) { // Obtiene el número máximo del array
  return num > max ? num : max;
}, 0)); // Imprime 54
```

6.8. reduceRight

Si preferimos hacer lo mismo que reduce pero en orden inverso, usaremos **reduceRight**.

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

console.log(a.reduceRight(function (total, num) { // Empieza con el último número y resta los demás
  return total - num;
}));

// Imprime 75 (Si no queremos pasarle un valor inicial, empezará con el valor de la última posición del array)
```

6.9. Dividir strings: split

Esta función es usada para dividir un string en partes usando un carácter delimitador, devolviendo un array con los "trozos". También admite un segundo parámetro opcional que indica cuantos elementos queremos que devuelva.

```
let mensaje = 'Soy un tipo feliz';

// Dividiendo la cadena "mensaje" usando el carácter espacio
let arr = mensaje.split(' ');

// El arreglo
console.log(arr); // ["Soy", "un", "tipo", "feliz"]


// Acceso a cada elemento del arreglo resultante
console.log(arr[0]); // "Soy"
console.log(arr[1]); // "un"
console.log(arr[2]); // "tipo"
console.log(arr[3]); // "feliz"

console.log(mensaje.split(' ', 2)); // ["Soy", "un"]
```

7. Formato JSON.

El formato JSON (https://es.wikipedia.org/wiki/JSON) nos permite definir objetos y objetos de arrray en JavaScript. Después veremos cómo trabajar con ficheros en formato JSON. Çpor ahora estamos interesados en este formato para vector, por ejemplo, podemos definir:

```
let datos = [
{nombre: "Nacho", telefono: "966112233", edad: 40},
{nombre: "Ana", telefono: "911223344", edad: 35},
{nombre: "Mario", telefono: "611998877", edad: 15},
{nombre: "Laura", telefono: "633663366", edad: 17}
];
```

Tenemos un array llamado datos con 4 posiciones. Para acceder a una posición del array: datos [0].

Y para acceder a un propiedad especifica dentro de una posición:

datos[0].edad

También podemos usar cualquier método de los vistos anteriormente con un array en formato json.