Módulo 2: JavaScript

AUTOR: ABREU, Pablo

Índice

Introducción

Según Mozilla (MDN 2020) , JavaScript, o simplemente JS, es un lenguaje de scripting multiplataforma y orientado a objetos. Es un lenguaje pequeño y liviano. Dentro de un ambiente de host, JavaScript puede conectarse a los objetos de su ambiente y proporcionar control programático sobre ellos. Se trata de un **lenguaje de programación tipo script**, basado en objetos y guiado por eventos, diseñados específicamente para el desarrollo de aplicaciones cliente-servidor dentro del ámbito de Internet. Los programas escritos con JavaScript se pueden probar directamente en cualquier navegador sin necesidad de procesos intermedios. JavaScript es un lenguaje de programación interpretado, por lo que no es necesario compilar los programas para ejecutarlos.

Objetivos

El objetivo de este tutorial es introducir el concepto de programación estructurada y el lenguaje de programación interpretado JavaScript. Nos enfocaremos en JavaScript del lado del cliente, lo cual proporciona además una serie de objetos para controlar un navegador y su modelo de objetos (o DOM, por las iniciales en inglés de Document Object Model). Por ejemplo, las extensiones del lado del cliente permiten que una aplicación coloque elementos en un formulario HTML y responda a eventos del usuario, tales como clics, ingreso de datos al formulario, navegación de páginas, etc.

Historia

¿Por qué se llama Javascript?, según Crockford (Crockford, 2020), el prefijo de Java sugiere que Javascript está de algún modo relacionado con Java, que es un subconjunto de instrucciones de Java o que es una versión menos potente. (...) “Parece que el nombre fue intencionalmente seleccionado para crear confusión y de esa confusión surge la mala interpretación” (...) .

Javascript es un lenguaje diferente a Java. Si bien Javascript tiene una sintaxis similar a Java no significa que tenga algo que ver con Java. Inclusive Java tiene una sintaxis similar al lenguaje C y esto no quiere decir que Java sea C. El lenguaje C es otro lenguaje de programación.

Javascript fue creado por Brendan Eich en 1995 para la empresa (y navegador popular en su época) Netscape (...) (Ben Aston, Medium, 2015). Como Java era un lenguaje nuevo en esa era y con mucha aceptación, se optó por utilizar parte del nombre para ganar adeptos. Brendan Eich, 2016).
Por último y para cerrar este apartado, vamos a decir que Javascript está estandarizado en Ecma International (asociación europea para la creación de estándares para la comunicación y la información) con el fin de ofrecer un lenguaje de programación estandarizado e internacional basado en Javascript y llamado ECMAScript para que todos los navegadores puedan realizar un intérprete que acepte la misma sintaxis de Javascript.

Programación estructurada

La programación estructurada es un paradigma de programación basado en utilizar funciones o subrutinas, y únicamente tres estructuras de control:

- secuencia: ejecución de una sentencia tras otra.
- selección o condicional: ejecución de una sentencia o conjunto de sentencias, según el valor de una variable booleana.
- iteración (ciclo o bucle): ejecución de una sentencia o conjunto de sentencias, mientras una variable booleana sea verdadera.

Este paradigma se fundamenta en el teorema correspondiente, que establece que toda función computable puede ser implementada en un lenguaje de programación que combine sólo estas tres estructuras lógicas o de control. La estructura de secuencia es la que se da naturalmente en el lenguaje, ya que por defecto las sentencias son ejecutadas en el orden en que aparecen escritas en el programa. Para las estructuras condicionales o de selección, Javascript habilita la sentencia **if** que puede combinarse con sentencias else if y/o else. Para los bucles o iteraciones existen las estructuras **while** y **for**.

La programación estructurada también sigue el principio de "divide y vencerás" y/o buscar reducir el problema en problemas más pequeños o modulares.

Entre las ventajas de la programación estructurada sobre el modelo anterior (hoy llamado despectivamente código espagueti), cabe citar las siguientes: Los programas son más fáciles de entender, pueden ser leídos de forma secuencial y no hay necesidad de tener que rastrear saltos de líneas (GOTO) dentro de los bloques de código para intentar entender la lógica interna. La estructura de los programas es clara, puesto que las sentencias están más ligadas o relacionadas entre sí. Se optimiza el esfuerzo en las fases de pruebas y depuración. El seguimiento de los fallos o errores del programa (debugging[1]), y con él su detección y corrección, se facilita enormemente. Se reducen los costos de mantenimiento. Análogamente a la depuración, durante la fase de mantenimiento, modificar o extender los programas resulta más fácil. Los programas son más sencillos y más rápidos de confeccionar. Se incrementa el rendimiento de los programadores.[2]

Versiones

Versión	Nombre oficial	Descripción
ES1	ECMAScript 1 (1997)	Primera edición
ES2	ECMAScript 2 (1998)	Cambios editoriales
ES3	ECMAScript 3 (1999)	Expresiones regulares Añadido try / catch, switch , do-while
ES4	ECMAScript 4	Nunca lanzado
ES5	ECMAScript 5 (2009)	Añadido "strict mode", soporte JSON Añadido String.trim (), Array.isArray () Añadido métodos de iteración de arreglos. Permite comas posterior para objetos literales
ES6	ECMAScript 2015	Añadido let y const Añadido valor de parámetros por defecto. Añadido Array.find() Añadido Array.findIndex().

