Introducción a JavaScript

Según [Mozilla (MDN 2020)](https://developer.mozilla.org/es/docs/Web/JavaScript), JavaScript, o simplemente JS, es un lenguaje de [scripting](https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/que-son-los-lenguajes-de-scripting/) [multiplataforma](https://www.lawebdelprogramador.com/diccionario/Multiplataforma/) y orientado a objetos. Es un lenguaje pequeño y liviano. Dentro de un [ambiente de host](https://es.wikipedia.org/wiki/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, diseñado 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](https://lenguajesdeprogramacion.net/diccionario/que-es-un-lenguaje-interpretado/), por lo que no es necesario [compilar](https://developer.mozilla.org/es/docs/Glossary/Compile) los programas para ejecutarlos.  

Objetivos

Introduciremos 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.

Un poco de historia

¿Por qué se llama Javascript ?, según [Crockford](https://en.wikipedia.org/wiki/Douglas_Crockford" \t "_blank) (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.  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](https://elpais.com/tecnologia/2020-08-28/el-lenguaje-de-programacion-que-cambio-la-web-y-tu-vida-cumple-25-anos.html" \t "_blank) en 1995 para la empresa (y navegador popular en su época) Netscape. "*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](https://www.ecma-international.org/" \t "_blank) (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](https://developer.mozilla.org/es/docs/Glossary/ECMAScript" \t "_blank) para que todos los navegadores puedan realizar un intérprete que acepte la misma sintaxis de Javascript.

## Sintaxis de JavaScript

# **Sintáxis**

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 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 a 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 hacerlo.

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 (palabras clave) que no podrán utilizarse por los programadores para usarlas como identificadores. Algunos ejemplo de ellas son: await, break, case, catch, class, const, continue, debugger, default, delete, do, else, enum, export, extensions, false, finally, for, function, if, import, in, instanceof, new, null, return, super, switch, this, throw, true, try, typeof, var, void, while, with, yield, etc.

## Mi primer código en Javascript

## Hola mundo - JavaScript

Tomemos un ejemplo de muestra para imprimir "Hola mundo". Agregamos un comentario HTML opcional que rodea nuestro código JavaScript. Esto es para guardar nuestro código de un navegador que no es compatible con JavaScript. El comentario termina con "// ->". Aquí "//" significa un comentario en JavaScript, así que lo agregamos para evitar que un navegador lea el final del comentario HTML como un fragmento de código JavaScript.  A continuación, llamamos a una función  **document.write**  que escribe una cadena en nuestro documento HTML.

## Tipos de datos en JavaScript

## El último estándar ECMAScript define nueve tipos:

* Seis tipos de datos primitivos, controlados por el operador typeof
  1. **Indefinido**: tipo de instancia === "indefinido"
  2. **Booleano**: tipo de instancia === "booleano"
  3. **Número**: tipo de instancia === "número"
  4. **Cadena**: tipo de instancia === "cadena"
  5. **BigInt**: tipo de instancia === "bigint"
  6. **Símbolo**: tipo de instancia === "símbolo"

* 7. **Nulo:**tipo de instancia === "objeto". Tipo primitivo especial que tiene un uso adicional para su valor: si el objeto no se hereda, se muestra null.
* 8. **Objeto:**tipo de instancia === "objeto". 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 nuevo

* 9. **Función:** una estructura sin datos, aunque también respondió 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.

## Variables y Ámbitos en JavaScript

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:

**1. 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 utilizándola 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.

