Introducción a JavaScript

Conceptos a aprender

- Cómo ejecutar y probar código JavaScript
- Variables y los tipos de datos
- Operadores básicos
- Bucles
- Funciones
- Objetos
- Arreglos
- Variables modernas

Introducción

JavaScript es un lenguaje relativamente sencillo y fácil de aprender. JavaScript es el lenguaje de programación más usado en el desarrollo de sitios web. Principalmente es usado del lado del cliente, es decir que todo el código escrito es ejecutado directamente por la computadora del usuario, y no por un servidor. Existen varias formas de ejecutar JavaScript con nuestro navegador:

- Directamente en la consola
- Escrito en línea dentro de un documento html
- Incluido en un archivo externo

Para escribir JavaScript directamente en la consola, en Chrome podemos activar las herramientas de desarrollador (F12) y haciendo click en la pestaña ‘Consola’, o ‘Console’. Podemos probar la consola con un código sencillo, como el siguiente:

```
'Hola mundo'
```

Esto ejecutará la cadena de caracteres ‘Hola mundo’. Las cadenas de caracteres son delimitadas por comillas, que pueden ser simples o dobles.

```
1 + 2;
```

Una suma regresará el resultado de la operación, en otras palabras, 3.

Como dijimos antes, también podemos escribir el código JavaScript directamente dentro de un documento html, escrito dentro de las etiquetas <script>. Un ejemplo, sería algo así:

```
<script>
 console.log("Hola mundo");
 console.log(1 + 2);
</script>
```

Para poder mostrar código JavaScript en la consola, necesitamos del método log dentro del objeto console (Es decir, console.log()) y le pasamos como parámetro lo que queremos que se compute. Es decir, el código anterior regresará primero 'Hola mundo' y luego, en otra línea, el número 3.

Para incluir un archivo externo, necesitamos también de la etiqueta <script>, pero pasaremos como parámetro la URL del archivo en cuestión:

```
<script src="/js/scripts.js"></script>
```

Y en el archivo scripts, podemos tener el siguiente código, sin necesidad de etiquetas html:

```
console.log('Hola mundo');
console.log(1 + 2);
```

Y producirá los mismos resultados que en los ejemplos anteriores.

Conceptos básicos

Para comenzar a hacer pruebas basta con abrir nuestro navegador favorito y activar la consola (En Chrome se logra abriendo las herramientas de desarrollador con F12 y haciendo click en la pestaña 'consola').

Tipos de datos

En JavaScript todos los tipos de datos pertenecen a uno de 2 grupos: Primitivo y No Primitivos.

Hay 7 tipos Primitivos:

1. **Boolean**: Unidad lógica que puede ser verdadera o falsa (true / false)
2. **Number**: Se lo usa para todos los números, ya sean de punto flotante o enteros.
3. **String**: Una secuencia de caracteres (letras, palabras, textos)
4. **Undefined**: Es el valor predeterminado de una variable antes de ser definida.
5. **Null**: Similar a Undefined, también significa que es un vacío.

6. **Symbol**: es un objeto incorporado que genera un valor único que no puede ser alterado.
7. **BigInt**: Usado para enteros que el tipo Number no alcanza a cubrir (ya que el número entero más grande que Number puede usar es 9.007.199.254.740.991.

Y 3 tipos de datos del tipo No Primitivo:

1. **Object:** Permiten almacenar múltiples propiedades y valores en una sola variable. Cada propiedad es una pareja clave-valor y puede ser accedida usando la notación de puntos o las corchetes. Los objetos en JavaScript son muy útiles para modelar datos complejos y estructurados.
2. **Array:** Es un objeto que permite almacenar múltiples valores en una sola variable
3. **Function:** Son bloques de código reutilizables que pueden ser invocados repetidamente

A diferencia de otros lenguajes de programación, JavaScript tiene un tipado dinámico, es decir que las variables no tienen un tipo de dato asignado. Esto se puede modificar usando TypeScript.

Declaración de una variable

Una variable se declara con la palabra clave **'var'** seguido del nombre de la variable (que puede tener letras, números, el signo pesos ($), y el guión bajo (_) pero no pueden comenzar con números.

