JAVASCRIPT

TEMARIO

- Historia de JavaScript
- Cómo y donde escribir JavaScript
- Sintaxis de JavaScript
- Variables, constantes y sus ámbitos
- Tipos de datos en JavaScript
- Trabajando con operadores
- Nuevos métodos en strings
- Objeto Math
- Condicionales
- Bucles
- Arrays e Iteradores
- Spread Operator
- Programación Orientada a Objetos (POO)
- Objetos
- Propiedades y métodos
- Tipos de funciones en JavaScript
- Funciones
- Clases y constructores en JavaScript
- Prototipos
- Symbols
- Maps
- Sets
- Integrando JavaScript con HTML
- Introducción a Eventos
- DOM (Document Object Model)
- Trabajando con eventos
- Formularios y Expresiones regulares
- Objeto Date
- Temporizadores
- APIs HTML5
- Prácticas Finales

¿QUÉ ES JAVASCRIPT?

- JavaScript es un lenguaje de programación.
- Lo creó Brendan Eich en una semana.
- Es el único lenguaje de programación que puede interpretar el navegador
- El nombre se le puso porque en ese momento estaba de moda Java.
- JavaScript ha sido mal visto desde su inicio hasta el 2009 aproximadamente.
- Hasta 2009 se podía utilizar sólo del lado del cliente. Actualmente con Node.js podemos ejecutarlo en un servidor.

HISTORIA DE JAVASCRIPT

- 1995 – Netscape crea JavaScript
- 1997 – Netscape se lleva JavaScript a ECMA (European Computer Manufacturers Association)
- 1997 – Se lanza el estándar ECMA-262 y se crea ECMAScript 1.0
- 1998 – ECMAScript 2 – Ajuste con el estándar internacional
- 1999 – ECMAScript 3
- 2009 – ECMAScript 5
- 2011 – ECMAScript 5.1
- 2015 – ECMAScript 6

TECNOLOGÍAS DESCENDIENTES DE JAVASCRIPT

Tecnologías Independientes

- AJAX
- jQuery
- nodeJS
- JSON

Frameworks

- Angular
- React
- VueJs

DÓNDE ESCRIBIR CÓDIGO

- Se necesita un editor de texto, no un procesador de texto, por lo que Word no sirve
- Existen varias alternativas:
  - La consola del navegador directamente
  - Instalar Node.js
  - Sublime Text
  - Atom
  - Visual Studio Code
  - Brackets
  - Notepad++
  - Bloc de notas

SINTAXIS DE JAVASCRIPT

- Es case sensitive.
  - Numero no es igual que numero
- Es de tipado débil o dinámico
  - Las variables son del tipo del dato que almacenen;
- Las sentencias finalizan con ;
  - No es obligatorio pero es muy recomendable
- Los bloques finalizan con }
  - De forma opcional se puede añadir un ; después de }

VARIABLES Y CONSTANTES
SCOPE O ÁMBITO

- Una variable es un espacio que reservamos en memoria para almacenar un dato que podrá cambiar durante la ejecución de nuestro programa.
  - La palabra reservada para declarar variables es "let" no es recomendable usar "var"

- Las variables se pueden: declarar, inicializar y modificar.

- Una constante es un espacio que reservamos en memoria para almacenar un dato que no cambiará durante la ejecución de nuestro programa.
  - La palabra reservada para declarar constantes es "const"

- El scope o ámbito es la zona donde existe nuestra variable o constante.

DECLARACIÓN, INICIALIZACIÓN Y MODIFICACIÓN

- Una variable se declara con la siguiente estructura:
  - let numero;
- Una variable se inicializa con la siguiente estructura:
  - numero = 5;
- Se puede declarar e iniciar en la misma sentencia:
  - let numero = 5;
- Para modificar el valor de una variable existente:
  - numero = 3;
- Las constantes solo admiten la declaración e inicialización en la misma sentencia.
  - const PI = 3.14;

TIPOS DE DATOS EN JAVASCRIPT

- Primitivos
  - Numeros -> let numero = 5;
  - Strings (cadenas) -> let palabra = 'hola'; let palabra = "hola";
  - Boolean -> let respuesta = true; let respuesta = false;
  - Undefined
  - Null
  - Symbol

EJERCICIO BÁSICO

- Crea una página **index.html** que llame a un archivo javascript cuyo nombre sea **scripts.js**
- En el archivo .js declara:
  - Una variable **numero** cuyo valor sea -4.
  - Una variable **palabra** cuyo valor sea ”Hola Mundo”
  - Una variable booleana cuyo nombre sea **respuesta** e inicialízala con valor **true**
  - Crea una constante con nombre **pi** cuyo valor sea 3.14
  - Modifica el valor de la variable numero para que en lugar de -4 sea 54 pero sin tocar la parte de la declaración de la variable.
  - ¿Qué pasa si modificas el valor de la variable antes de que esta sea declarada?
  - Muestra por consola los valores de todas las variables creadas. Para pintar por consola se utiliza la instrucción **console.log(XXXXXXXXX);** Donde XXXXX será el nombre de la variable a pintar.
  - Modifica el valor de **pi** por el valor 3.1415. ¿Qué ocurre?.