[1] https://code.visualstudio.com/docs/editor/debugging
[2] https://entrenamiento-python-basico.readthedocs.io/es/latest/leccion5/programacion_estructurada.html

	ECMAScript 2016	Añadido operador exponencial (**) Añadido Array.includes ()
	ECMAScript 2017	Añadido relleno de cadenas Añadido Object.entries(), Object.values Añadido funciones asincrónicas (async) Añadido memoria compartida
	ECMAScript 2018	Añadido propiedades rest / spread Añadido iteración asincrónica Añadido Promise.finally() Se agregaron RegExp

Nota: A partir de la versión ES6 / ECMAScript 2015 se comenzó a utilizar el nombre con el año consecutivo. Es por eso que a partir de esa versión se llama oficialmente "ECMAScript XXXX", donde XXXX es el año correspondiente a la versión de Javascript.

Sintaxis

A continuación vamos a ver la sintaxis del lenguaje JS (JavaScript) para familiarizarnos con la forma en que se codifica. Antes de comenzar con apartados más específicos veremos cuestiones generales sobre el lenguaje.

Javascript es un lenguaje case-sensitive, o sea que distingue entre mayúsculas y minúsculas. Por ejemplo identificador (o nombre) "OnClickEvent" es distinto a "onclickevent". Esta característica hace que prestemos mucha atención a la forma correcta de llamar usar y/o llamar los identificadores. Las instrucciones son llamadas sentencias y son separadas por punto y coma (;) al final de la misma, si bien no es necesario en todos los casos es una buena práctica.

Existen dos formas de realizar comentarios: MultiLine o SingleLine. Para el caso de multilínea se utilizan los caracteres "/*" y "*/" por ejemplo:

```
/* Esto es un comentario
     multilínea */
```

En el caso de un comentario de línea solamente es necesario usar los caracteres "//" al inicio de la línea a comentar por ejemplo:

```
// Esto es un comentario de una sola línea
```

Por último debemos tener en cuenta que como cualquier otro lenguaje de programación, JS también usa palabras reservadas (reserved words/ keywords) que no podrán utilizadas por los programadores para usarlas como identificadores. En ejemplo de ellas son: ***await, break, case, catch, class, const, continue, debugger, default, delete, do, else, enum, export, extends, false, finally, for, function, if, import, in, instanceof, new, null, return, super, switch, this, throw, true, try, typeof, var, void, while, with, yield*** *, etc.*

Formas de referenciar

En esta sección vamos a aprender cómo incluir código Javascript dentro de nuestras páginas web. En este instante es necesario aclarar que nos enfocaremos en JavaScript del lado del cliente (client-side en inglés) puesto que es el navegador quien tiene la responsabilidad de interpretar el código de la aplicación web. Actualmente y de manera muy creciente existen otras formas de uso del lenguaje como de lado del servidor, las cuales no son del alcance de este documento.

Adicionalmente no existe una manera específica para referenciar código JS dentro de un documento HTML, es decir, podemos hacerlo tanto en el **<head>** como en el **<body>**. Para el uso adecuado dentro de nuestros proyectos es recomendable referencial al final del <body> justo antes del cierre etiqueta. Si bien existen otras teorías y documentos donde se trata de forma avanzada este apartado a continuación ejemplificamos nuestra propuesta.

1. **Dentro de etiquetas <script type="text/javascript">...<script>**

```
<!DOCTYPE html>
<html lang="es">
    <head>
        <meta charset="utf-8" />
        <title>Referenciando JS en HTML5</title>
    </head>
    <body>
        <section>
            <button id="btn_alerta">Click aquí</button>
        </section>
        <script type="text/javascript">
            document.getElementById('btn_alerta').onclick = function(){
                alert('Hola Mundo!');
            }
        </script>
    </body>
</html>
```

2. **Link a un archivo externo <script type="text/javascript" src="..."><script>**

Forma recomendada para el uso de Javascript dentro de nuestras páginas web. El uso de archivos externos permite separar el código HTML y JS de haciendo más clara su posterior lectura y mantenimiento.

```html
<!DOCTYPE html>
<html lang="es">
    <head>
        <meta charset="utf-8" />
        <title>Referenciando JS en HTML5</title>
    </head>
    <body>
        <section>
            <button id="btn_alerta">Click aquí</button>
        </section>
        <script type="text/javascript" src="./index.js"></script>
    </body>
</html>
```

Tipos de datos

El último estándar ECMAScript define nueve tipos:

- Seis tipos de datos primitivos, controlados por el operador **typeof**
    - **Undefined**: typeof instance === "undefined"
    - **Boolean**: typeof instance === "boolean"
    - **Number**: typeof instance === "number"
    - **String**: typeof instance === "string"
    - **BigInt**: typeof instance === "bigint"
    - **Symbol**: typeof instance === "symbol"

- **Null**: typeof instance === "object". Tipo primitivo especial que tiene un uso adicional para su valor: si el objeto no se hereda, se muestra null.