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

**3. 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 guion 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:

**1. 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.

**2. 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.

**3. 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 en JavaScript

Los operadores en los lenguajes de programación son símbolos que indican cómo se deben manipular los operandos. 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

|  |  |  |  |
| --- | --- | --- | --- |
| Operador | | Descripción | Ejemplo |
| + | Suma | Suma dos números | let ​​suma = 5 + 7; // = 12 |
| - | Resta | Resta dos números | let ​​resta = 7-2; // = 5 |
| \* | Multiplicación | Multiplica dos números | let ​​mul = 5 \* 7; // = 35 |
| / | División | Dividir 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; i--; // = 1 |

## Operadores lógicos

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

**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 verdadera, 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";**

## Funciones y estructuras en JavaScript

## 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(expression) {**

**case x:**

**// code block**

**break;**

**case y:**

**// code block**

**break;**

**default:**

**// code block**

**}**

Primero busca una cláusula case con una etiqueta que coincide 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 predeterminada y transfiere el control a esa cláusula ejecutando las sentencias asociadas. Si no se encuentra la cláusula predeterminada, 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.

La declaración de una función consiste en:

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 que ha de 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.

# **Estructuras repetitivas**

Las estructuras repetitivas o cíclicas nos ejecuta 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 la que 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:

**Estructuras repetitivas en JavaScript:**

* **FOR**
* **WHILE**
* **DO - WHILE**

## Sentencia for

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

**for ([expresion-inicial]; [condicion]; [expresion-final]) {**

**// código ha ser ejecutado;**

**}**

Cuando un bucle for se ejecuta, ocurre lo siguiente: la [expresión-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 [condición]. Si el valor de condición es true, se ejecuta la sentencia del bucle. Si el valor de condición es falso, el bucle para, finaliza. Si la expresión condición es omitida, la condición es asumida como verdadera. Se ejecuta la expresión [expresión-final], si hay una, y el control vuelve a evaluar la expresión [condición].

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; i <10; i ++) {**

**console.log ("Número:" + i);**

**}**

## Sentencia while (mientras)

Una sentencia while ejecuta sus sentencias mientras la condición sea 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 como falso, se detiene la ejecución y el control pasa a la sentencia siguiente al while.

## Sentencia do-while (hacer - mientras)

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 hacer mientras (do while) se ejecutará al menos una vez. Ejemplo:

**let resultado = '';**

**let ​​i = 0;**

**do {**

**i = i + 1;**

**resultado = resultado + i;**

**} while (i <5);**

**console.log (resultado);**

**// resultado esperado: "12345"**

## Sentencia continue (continuar)

La sentencia continue puede usar 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 (romper)

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 en JavaScript

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:

1. Obtener todos los elementos del arreglo que cumplen una condición específica. (**Filter**)
2. Obtener al menos uno de los elementos del arreglo que cumple dicha condición. (**Find**)
3. Obtener si un valor específico es parte del arreglo (**Includes**)
4. 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](https://developer.mozilla.org/es/docs/Glossary/Callback_function" \t "_blank) 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 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 XNo. En el mejor de los casos, el primer elemento del arreglo es el elemento que estamos buscando, por eso para ese caso ese elemento 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:

const equipos = ['Real Madrid', 'Manchester Utd', 'Bayern Munich', 'Juventus'];

equipos.sort();

// ['Bayern Munich', 'Juventus', 'Manchester Utd', 'Real Madrid']

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) {

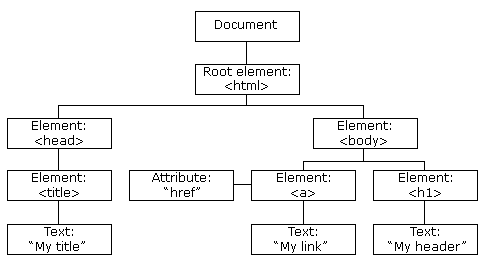
  return a - b;

}); // --> 1, 2, 3, 4, 5

¿DOM que es?

"*El Modelo de Objetos de Documento (*[*DOM*](https://www.w3.org/2005/03/DOM3Core-es/introduccion.html)*) 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*"

Con HTML DOM, JavaScript puede acceder y cambiar todos los elementos de un documento HTML. 