Luego, para terminar de declararla es necesario el signo igual (=) seguido del valor que le asignaremos a la variable

```
var foo = 1;
```

Comentarios

Los comentarios pueden venir en dos formas, en línea o en bloque:

```
//Esto es un comentario en línea, la línea siguiente a éste no estará
comentada a menos que también incluya '//'
/* Esto es un comentario en bloque
Sigue siendo un comentario hasta que lo cerramos */
```

Operadores básicos

```
2 + 2 // Devuelve 4.  El signo + con números, es una suma.
'hola ' + 'mundo' //Devuelve 'hola mundo'. El signo + con cadenas de
```

```
carácteres es una concatenación.
'1' + 2 // Devuelve '12'. Incluso si son valores numéricos, si uno es
una cadena de carácteres (tiene las comillas, ya sean simples o dobles),
el signo + siempre será una concatenación.

3 * 2 //Devuelve 6. El signo * es una multiplicación.
6 / 2 // Devuelve 3. El signo / es una división.
5 % 2 // Devuelve 1. El signo % devuelve el resto de una división (5/2 =
2 y 1 de resto).


var i = 1;
var j = ++i;  // incrementación previa:  j es igual a 2; i es igual a 2
var k = i++;  // incrementación posterior: k es igual a 2; i es igual a
3
```

Operadores lógicos

&&: Operador lógico 'AND'.

```
true && true // es true
true && false // es false
false && false // es false
```

El operador **&&** (AND lógico) devuelve el valor del primer operando si éste es falso; caso contrario devuelve el segundo operando. Cuando ambos valores son verdaderos devuelve verdadero (true), sino devuelve falso (false).

||: Operador lógico 'OR'

```
true || true  //es true
true || false // es true
false || false // es false
```

El operador **||** (OR lógico) devuelve el valor del primer operando, si éste es verdadero; caso contrario devuelve el segundo operando. Si ambos operandos son falsos devuelve falso (false).

Operadores de comparación

- == Comparación de igualdad (1 == 1 devuelve true)

- != Comparación de no-igualdad (1 != 1 devuelve false)
- === Comparación de igualdad estricta (Es decir, la cadena '1' comparándola con el número 1, usando el comparador == devuelve true, pero con el === devuelve false, porque '1' es una cadena y 1 es un número, entonces no son estrictamente iguales).
- !== Comparación de no-igualdad estricta (Es decir, la cadena '1' comparándola con el número 1, usando el comparador != devuelve false, pero usando !== devuelve el true, porque '1' es una cadena y 1 es un número.)
- > Mayor que (1 > 2 devuelve false, 1 > 1 devuelve false, 1 > 0 devuelve true)
- < Menor que (1 < 2 devuelve true, 1 < 1 devuelve false, 1 < 0 devuelve false)
- >= Mayor o igual que (1 >= 2 devuelve false, 1 >= 1 devuelve true, 1 >= 0 devuelve true)
- <= Menor o igual que (1 <= 2 devuelve true, 1 <= 1 devuelve true, 1 <= 0 devuelve false)

Condicionales

El código condicional se ejecuta cuando se cumple una condición. Por ejemplo, usando 'if' es así:

```
if (true) {
 console.log('Es verdadero');
}
```

Es decir, luego de la palabra clave 'if', entre paréntesis se escribe una condición, en este caso es el valor booleano 'true', que evidentemente siempre evalúa como true. Luego, entre llaves, se escribe el código que se ejecutará si la condición es verdadera.

Algo un poco más complicado:

```
var foo = 2;
var bar = 1;
if (foo > bar) {
 console.log('Foo es mayor que Bar');
} else {
 console.log('Foo no es mayor que Bar');
}
```

Como podemos ver, la condición aquí es una operación de comparación, es decir, foo > bar, que en este caso es verdadero (Porque 2 es mayor a 1), y entonces el bloque de código "console.log('Foo es mayor que Bar);" se ejecutará. Si foo fuera menor que bar (digamos, si bar fuera 3), entonces la condición sería falsa, y se ejecutaría el bloque de código "console.log('Foo no es mayor que Bar);" que le sigue al 'else'.