- **Object**: typeof instance === "object". Tipo estructural especial que no es de datos pero para cualquier instancia de objeto construido que también se utiliza como estructuras de datos: new Object, new Array, new Map, new Set, new WeakMap, new WeakSet, new Date y casi todo lo hecho con la palabra clave new

- **Function**: una estructura sin datos, aunque también responde al operador typeof: typeof instance === "function". Esta simplemente es una forma abreviada para funciones, aunque cada constructor de funciones se deriva del constructor Object.

Nota: El único propósito valioso del uso del operador typeof es verificar el tipo de dato. Si deseamos verificar cualquier Tipo Estructural derivado de Object, no tiene sentido usar typeof para eso, ya que siempre recibiremos "object". La forma correcta de comprobar qué tipo de Objeto estamos usando es la palabra clave **instanceof**. [3]

[3] https://developer.mozilla.org/es/docs/Web/JavaScript/Data_structures

Variables

El concepto de variable nos va a ser útil durante todo el proceso de programación en los distintos lenguajes. Javascript implementa el mismo concepto: Las variables son un espacio designado en la memoria para almacenar un dato. Se le asocia un nombre que la identifica y un tipo de datos.

Ejemplo de declaración y asignación de valor.

```
var mi_dato_numerico = 3;
var mi_dato_texto = "hola";
```

Declaraciones:

En JS existen tres tipos de declaraciones de variables.

Usando **var:** declara una variable, inicializándola opcionalmente a un valor. Si bien esta ha sido exclusivamente la forma de declarar variables durante mucho tiempo, sufre de unos efectos no deseados en el alcance (scope) de la declaración. En otras palabras, se aconseja no continuar utilizándose ya que el nuevo estándar de javascript permite otras formas de declaración que corrigen este problema.

El alcance o bloque de ámbito es el lugar en donde es visible esa variable. Por ejemplo, si se declara una variable dentro de una función, esta variable mantiene su valor y es visible sólo dentro de una función.

Usando **let:** declara una variable local en un bloque de ámbito (scope), inicializándola opcionalmente a un valor.

Usando **const:** declara una constante de sólo lectura en un bloque de ámbito. Una constante funciona como una variable pero no cambia su valor, es decir, representa un lugar de almacenamiento de tipos de datos en la memoria pero una vez asignado el valor inicial este no puede ser modificado. La sintaxis de la definición del identificador es la misma que la de las variables.

Las variables se usan como nombres simbólicos para valores en tu aplicación. Los nombres de las variables, llamados identificadores, se rigen por ciertas reglas. Un identificador en JavaScript tiene que empezar con una letra, un guión bajo (_) o un símbolo de dólar ($).Los valores subsiguientes pueden ser números. Debido a que JavaScript diferencia entre mayúsculas y minúsculas, las letras incluyen tanto desde la "A" hasta la "Z" (mayúsculas) como de la "a" hasta la "z". Por Ejemplo:

```
var_numero = 100;
var numero = 100;
let $numero = 100;
const numeroPar = 100;
const PI = 3.14;
```

Ámbito

A partir de la versión ES6 / ECMAScript 2015 Javascript habilita tres ámbitos para la declaración de una variable:

Ámbito **Global**: Cuando se declara una variable fuera de una función, se le denomina variable global, porque está disponible para cualquier otro código en el documento actual.

Ámbito **Bloque:** Este nuevo ámbito es utilizado para las variables declaradas con la palabra reservada **let** / **const** dentro de un bloque de código delimitados por llaves { }, esto implica que no pueden ser accesibles fuera de este bloque.

Ámbito **Función**: Cuando se declara una variable dentro de una función, se le denomina variable local, porque está disponible sólo dentro de esa función donde fue declarada.

Operadores

Los operadores en los lenguajes de programación son símbolos que indican cómo se deben manipular los operadores. Los operadores en conjunto con los operandos forman una expresión, una fórmula que define el cálculo de un valor. Los operandos pueden ser constantes, variables o llamadas a funciones.