****

**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.

**Métodos**

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

Los más comunes son los  utilizados para encontrar elementos:

1. Encontrar elementos HTML por ID (**getElementById**)
2. Encontrar elementos HTML por nombre de etiqueta (**getElementsByTagName**)
3. Encontrar elementos HTML por nombre de clase (**getElementsByClassName**)
4. Encontrar elementos HTML mediante selectores CSS (**querySelectorAll**)

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 |
| 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);

}

## Archivos JSON

**JSON (notación de objetos javascript)** es un formato de intercambio de datos. Es muy parecido a un subconjunto de sintaxis JavaScript, aunque no es un subconjunto en sentido estricto. Aunque muchos lenguajes de programación lo soportan, JSON es especialmente útil al escribir cualquier tipo de aplicación basada en JavaScript, incluyendo sitios web y extensiones del navegador. Por ejemplo, es posible almacenar la información del usuario en formato JSON en una cookie o almacenar las preferencias de extensión en JSON en una cadena de valores de preferencias del navegador.  
  
JSON es capaz de representar números, valores lógicos, cadenas, valores nulos, arreglos y matrices (secuencias ordenadas de valores) y objetos (mapas de cadena de valores) compuestos de estos valores (o de otras matrices y objetos). JSON no representa de manera nativa tipos de datos más complejos como funciones, expresiones regulares, fechas, y así sucesivamente (en objetos de fecha serializados por defecto como una cadena que contiene la fecha en formato ISO, al no hacerlo de ida y vuelta, la información no se pierde por completo). Si se necesita que JSON represente tipos de datos adicionales, se puede transformar los valores, ya que son serializados, o antes de su deserialización.

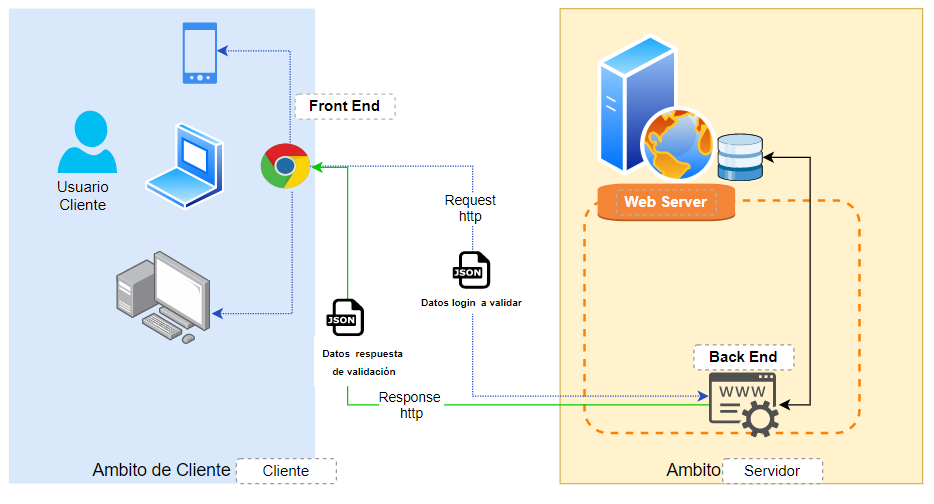
Los programadores conocemos estas convenciones utilizadas por JSON:

* - JSON son las siglas de JavaScript Object Notation.
* - El formato fue especificado por Douglas Crockford.
* - Fue diseñado para el intercambio de datos legibles por humanos.
* - Se ha ampliado desde el lenguaje de secuencias de comandos JavaScript.
* - La extensión del nombre de archivo es .json.
* - El tipo de Internet Media type es application / json.

##### **¿Para qué se usa?**

* - El formato JSON se utiliza para serializar y transmitir datos estructurados a través de una conexión de red.
* - Se utiliza principalmente para transmitir datos entre un servidor y aplicaciones web.
* - Los servicios web y las API utilizan el formato JSON para proporcionar datos vía internet.
* - Se puede utilizar con los lenguajes de programación y frameworks más actuales como  Typescript, Java, Angular, Spring boot, entre muchos otros.