Además del if/else, existe otra declaración similar llamada ‘Switch’. Donde básicamente tomamos un valor y lo comparamos con una serie de valores, ejecutando distintos bloques de código dependiendo del valor de esa variable.

```
var dado = Math.round(Math.random() * 5) + 1;
switch (dado) {
 case 1:
   console.log('Salió 1');
   break;
 case 2:
   console.log('Tenemos 2');
   break;
 case 3:
   console.log('Tres es el 3x1');
   break;
 case 4:
   console.log('El Cuatro está en la cara opuesta al tres');
   break;
 case 5:
   console.log('5 es 1 menos que 6');
   break;
 case 6:
   console.log('Salió el número más alto');
   break;
}
```

Básicamente comenzamos con una variable (En este caso, usando el método Math.random() obtenemos un número aleatorio del 0 al 5, y sumandole 1 lo convertimos en un número aleatorio del 1 al 6. Luego, con las palabras claves switch, case, y break, controlamos el flujo de lo que se ejecutará. Es decir, switch comienza la evaluación, case compara el valor entregado a switch con otro, y break se asegura de separar los bloques de código de switch.

Bucles

Bucle For

El bucle ‘for’ se lo utiliza para iterar código, se lo escribe de la siguiente manera:

```
for ([expresiónInicial]; [condición]; [incrementoDeLaExpresión]) {
 [cuerpo]
}
```

por ejemplo:

```
for (var i = 0; i < 9; i++) {
 console.log(i * (i + 1));
}
```

Ejecutará un código que regresará lo siguiente:

```
0
2
6
12
20
30
42
56
72
```

La variable i comienza en 0, luego se hace la comprobación i < 9 y como 0 < 9, se ejecuta el código dentro del bloque for, luego se ejecuta i++, que colocaría el valor de i en 1, y sigue así hasta que la condición i < 9 sea falsa.

Bucle while

Similar al for, el bucle while se lo utiliza para iterar código, aunque funciona ligeramente diferente, se lo escribe de la siguiente manera:

```
while ([condición]) {
 //código
 [cambio en la condición]
}
```

por ejemplo:

```
var numero = 1;
while (numero < 5) {
 console.log(numero);
 numero = numero + 1;
}
```

Ejecutará un código que regresará lo siguiente:

```
1
2
3
4
5
```

Funciones

Las funciones contienen bloques de código que se pueden ejecutar repetidamente. A las mismas se le pueden pasar argumentos, y opcionalmente la función puede devolver un valor. Si las funciones son parte de un objeto, se les llama ‘métodos’.

Se puede declarar funciones de tres maneras:

```
function foo() { //declaración
 // bloque de código
}
var foo = function () { //expresión
 //bloque de código
}
var foo = () => { //expresión con función de flecha
 //bloque de código
}
```

Función simple

```
var saludar = (persona) => {
 console.log('Hola ' + persona);
}
```

Su uso sería básicamente saludar([nombre de la persona a saludar]), por ejemplo:

```
saludar('mundo'); //mostrará en consola la cadena ‘Hola mundo’
```

Función que devuelve un valor

```
var saludo = (persona) => {
 return 'Hola ' + persona;
}
```

En este caso, al ejecutar el código saludo('mundo') no mostrará nada en consola de por sí, pero podemos guardar la cadena generada 'Hola mundo' en una variable:

```
var saludoMundo = saludo('mundo');
console.log(saludoMundo);
```

O también podemos usar el valor retornado directamente:

```
console.log(saludo('mundo'));
```

Objetos

De cierta manera, los objetos son similares a los vectores. Pero los objetos no están definidos por un índice. pueden contener cero o más conjuntos de pares de nombres claves y valores asociados a dicho objeto. Los nombres claves pueden ser cualquier palabra o número válido.

Ejemplo:

```
var miPrimerObjeto = {
 nombre: 'Juan',
 edad: 23,
 saludar: (persona) => {
   console.log(`${this.nombre} saluda a ${persona}`);
 }
}
```

En este ejemplo, el objeto tiene tres propiedades, ‘nombre’, ‘edad’, y ‘saludar’. Los primeros dos son básicamente variables dentro del objetos, mientras que el tercero es un método del mismo, que podemos llamar de la siguiente manera:

```
miPrimerObjeto.saludar('Viviana');
```

Lo que mostrará en consola el mensaje *‘Juan saluda a Viviana’*.

Cabe destacar que en este caso, en lugar de usar la suma para concatenar las cadenas de caractéres, usamos el acento grave (En teclados latinoamericanos se puede insertar a un texto usando **ALTGR+}**), lo que simplifica bastante el trabajo de tener que cerrar comillas, usar el +, abrir comillas de nuevo y así.