Operadores Matemáticos o Aritméticos

Operador		Descripción	Ejemplo
+	Suma	Suma dos números	let sum=5+7; //=13
-	Resta	Resta dos números	let resta=7-2; //=5
*	Multiplicación	Multiplica dos números	let mul=5*7; //=35

/	División	Divide dos números	let div=15/3; //= 5
%	Módulo	Devuelve el resto de dividir de dos números	let mod=17%4; //=1
++	Incremento	Suma 1 valor al contenido de una variable	let i=1; i++; //=2
--	Decremento	Resta un valor al contenido de una variable	let i=2; let i--; //=1

Operadores de Comparación

Operador		Descripción
==	Igualdad	Devuelve verdadero (true) si ambos operandos son iguales.
!=	Desigualdad	Devuelve verdadero (true) si ambos operandos no son iguales.
>=	Mayor que	Devuelve verdadero (true) si el operando de la izquierda es mayor o igual que el operando de la derecha.
>	Mayor	Devuelve verdadero (true) si el operando de la izquierda es mayor que el operando de la derecha.
<	Menor	Devuelve verdadero (true) si el operando de la izquierda es menor que el operando de la derecha.
=<	Menor que	Devuelve verdadero (true) si el operando de la izquierda es menor o igual que el operando de la derecha

Operadores Lógicos

Operador	USO	Descripción
AND (&&) Y lógico	expr1 && expr2	Devuelve expr1 si se puede convertir a false; de lo contrario, devuelve expr2. Por lo tanto, cuando se usa con valores booleanos, && devuelve true si ambos operandos son true; de lo contrario, devuelve false.
OR(||) O lógico	expr1 || expr2	Devuelve expr1 si se puede convertir a true; de lo contrario, devuelve expr2. Por lo tanto, cuando se usa con valores booleanos, || devuelve true si alguno de los operandos es true; si ambos son falsos, devuelve false.
NOT (!) NO lógico	!expr	Devuelve false si su único operando se puede convertir a true; de lo contrario, devuelve true.

Operador condicional (ternario)

El operador condicional es el único operador de JavaScript que necesita tres operadores. El operador asigna uno de dos valores basado en una condición. La sintaxis de este operador es: condición ? valor1 : valor2. Si la condición es true, el operador tomará el valor 1, de lo contrario tomará el valor 2. Se puede usar el operador condicional en cualquier lugar que use un operador estándar. Ejemplo: esta sentencia asigna el valor adulto a la variable estado si la edad es mayor o igual a 18, de lo contrario le asigna el valor menor.

```
var estado = (edad >= 18) ? "adulto" : "menor";
```

Switch

Una sentencia switch permite a un programa evaluar una expresión e intentar igualar el valor de dicha expresión a una etiqueta de caso (case). Si se encuentra una coincidencia, el programa ejecuta la sentencia asociada. Una sentencia switch se describe como se muestra a continuación:

```
switch (expresión) {
    case etiqueta_1:
        sentencias_1
        [break;]
    case etiqueta_2:
        sentencias_2
        [break;]
    ...
    default:
        sentencias_por_defecto
        [break;]
}
```

Primero busca una cláusula case con una etiqueta que coincida con el valor de la expresión y entonces, transfiere el control a esa cláusula, ejecutando las sentencias asociadas a ella. Si no se encuentran etiquetas coincidentes, busca la cláusula opcional default y transfiere el control a esa cláusula ejecutando las sentencias asociadas. Si no se encuentra la cláusula default, el programa continúa su ejecución por la siguiente sentencia al final del switch. Por convención la cláusula por defecto es la última cláusula aunque no es necesario que sea así.

La sentencia opcional break asociada con cada cláusula case asegura que el programa finaliza la sentencia switch una vez que la sentencia asociada a la etiqueta coincidente es ejecutada y continúa la ejecución por las sentencias siguientes a la sentencia switch. Si se omite la sentencia break, el programa continúa su ejecución por la siguiente sentencia que haya en la sentencia switch.

Funciones

Las funciones son uno de los bloques de construcción fundamentales en JavaScript. Una función en JavaScript es similar a un procedimiento — un conjunto de instrucciones que realiza una tarea o calcula un valor, pero para que un procedimiento califique como función, debe tomar alguna entrada y devolver una salida donde hay alguna relación obvia entre la entrada y la salida. Para usar una función, debes definirla en algún lugar del ámbito desde el que deseas llamarla.[4]

La declaración de una función consiste:

1. Un nombre
2. Una lista de parámetros o argumentos encerrados entre paréntesis.
3. Conjunto de sentencias JavaScript encerradas entre llaves.

```
function nombre (parámetro1, parámetro2)
{
    //código ha ser ejecutado;
}
```

En resumen, es un conjunto de instrucciones o sentencias que se agrupan para realizar una tarea concreta y que se pueden reutilizar fácilmente. Realizan varias operaciones invocando un nombre. Esto permite simplificar el código pudiendo crear nuestras propias funciones y usarlas cuando sea necesario.

[4] https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Functions

Estructuras repetitivas

Las estructuras repetitivas o cíclicas nos permiten ejecutar varias veces un conjunto de instrucciones. A estas repeticiones se las conoce con el nombre de ciclos o bucles.

