Javascript

Sumario

Introducción

Es un lenguaje que se usa especialmente desde el navegador web (lo interpreta el navegador) y su uso es principalmente para la parte de desarrollo del lado del cliente (si bien también se puede usar en el servidor mediante nodejs)

JavaScript y el DOM permite que existan programadores que hagan un uso inapropiado para introducir scripts que ejecuten código con contenido malicioso sin el consentimiento del usuario y que pueda así comprometer su seguridad.

Los desarrolladores de los navegadores tienen en cuenta este riesgo utilizando dos restricciones. En primer lugar, los scripts se ejecutan en un sandbox en el que sólo se pueden llevar a cabo acciones relacionadas con la web, no con tareas de programación de propósito general, como la creación de archivos. En segundo lugar, está limitada por la política del mismo origen: los scripts de un sitio web no tienen acceso a la información enviada a otro sitio web (de otro dominio) como pudiera ser nombres de usuario, contraseñas o cookies. La mayoría de los fallos de seguridad de JavaScript están relacionados con violaciones de cualquiera de estas dos restricciones.

Para mostrar mensajes en la consola usamos:

```
console.log(“mensaje”);
```

si queremos que nos muestre una ventana con el mensaje:

```
alert(“mensaje”);
```

Javascript es un lenguaje funcional. Si bien soporta objetos su origen y uso principal es funcional

Así por ejemplo, cuando queremos declarar una variable y escribimos:

```
for( var i = 0; i < 10; i++){
	console.log(i);
}
console.log(i);
```

observar que la variable tiene un ámbito mayor que los brackets . De hecho su ámbito es el de la función en la que está contenida (si no declarado dentro de una función entonces es global)

hoy en día eso se puede solucionar mediante el uso de let en lugar de var pero ya nos da una idea de que este lenguaje tiene una clara orientación funcional

Continuando con el ámbito, probar en consola:

```
var a = 10;
```

luego miramos lo que contiene a:

```
a
```

vemos que nos mostrará 10

y esto es coincidente con:

```
window.a
```

Esto es porque todo cuelga desde nuestra ventana de trabajo

Tipos de datos

Javascript no está pensado inicialmente para tipado estático (como hacemos con Java que definimos el tipo de la variable antes de usarla) sino que lo deduce de la asignación inicialmente

Así:

```
let num = “hola”;
```

hara que num sea una variable de texto

Si queremos estar seguros que nos está tomando un número podemos hacer uso de la función Number:

let num = Number(5);

aunque también soporta parseint:

```
parseInt(“20€”);
```

devuelve: 20

Null y undefined

Pensemos un momento en las variables locales de un método en Java:
String mivar;
System.out.println(mivar);

El IDE protestará porque la variable no ha sido inicializada
Si en lugar de lo anterior ponemos:
String mivar=null;
System.out.println(mivar);

veremos que muestra null en pantalla

Así pues hay una diferencia entre null y una variable sin inicializar en Java. De forma parecida en Javascript existe: undefined y null

Si nosotros escribimos:

```
let mivar;
console.log(mivar);
```

nos mostrará: “undefined”

Si escribimos:

```
let mivar2=null;
console.log(mivar2);
```

nos mostrará: “null”

y si queremos hacer operaciones con los números tenemos Math:

Math.round(), Math.random(), Math.abs(), Math.pow(), Math.trunc(), ...

también en lugar de usar round podemos redondear un número directamente mediante:

```
 num.toFixed()
```

Manejando strings

Tenemos bastantes cosas parecidas a Java:

```
"Hola".length;      //→  longitud de la cadena

"Hola".charAt(1);  //→ el char de pos 1
```

pero tenemos otras opciones:

```
"Hola".charCodeAt(1)  //  -> da el código ascii
```

también funciona:

```
"Hola"[1]
```

Más sobre trabajo de cadenas:

```
"Animor".indexOf("i");   // -> da la posición del caracter

"Hola amigo".search("am")    //  -> busca la expresión regular y da la posición localizada

"En un lugar de la aancha".replace("aancha","Mancha");
```

Crear cadenas de texto para formatear las variables:

```
`Hola, el valor de a es: ${a}`
```

Expresiones regulares:

```
var re = new RegExp(/d.n/);
re.test("Cadena");
```

devuelve true

Reemplazo en cadenas:

```
str.replace(/o/g, "i");   // <- reemplaza todas las o por i
```

Trabajando con Fechas:

Para crear una fecha tenemos que hacer:

```
let fecha = new Date();  //  --> guarde lla vecha actual.  milisegundos desde 1-1-1970

fecha.getDate();   //→ día mes
fecha.getDay();   // <- día   semana
fecha.getMonth();  // <- mes pero cuidado lo da desde 0 no desde el 1
fecha.getFullYear();  // <- da el año
```

array:

```
var a = new Array();

var a = new Array( 4, 3, 1 );

var a = [];
var a = [ 6, 3, 2 ];
```

Para tener una cantidad específica de posiciones tenemos que hacer:

```
var a = new Array(5);
```

tenemos soporte para una pila:

```
b = a.pop();  <- toma el último elemento y lo borra del array
a.push(4); <- agregas el número al final del array. Te devuelve la posición que ocupa
```

y también para una cola:

```
a.shift()
a.unshift(4);
```

Array de arrays:

```
var a = [ [2, 3], [4, 5, 2]];
```

buscar elemento y tomar indice:

```
a.indexOf()
```

Mapas

Hay soporte para mapas:

```
var m= new Map();
```

podemos hacerlo directamente:

```
var m = new Map([["uno",1], ["dos",2]]);
```

obtenemos por key:

```
m.get("uno");
```

podemos recorrer por las key y obtener el valor:

```
for(k of m.keys()){
  console.log(k,m.get(k));
}
```