Los otros valores se obtienen de la misma manera, por ejemplo:

```
if (miPrimerObjeto.edad > 18) {
```

```
 console.log('Es mayor de edad.');
}
console.log(miPrimerObjeto.nombre);
```

Arreglos

Un arreglo, también conocido como ‘vector’, ‘matriz’, o ‘array’ (su nombre en inglés), es una lista de valores con índice cero, es decir, el primer valor de la lista será el 0, y no el 1.

Ejemplo:

```
var miArray = ['hola', 'mundo'];
console.log(miArray[0]);
console.log(miArray[1]);
```

En este ejemplo creamos una lista que sería:

```
miArray[0] == 'hola';
miArray[1] == 'mundo';
```

Cuando hacemos un console.log a ‘miArray[0]’ estamos obteniendo el valor en el índice 0 del vector, o sea ‘hola’.

Podemos obtener la cantidad de elementos que tiene una matriz mediante la propiedad ‘length’

```
console.log(miArray.length); // Devolverá 2, porque tiene dos elementos.
```

Pero tenemos que tener cuidado, porque ‘length’ devuelve un número más que el último índice. Es decir, si tenemos algo así:

```
var frutas = ['naranja', 'manzana', 'pomelo', 'banana', 'ananá'];
```

El último índice es 4 (con un valor igual a la cadena ‘ananá’. Pero frutas.length es igual a 5:

```
console.log(frutas.length); // será 5
```

Sabiendo esto, podemos usar el bucle for de la siguiente manera:

```
for (var i = 0; i < frutas.length; i++) {
 console.log(frutas[i]);
```

```
}
```

Es decir, inicializamos la variable ‘i’ en 0, y luego como condición establecemos que i es menor a la cantidad de elementos en el vector ‘frutas’ (en nuestro caso, 5), e iterar el valor frutas[i] hasta que esa condición no se cumpla (cuando i es igual a 5, se cumple!), entonces i iterará en 0, 1, 2, 3, 4, pero no en 5.

Podemos cambiar el valor de uno de los índices de la matriz:

```
frutas[2] = 'sandía';
```

Esto reemplazará el valor de frutas[2] (Que era ‘pomelo’, por el nuevo valor ‘sandía’).

También podemos agregar un nuevo elemento al vector, usando el método ‘push’:

```
frutas.push('pomelo');
```

Map

Map es un método de los arreglos que nos permite ‘mapear’ el contenido del mismo y obtener datos acotados. Por ejemplo, dado el siguiente arreglo:

```
var usuarios = [
 { nombre: 'Carlos', edad: 55 },
 { nombre: 'Juana', edad: 60 },
 { nombre: 'Martina', edad: 27 },
 { nombre: 'Oscar', edad: 30 }
];
```

Si queremos obtener los nombres de todos los usuarios, necesitamos que los datos se vean en un solo arreglo, así:

```
['Carlos', 'Juana', 'Martina', 'Oscar']
```

Para llegar a esos datos, usamos map:

```
var nombresUsuarios = usuarios.map((usuario) => {
 return usuario.nombre
});
```

Esto guardará de manera sencilla en la variable “nombresUsuarios” los datos que esperabamos:

```
['Carlos', 'Juana', 'Martina', 'Oscar']
```

Reduce

Digamos que partiendo del ejemplo anterior, lo que la aplicación requiere es la suma de todas las edades de los usuarios.

Para eso, podríamos usar un bucle, como for y sumar de a uno cada edad hasta llegar al total.

Una forma más rápida es usando reduce.

```
var sumaEdades = usuarios.reduce((acumulador, usuario) => {
 return acumulador + usuario.edad;
}, 0);
```

El método “reduce” pide dos valores, primero una función a ejecutarse y luego un valor inicial, que es el acumulador y en este caso lo comenzamos en 0.

Filter

Siguiendo con el ejemplo anterior, tal vez lo que queramos hacer es filtrar los elementos del arreglo, por ejemplo, obtener todos los usuarios cuya edad sea de 30 o menor.

```
var menoresDe30 = usuarios.filter((usuario) => {
 return usuario.edad <= 30;
});
```

Esto nos guardará en la variable “menoresDe30” el arreglo con los usuarios que cumplen esta condición (Martina y Oscar)

```
[
 { nombre: 'Martina', edad: 27 },
 { nombre: 'Oscar', edad: 30 }
]
```

Más allá del var

Además de **var**, desde la salida de ES2015 existen también **let** y **const**.