Los lenguajes de programación son muy útiles para completar rápidamente tareas repetitivas, desde múltiples cálculos básicos hasta cualquier otra situación en donde tengas un montón de elementos de trabajo similares que completar. Aquí vamos a ver las estructuras de bucles disponibles en JavaScript que pueden manejar tales necesidades[5]

Estructuras repetitivas en JavaScript:

- FOR
- WHILE
- DO WHILE

Sentencia for

Un bucle **for** se repite hasta que la condición especificada se evalúa como **false**.

```
for ([expresion-inicial]; [condicion]; [expresion-final]) {
    //código ha ser ejecutado;
}
```

Cuando un bucle **for** se ejecuta, ocurre lo siguiente: la **[expresion-inicial]**, si existe, se ejecuta. Esta expresión habitualmente inicializa uno o más contadores del bucle, pero la sintaxis permite una expresión con cualquier grado de complejidad. Esta expresión puede también declarar variables. Se evalúa la expresión **[condicion]**. Si el valor de condición es **true**, se ejecuta la sentencia del bucle. Si el valor de condición es **false**, el bucle for finaliza. Si la expresión condición es omitida, la condición es asumida como verdadera. Se ejecuta la **[expresion-final]**, si hay una, y el control vuelve evaluar la expresión **[condicion].**

El ejemplo a continuación muestra un ciclo que se ejecuta 10 veces e imprime por consola la expresión “Número: X” donde x va del valor 0 al 9.

```
for (var i = 0; 1 < 10; i++) {
    console.log("Número: " + i);
}
```

[5] https://developer.mozilla.org/es/docs/Learn/JavaScript/Building_blocks/Looping_code

Sentencia while

Una sentencia **while** ejecuta sus sentencias mientras la condición sea evaluada como verdadera. Una sentencia **while** tiene el siguiente aspecto:

```
while ([condicion]) {
    //código ha ser ejecutado;
}
```

Si la condición cambia a falsa, la sentencia dentro del bucle deja de ejecutarse y el control pasa a la sentencia inmediatamente después del bucle. La condición se evalúa antes de que la sentencia contenida en el bucle sea ejecutada. Si la condición devuelve verdadero, la sentencia se ejecuta y la condición se comprueba de nuevo. Si la condición es evaluada como falsa, se detiene la ejecución y el control pasa a la sentencia siguiente al **while**.

Sentencia do-while

Se utiliza para repetir instrucciones un número indefinido de veces, hasta que se cumpla una condición. A diferencia de la estructura mientras (while), la estructura hasta (do while) se ejecutará al menos una vez. Ejemplo:

```
let result = '';
let i = 0;

do {
  i = i + 1;
  result = result + i;
} while (i < 5);

console.log(result);
// resultado esperado: "12345"
```

Sentencia continue

La sentencia **continue** puede usarse para reiniciar una sentencia **for, while** o **do-while**, **continue** termina la iteración en curso del código y continúa la ejecución del bucle con la siguiente iteración. A diferencia de la sentencia **break**, **continue** no termina completamente la ejecución del bucle.

```
do {
  i = i + 1;
  continue;
} while (i < 5);
```

Sentencia break

La sentencia **break** se utiliza para salir de un bucle, switch. Break finaliza inmediatamente el código encerrado en un **for**, **while**, **do-while** o **switch** y transfiere el control a la siguiente sentencia.

```
do {
  i = i + 1;
  if(i == 3) break;
} while (i < 5);
```

Búsquedas

Existen diferentes métodos que se pueden usar en JavaScript para buscar elementos dentro de un arreglo. El método a elegir depende del caso de uso particular, por ejemplo:

- Obtener todos los elementos del arreglo que cumplen una condición específica. (Filter)
- Obtener al menos uno de los elementos del arreglo cumple dicha condición. (Find)
- Obtener si un valor específico hace parte del arreglo (Includes)
- Obtener el índice de un valor específico (IndexOf).

Array.filter()

Podemos usar el método Array.filter() para encontrar los elementos dentro de un arreglo que cumplan con cierta condición. Por ejemplo, si queremos obtener todos los elementos de un arreglo de números que sean mayores a 10, podemos hacer lo siguiente:

```
let arreglo = [10, 11, 3, 20, 5];
let mayorQueDiez = arreglo.filter(element => element > 10);
console.log(mayorQueDiez) // resultado esperado: [11, 20]
```

Array.find()

Usamos el método Array.find() para encontrar el primer elemento que cumple cierta condición. Tal como el método anterior, toma un Callback como argumento y devuelve el primer elemento que cumpla la condición establecida. Usemos el método find en el arreglo del ejemplo anterior.

```
let arreglo = [10, 11, 3, 20, 5];
let existeElementoMayorQueDiez = arreglo.find(element => element > 10);
console.log(existeElementoMayorQueDiez) // resultado esperado: 11
```

Array.includes()

El método includes() determina si un arreglo incluye un valor específico y devuelve verdadero o falso según corresponda. En el ejemplo anterior, si queremos revisar si 20 es uno de los elementos del arreglo, podemos hacer lo siguiente:

```
let arreglo = [10, 11, 3, 20, 5];
let incluyeVeinte = arreglo.includes(20);
console.log(incluyeVeinte) // resultado esperado: true
```