**Nota:** La mayoría de las bases de datos pueden almacenar los archivos JSON, aunque almacenar un JSON completo rompe la normalización de la base de datos.  
  
En la siguiente imagen veremos un esquema que ilustra cómo viaja el archivo JSON:



Analizando la secuencia del esquema anterior, podemos observar la siguiente:

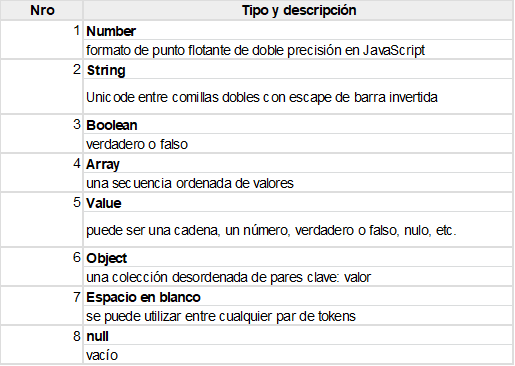
1. Un usuario ingresa sus datos de login en la página (Front End).
2. El código del Fornt End toma los datos y los serializa (convertir los datos/objeto del código a texto) en formato JSON y los envía al backend.
3. El Back End recibe el archivo JSON y lo deserializa (convertir el texto a objeto/código) para ser procesados, consultar la base de datos, etc.
4. Luego del procesamiento el Back End  la respuesta es serializada (convertir los datos/objeto del código a texto) para enviársela  al Front End notificando el resultado.
5. El código de la página Front End recibe el archivo JSON lo deserializa (convertir el texto a objeto/código) y muestra el resultado al usuario en la pantalla del navegador.

En la siguiente imagen te dejamos un ejemplo de cómo se ve un formato JSON en el código o en grilla,



##### **Tipos de datos**

El formato JSON admite los siguientes tipos de datos:



Reglas de sintaxis JSON:

* - Los datos están en pares de nombre / valor
* - Los datos están separados por comas
* - Las llaves sostienen objetos
* - Los corchetes sostienen matrices

**Los datos en JSON: un nombre y un valor**

Los datos JSON se escriben como pares de nombre / valor, al igual que las propiedades de los objetos de JavaScript. Un par de nombre / valor consta de un nombre de campo (entre comillas dobles), seguido de dos puntos, seguido de un valor:

"firstName":"Juana"

**Los Objetos en JSON:**

Los objetos JSON se escriben entre llaves. Al igual que en JavaScript, los objetos pueden contener varios pares de nombre / valor:

{"firstName":"Juana", "lastName":"Fernandez"}

**Las Matrices en JSON**

Las matrices JSON se escriben entre corchetes. Al igual que en JavaScript, una matriz puede contener objetos, te dejamos un ejemplo de una matriz Empleado con 3 Objetos:

"employees":[  
  {"firstName":"Juana", "lastName":"Fernandez"},  
  {"firstName":"Ana", "lastName":"Rntani"},  
  {"firstName":"Francisco", "lastName":"Petrol"}  
]

**Validar formato JSON:**

Existen muchas páginas para validar formato, te dejamos una para empezar, cada vez que trabajes con un formato JSON siempre debes estar seguro si el formato esta correcto y es válido, para ir a la página has clic [aquí](https://jsongrid.com/json-grid" \t "_blank)

**Crear objetos simples de pruebas:**

Los objetos JSON se pueden crear con JavaScript. Veamos las diversas formas de crear objetos JSON usando JavaScript:

* - Creación de un objeto vacío

var JSONObj = {};

* - Creación de un nuevo objeto -

var JSONObj = new Object();

* -  Creación de un objeto con atributo **nombrelibro**con valor en cadena, atributo **precio** con valor numérico. Se accede al atributo usando '.' Operador -

var JSONObj = { "nombrelibro":"El hacedor", "precio":500 };