El alcance de var

El **alcance** nos dice las desde donde puede ser accedida una pieza de código. Puede ser un alcance **global** (es decir, se puede acceder desde cualquier punto del código) o **local** (solo podrá ser accedido desde el bloque de código desde donde fue generado)

El alcance de **var** puede variar dependiendo donde es declarada. Si es declarada fuera de una función, tendrá un alcance global, es decir, podrá ser usada en cualaquier lugar de nuestro código. Si la función var, en cambio, es declarada dentro de un bloque de función, solo tendrá un alcance dentro de esa función, es decir, si intentamos usar una variable declarada en una función, fuera de esa función, tendrémos un error.

```
var saludo = "Hola mundo";
function saludarYPrepararDespedida() {
 console.log(saludo); //se ejecutará correctamente, ya que la variable
"saludo" tiene un alcance global.
 var despedida = "Nos vemos luego"
}
console.log(despedida); //¡esto devuelve un error!
```

Hoisting

Hoisting es un mecanismo interno de JavaScript que se realiza durante la fase de compilación del código donde se aloja en memoria las variables y funciones. Esto nos permite ejecutar una función antes de declararla si quisieramos; sin embargo, con las variable no pasa lo mismo ya que al asignarseles memoria el valor por defecto es undefined.

Posterior a la etapa de hoisting, JavaScript ejecuta el código asignando valores a variables que ya tiene previamente en memoria.

```
var h = 'Hola';
console.log(h + " " + m); //Devolverá "Hola undefined"
var m = 'Mundo';
```

Redeclaración de una variable

Otro detalle de var es que puede ser re-declarada sin generar ningún error.

```
var saludo = "Hola mundo";
var saludo = "Te doy la bienvenida"; //¡No genera error!
```

Esto en un bloque de código pequeño no causa problemas, pero en una aplicación mediana ya puede generar mucha confusión, particularmente si ambas var son declaradas con mucha ‘distancia’ una de la otra.

Let

A diferencia de var, let siempre tiene un alcance local. Es decir, solo afectará lo que suceda en el bloque en el que está. Si generamos una ‘copia’ de ese let, el nuevo let y el anterior co-existirán ya que tienen alcances diferentes. Además, si intentamos usar un let fuera de ese alcance, generará un error.

```
let saludado = "mundo";
let dia = "Lunes"
if ("Lunes" === dia ) {
 let saludado = "semana";
 let finDeSemana = false;
 console.log(`Bienvenida la nueva ${saludado}`); //Devolverá "Bienvenida
la nueva semana"
 console.log(finDeSemana); //Mostrará “false”
}
console.log(`Hola ${saludado}`); //Devolverá "Hola mundo"
console.log(finDeSemana); //¡Generará un error!
```

En este ejemplo podemos ver primero, cómo la variable “saludo” es definida inicialmente como “mundo” y dentro del bloque if como “semana”. Sin embargo, como se usa “let” para definir la variable, ambas variables tienen alcances diferentes. El primer console.log se ejecuta correctamente con el valor definido dentro del bloque if y el segundo se ejecuta correctamente con el valor definido inicialmente. Además la segunda variable “finDeSemana” que se define dentro del bloque if, no puede ser accedida fuera de éste.

Más allá de estas diferencias, let también genera un error cuando, en el mismo alcance, se intenta volver a definir. Otra diferencia es que el Hoisting de let funciona ligeramente diferente al de var, ya que su valor no se inicializa como “undefined”. Si se intenta usar una variable let antes de ser declarada, se generará un error.

Const

Las “variables” delcaradas con la palabra clave “const” son en esencia iguales a las “let” con la diferencia de que NO pueden ser re-definidas luego de ser declaradas.

En general, es recomendable usar siempre const a menos que estemos seguros de que la variable podrá cambiar de valor, en cuyo caso, lo recomendable es usar let.

Próximos pasos

- Manipulación del DOM
- Manejo de clases y programación orientada a objetos
- Promesas y código asíncronico
- Typescript