Array.indexOf()

El método indexOf() devuelve el primer índice encontrado de un elemento específico. Devuelve -1 si el elemento no existe en el arreglo. Volvamos a nuestro ejemplo y encontremos el índice de 3 en el arreglo.

```
let arreglo = [10, 11, 3, 20, 5];
let indiceDeTres = arreglo.indexOf(3);
console.log(indiceDeTres) // resultado esperado: 2
```

Búsqueda de Máximos y MÍnimos

Uno de los problemas académicos más comunes es el de la búsqueda del valor máximo o mínimo dentro de una lista. JavaScript dispone de las funciones Math.max() y Math.min() con las que es posible obtener el máximo y mínimo respectivamente de un conjunto de números, por ejemplo:

```
Math.max(1, 2, 3, 4, 5); // resultado esperado: 5
Math.min(1, 2, 3, 4, 5); // resultado esperado: 1
```

El problema de estas funciones es que no permiten entradas de tipo array, solamente de tipo numérico. Normalmente se puede solucionar empleando diferentes aproximaciones como son los métodos reduce() o apply(). La forma más fácil de aplicar una función a un array es utilizando el método apply(). Simplemente se tiene que aplicar apply() a la función pasando como primer parámetro null y como segundo parámetro el array. Así se puede obtener el máximo o mínimo de un array simplemente con el siguiente código.

```
Math.max.apply(null, values) // resultado esperado: 5
Math.min.apply(null, values) // resultado esperado: 1
```

Búsqueda de Secuencial

La búsqueda secuencial se define como la búsqueda en la que se compara elemento por elemento del vector/array con el valor que buscamos. Es decir, un clásico recorrido

secuencial **(for)**. De hecho, tenemos varias maneras de implementarlo, pero más allá de eso, el principio es el mismo. Comparar elemento por elemento hasta encontrar el/los que buscamos. Este tipo de búsqueda en el peor de sus casos ejecuta las instrucciones del loop n veces, es decir es la cantidad de elementos del arreglo X(n). En el mejor de los casos, el primer elemento del arreglo es el que elemento que estamos buscando por eso para ese caso es pasaría a ser X(1).

Veamos un ejemplo de una función que implementa búsqueda secuencial en un arreglo:

```
// Devolverá el índice donde encontró al elemento. Recibe el valor a
buscar y el arreglo donde buscará
function sequentialSearch(element, array){
  for (var i in array){
    if (array[i] == element) return i;
  }
  return -1;
}

var letters = ["a", "b", "c", "d", "f", "g", "h", "i", "j", "k", "l",
"m", "n"];
sequentialSearch("g",letters);
```

Ordenamiento

Javascript provee un método que ordena los elementos de un arreglo localmente y devuelve el arreglo ordenado Array.prototype.sort*([compareFunction(a, b)])*. El modo de ordenación por defecto responde a la posición del valor del string de acuerdo a su valor en el juego de caracteres Unicode. Cuando se utiliza el método sort(), los elementos se ordenarán en orden ascendente (de la A a la Z) por defecto:[6]

```
const equipos = ['Real Madrid', 'Manchester Utd', 'Bayern Munich',
'Juventus'];
equipos.sort();
// ['Bayern Munich', 'Juventus', 'Manchester Utd', 'Real Madrid']
```

[6] https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array/sort

Dada esta característica el ordenar números pasa a ser una tarea no tan simple. Si aplicamos el método sort() directamente a un arreglo de números, veremos un resultado inesperado:

```
const numeros = [3, 23, 12];
numeros.sort(); // --> 12, 23, 3
```

El método sort() puede ordenar valores negativos, cero y positivos en el orden correcto. Cuando compara dos valores, se puede enviar la función *compareFunction(a, b)*, como función de comparación y luego se ordenan los valores de acuerdo al resultado devuelto.

1. Si el resultado es negativo, a se ordena antes que b.
2. Si el resultado es positivo, b se ordena antes de a.
3. Si el resultado es 0, nada cambia.

Por ende si queremos ordenar los números en orden ascendente, esta vez necesitamos restar el primero parámetro (a) del segundo (b):

```
var numbers = [4, 2, 5, 1, 3];
numbers.sort(function(a, b) {
  return a - b;
}); // --> 1, 2, 3, 4, 5
```

Programación Web

La programación de los sitios web es una de las disciplinas dentro del mundo de Internet que más se ha desarrollado y no deja de sorprender con las posibilidades que abre y genera, ya que no sólo consigue satisfacer necesidades que se generan, sino que sin la generación de necesidades ofrecen servicios a los usuarios que éstos no habían imaginado.

En principio, el gran desarrollo de Internet se fundamentó en la posibilidad de enlazar a través de hipervínculos a diferentes páginas web lo que generó la enorme interconexión que es hoy Internet. La creatividad humana no tiene límites y lejos de contentarse con el desarrollo del lenguaje HTML, enriqueciéndose en su sintaxis, aparecieron otros lenguajes, que a su vez desataron una reacción en cadena con respecto a las operaciones que se podían lograr en un sitio web.