Conjuntos (Set)

tenemos el equivalente a un hashset:

```
Set set = new Set();

set.add(2);

set.add(2); // <- no lo agrega
```

diferencia entre == y ===

Observar lo siguiente:

```
var a = 50;
var b = '50';
```

(a == b) // devuelve true el hace una transformación interna---> mira el valor

si se quiere la igualdad que conocemos
(a === b)

El problema de los for:

```
let array = [5,4,3];
for ( i in array ){
   console.log(i);  // va a mostrar los índices: 0, 1, 2. Para que de los valores: array[i]
}
```

pero se puede usar of:

```
for( i of array ){
    console.log(i); // nos da los valores del array
```

```
}
```

Funciones

para crear una función:

```
function func(a,b){
	console.log("hola...");
	console.log("a: " + a + " b: " + b );
	return 3;
}
```

```
// funciones anónimas:
let f = function (){
   console.log("holahola");
} // tener en cuenta que se queda guardada en la variable
```

se puede ejecutar mediante:

```
f()
```

Acceso al DOM

para acceder a los objetos de nuestra página web podemos usar la sintaxis de CSS:

en este ejemplo obtiene nodo con clase: decocat

```
document.querySelector(".decocat")  // cuidado!! devuelve únicamente el primer elemento
```

```
//si queremos que nos de todos hay que usar:
document.querySelectorAll(".decocat");
```

obtener las imagenes dentro de un id: #pagina

```
var elemento = document.querySelectorAll("#pagina img");
```

una buena práctica es tener una clase que defina los puntos del DOM que vamos a usar desde Javascript:

```
const DOM = {
        imgs: document.querySelectorAll("img")
}
```

de esta forma usamos luego:

```
DOM.imgs[0]
```

y estaremos accediendo a la primera imagen del documento

```
//modificando el Html
var element = document.querySelector("#pagina div");

//con innerhtml tenemos el interior del div ( sin las etiquetas div )
element.innerHTML

// y con outer tomamos también las etiquetas div
element.outerHTML


//vamos a modificar el html del div:
element.innerHTML = "<h1> jajaja </h1>"


//vamos a crear elementos y agregarlos al final del documento:
var div = document.createElement("div");
div.textContent = "esto es un texto agregado al div";
div.id = "pieDePagina";
document.body.appendChild(div);
```

Es fácil observar que innerHTML es destructivo. Una forma de evitarlo es mediante insertadjacentHTML:

```
var h1 = document.querySelector("h1");
h1.insertAdjacentHTML("afterbegin","<span>after begin</span>");
```

Veamos ahora como borrar:

```
document.querySelector("span").remove();
```

//actuando directamente sobre el estilo style de html:

```
var sw = document.querySelector("#switch");
sw.style.backgroundColor = "green";
```

//esto se recomienda únicamente para cambios muy pequeños-atómicos (recordar que al tocar
//el atributo style tengo más prioridad que otra opción) Mejor hacer uso de clases CSS:

```
sw.classList //nos da una lista de las clases que tiene el elemento
sw.classList.length // el número de clases cargadas

sw.classList.add("button");  //agregamos una clase más
sw.classList.contains("on"); // busca si tenemos la clase on cargada;
sw.classList.remove("on"); //nos elimina la clase
sw.classList.toggle("on"); // es circular. Primera iteración lo agrega, segundo lo quita, tercera iteración lo agrega, …
```

Clases-Objetos

```
var obj = new Object();
```

//nota Creas un objeto simplemente asociando llaves: var o = {}

```
var obj = {id:4,name:"pepo"};
```

Estos objetos no se crean fijos. Se pueden agregar propiedades a posteriori

//agregando una función al objeto

```
obj.saludar = function(){ console.log("eooo. Soy " + this.name); }
```

Observar que se permite reducir la declaración de las funciones en un objeto:

```
var per = {
   nombre: "luis",
apellidos: "martín",
nombre_completo: function(){
console.log(this.nombre + " " + this.apellidos);
}
};
```

//equivale a:

```
var per = {
   nombre: "luis",
apellidos: "martín",
nombre_completo(){
console.log(this.nombre + " " + this.apellidos);
}
};
```

```
//al igual que en Java cuando igualamos dos variables es únicamente la referencia
var per2 = per;
```

```
//Para copiar un objeto en lugar de la referencia
//funciona únicamente para atributos primitivos
var per2 = Object.assign({},per);
```

El ámbito de this:

```
//el ámbito de this:
function f(){
        return this;
}
//como se ve devuelve: "window"


//Ahora bien si lo hacemos dentro de un ojeto:
var o = {
        get(){
                return this;
        }
}
//así que dentro de un objeto this hace referencia a él mismo. Pero si no es así es la ventana
```

Clases con ES6:

```
//Lo bueno... usar lo que introduce es6 respecto a clases ( es azucar sintáctico )
```

```
class Monster{
	//es6 únicamente permite un constructor por clase
	constructor(){
		this.life = 20;
		this.type = "generic monster";
		this.power = 5;

	}

	//métodos
	talk(){
		return "Grrrr!";
	}

	stats(){
		//las circunflejas nos permite usa $ para agregar al estilo de format
		return `Tipo: ${this.type} \nfida: ${this.life}`;
	}
}

m = new Monster();
m.talk();
```

herencia

//herencia pero usando como ejemplo una nueva clase anónima

```
var Ghost = class extends Monster{
	constructor(){
		super();
		this.life = 40;
		this.type = "ghost";
	}

	talk(){
		return "uuuuuuhhhh!";
	}
}

var g = new Ghost();
```

instanceof

// tenemos instanceof:

```
g instanceof Ghost // true
```

```
g instanceof Monster // true
```