De esta manera nació JavaScript, que a su vez incentivó a la creación de otros lenguajes que apuntaron al mismo objetivo como PHP[7], y así constantemente se están buscando mejoras y nuevas alternativas, todas que apuntan a la interacción del usuario con el sitio[8].

[7] https://www.php.net/
[8] https://www.aniel.es/desarrollo-web/programacion-web/

Document Object Model (DOM)

"El Modelo de Objetos de Documento (DOM) del W3C es una plataforma e interfaz de lenguaje neutro que permite a los programas y scripts acceder y actualizar dinámicamente el contenido, la estructura y el estilo de un documento" [9]

Con HTML DOM, JavaScript puede acceder y cambiar todos los elementos de un documento HTML. (ver figura 1.1)

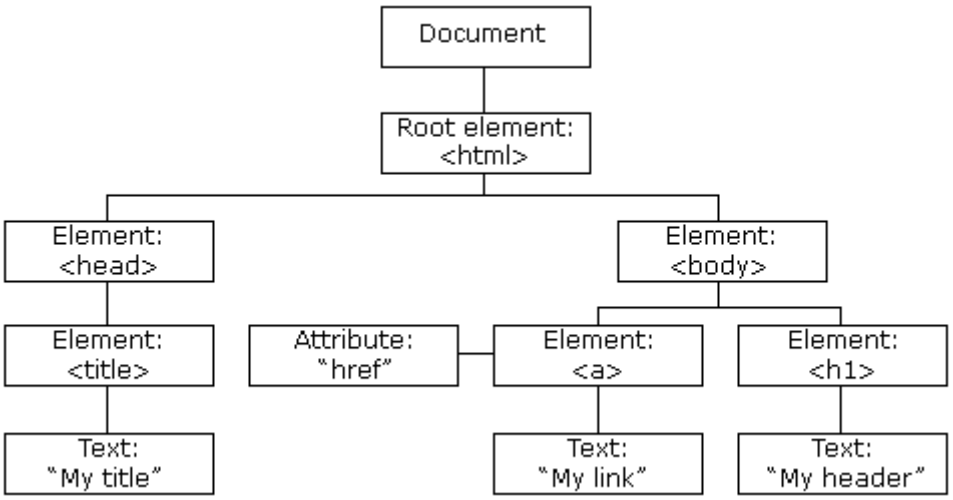

Figura 1.1 DOM

Tipos de nodos:

- **Document**, nodo raíz del que derivan todos los demás nodos del árbol.
- **Element**, representa cada una de las etiquetas HTML. Se trata del único nodo que puede contener atributos y el único del que pueden derivar otros nodos.
- **Attr**, se define un nodo de este tipo para representar cada uno de los atributos de las etiquetas HTML, es decir, uno por cada par atributo=valor.
- **Text**, nodo que contiene el texto encerrado por una etiqueta HTML.

Nota: **Document** representa la página web, por ende, para acceder a cualquier elemento de una web, se debe primero acceder a **document**.

[9] https://www.w3schools.com/js/js_htmldom.asp

Métodos

Los métodos son acciones que se pueden realizar a los elementos HTML mediante DOM.

Los más comunes son los utilizados para encontrar elementos:

- Encontrar elementos HTML por ID (getElementById)
- Encontrar elementos HTML por nombre de etiqueta (getElementsByTagName)
- Encontrar elementos HTML por nombre de clase (getElementsByClassName)
- Encontrar elementos HTML mediante selectores CSS (querySelectorAll)[10]

Este ejemplo encuentra el elemento con id="intro":

```
const element = document.getElementById("intro");
```

Este ejemplo encuentra todos los <p>elementos:

```
const element = document.getElementsByTagName("p");
```

Este ejemplo devuelve una lista de todos los elementos con class="intro".

```
const x = document.getElementsByClassName("intro");
```

Este ejemplo devuelve una lista de todos los elementos <p> con class="intro".

```
const x = document.querySelectorAll("p.intro");
```

Elementos

Los elementos del DOM pueden ser creados, modificados o eliminados. A continuación veremos las principales acciones que se pueden realizar.

Cambiar elementos HTML:

Propiedad	Descripción
element.innerHTML = new html content	Cambia el contenido de un elemento HTML
element.attribute = new value	Cambia el valor del atributo de un elemento HTML
element.style.property = new style	Cambia el valor del atributo de un elemento HTML

[10] https://www.w3schools.com/js/js_htmldom_elements.asp

Método	Descripción
element.setAttribute(attribute, value)	Cambia el estilo de un elemento HTML.

Agregar y eliminar elementos:

Método	Descripción
document.createElement(element)	Crea un elemento HTML
document.removeChild(element)	Elimina un elemento HTML
document.appendChild(element)	Agrega un elemento HTML
document.replaceChild(new, old)	Reemplaza un elemento HTML

Ejemplo:

```
function addElement () {
  // obtener el elemento div con id = "div_example"
  const existDiv = document.getElementById("div_example");
  // crear un nuevo elemento div
  const newDiv = document.createElement("div");
  // agregar el nuevo elemento div existente
  existDiv.appendChild(newDiv);
}
```

Eventos

Los eventos hacen posible que el usuario interactúe con el programa ya que cada elemento HTML tiene una lista de eventos que se le puede asignar. El mismo evento puede ser asignado a varios elementos HTML permitiendo que JavaScript reaccione a eventos.

Ejemplos de eventos HTML:

- Cuando un usuario hace clic con el mouse (onclick)

- Cuando se ha cargado una página web (onload)
- Cuando se ha cargado una imagen (onload)
- Cuando el mouse se mueve sobre un elemento (onmouseover)
- Cuando se cambia un campo de entrada (input)
- Cuando se envía un formulario HTML (onsubmit)
- Cuando un usuario pulsa una tecla (onkeypress)[11]

En este ejemplo, el contenido del <h1> elemento cambia cuando un usuario hace clic en él:

```
<!DOCTYPE html>
<html>
<body>
    <h1 onclick="this.innerHTML = 'Ooops!'">Click este texto</h1>
</body>
</html>
```

[11] https://www.w3schools.com/js/js_htmldom_events.asp

En este ejemplo, se llama a una función desde el controlador de eventos:

```
<!DOCTYPE html>
<html>
<body>
    <h1 onclick="changeText(this)">Click on this text!</h1>
<script type="text/javascript">
    function changeText(id) {
    id.innerHTML = "Ooops!";
    }
</script>
</body>
</html>
```

En este ejemplo, se muestra un cuadro de alerta cuando la página ha terminado de cargarse:

```
<!DOCTYPE html>
<html>
<body onload="mymessage()">
<script type="text/javascript">
    function mymessage() {
            alert("Este mensaje se muestra cuando se ejecuta el evento
onload");
    }
</script>
</body>
</html>
```

Escuchadores

El DOM define el método **addEventListener** (según siglas en inglés) que nos permite precisamente indicar al agente de usuario que permanezca atento a la interacción de un usuario sobre un elemento en concreto, sin necesidad de tocar una sola línea de nuestro código.

La sintaxis de addEventListener es muy sencilla[12]:

```
elemento_que_se_escucha.addEventListener('evento',función_a_lanzar,bool
eano);
```

elemento_que_se_escucha: es cualquier elemento presente en un documento.
evento: es el suceso ocurrido sobre el elemento, con o sin interacción del usuario.
función_a_lanzar: es cualquier función definida que queramos que se ejecute cuando ocurra el evento.
booleano: es un valor que define el orden del flujo de eventos.

Ejemplo: un formulario que se validase antes de ser enviado al servidor

HTML:

```
<button type="submit" id="enviar">Enviar formulario</button>
```

JavaScript:

```
document.getElementById('enviar').addEventListener('click',validar,fals
e);
```

Ejemplo: al dar click sobre un div se debe lanzar una alerta.

HTML:

```
<div id="zona-interactiva">
    <p>Elemento interactivo</p>
</div>
```

JavaScript:

```
window.onload=function(){document.getElementById('zona-interactiva').ad
dEventListener('click',alerta,false);
```

[12] http://www.codexexempla.org/curso/curso_4_3_e.php

```
}
```

Para el caso que sea necesario eliminar un controlador de eventos adjuntado con el método addEventListener el DOM define el método **removeEventListener.**

Ejemplo: al dar click sobre el botón “Habilita Escuchador”/”Inhabilita Escuchador” el div puede o no lanzar una alerta.

HTML:

```
<div id="zona-interactiva">
    <p>Elemento interactivo</p>
</div>
<div>
    <button id="btn_habilita">Habilita Escuchador</p>
    <button id="btn_inhabilita">Inhabilita Escuchador</p>
</div>
```

JavaScript:

```
function alerta() {
    alert('zona-interactiva');
}

window.onload=function(){

document.getElementById('btn_habilita').addEventListener('click',functi
on (){

document.getElementById('zona-interactiva').addEventListener('click',al
erta,false);
    });


document.getElementById('btn_inhabilita').addEventListener('click',func
tion (){

document.getElementById('zona-interactiva').removeEventListener('click'
,alerta,false);
    });
}
```

Referencias

1. https://tc39.es/ecma262/
2. https://www.w3schools.com/js/
3. https://www.w3schools.com/js/js_htmldom.asp
4. https://developer.mozilla.org/en-US/docs/Web/JavaScript
5. https://crockford.com/javascript/javascript.html

6. https://entrenamiento-python-basico.readthedocs.io/es/latest/leccion5/programacion_estructurada.html
7. https://www.freecodecamp.org/espanol/news/cuatro-maneras-diferentes-de-buscar