OPERADORES EN JAVASCRIPT

Matemáticos	
Suma / Concatenación	+
Resta	-
Multiplicación	*
División	/
Módulo	%

Asignación	
Asignación	=
Suma y asignación	a += 3 (a = a + 3)
Resta y asignación	a -= 3 (a = a - 3)
Multiplicación y asignación	a *= 3 (a = a * 3)
División y asignación	a /= 2 (a = a / 2)
Módulo y asignación	a %= 2 (a = a % 2)

Incremento/Decremento	
Post-incremento en 1	a++
Pre-incremento en 1	++a
Post-decremento en 1	a--
Pre-decremento en 1	--a

EJERCICIO BÁSICO – Operadores aritméticos

- Crea una página **index.html** que llame a un archivo javascript cuyo nombre sea **scripts.js**
- En el archivo .js declara:
  - Declara 2 variables y asígnales los valores 5 y 2 respectivamente
  - Muestra por consola la **suma** de ambas variables.
  - Muestra por consola la **resta** de ambas variables.
  - Muestra por consola la **multiplicación** de ambas variables.
  - Muestra por consola la **división** de ambas variables.
  - Muestra por consola la **módulo** de ambas variables.
  - Declara una variable cuyo valor sea tu nombre.
  - Muestra por consola el texto **Hola** concatenado con el valor de la variable anterior.

EJERCICIO BÁSICO - Incremento/Decremento

- Si declaramos una variable a=5 y utilizamos las siguientes instrucciones de incremento:
  - console.log(a);
  - console.log(a++);
  - console.log(a);
  - ¿qué se ha mostrado por consola?
  - Si añadimos console.log(++a) ¿qué se mostraría en pantalla)
- Prueba las mismas instrucciones pero con el decremento
- Utiliza los operadores de asignación para sumar 3 a la variable **a** y muéstrala por consola. Al resultado obtenido réstale 3 y muéstralo por pantalla.

MÉTODOS Y PROPIEDADES DE LOS STRING

- Método: Es todo aquello que la cadena puede hacer. Ej: convertirse en mayúsculas
- Propiedad: Son las características que la cadena tiene por ser una cadena. Ej: Longitud.

Se emplean usando la nomenclatura del punto:

cadena.método

cadena.propiedad.

Listado de todas las propiedades y métodos de String: https://www.w3schools.com/jsref/jsref_obj_string.asp

PRINCIPALES MÉTODOS DE STRING

Todos estos métodos devuelven una cadena nueva, la cadena original no será modificada.

- **toUpperCase():** Devuelve la cadena a mayúsculas.
- **toLowerCase():** Devuelve la cadena a minúsculas.
- **indexOf():** Devuelve la posición en la que se encuentra el string, si no lo encuentra devuelve -1
- **replace(valor a buscar, valor nuevo):** Remplaza el fragmento de la cadena que le digamos y pone el valor nuevo.
- **substring(inicio [,fin]):** Extrae los caracteres desde inicio hasta fin (el final no se incluye). Si no se incluye el fin extrae hasta el final.
- **slice(inicio [,fin]):** Igual que substring pero admite valores negativos, si ponemos valores negativos empezará desde atrás. Si no se incluye el final extrae hasta el final.
  Por ejemplo:(2,-4) -> Empieza a contar en el tercer carácter y los 4 últimos no los considera
- **trim(): E**limina los espacios al inicio y al final de la cadena

NUEVOS DE EMACSCRIPT6:

- **startsWith(valor [,inicio]):** Sirve para saber si la cadena empieza con ese valor. Devuelve true o false
- **endsWith(valor [,longitud]):** Sirve para saber si la cadena termina con ese valor. Devuelve true o false
- **includes(valor [,inicio]):** Igual que indexOf pero devuelve true o false
- **repeat(valor):** Repite el string el número de veces que le indiquemos.

TEMPLATE STRINGS

EJEMPLO

```
let nombre=“Pepito”;
let apellido=“Grillo”;
let edad = 20;

console.log("Hola " + nombre + " " + apellido + ". Tienes " + (edad+1) + " años.");
```

¿Cómo sería utilizando TEMPLATE STRINGS?

Se añaden dos acentos graves (la tecla al lado de la letra p). Se utilizan para incluir dentro de un string cualquier código válido javascript.

```
console.log(`Hola ${nombre} ${apellido}. Tienes ${edad} años.`);

console.log(`Hola ${nombre} ${apellido}. El año que viene tendrás ${edad+1} años.`);
```

PRINCIPALES PROPIEDADES DE STRING

- **length**: devuelve la longitud de la cadena

EJERCICIO BÁSICO - STRINGS

- Declara una variable cuyo nombre sea **cadena** y tenga el valor “Hola Mundo”.
- Muestra por consola la longitud de la cadena.
- Muestra por consola la cadena con todos sus caracteres en mayúsculas.
- Muestra por consola la cadena con todos sus caracteres en minúsculas.
- Muestra por consola la posición de la cadena en la que se encuentra la letra **o.**
- Muestra por consola la posición de la cadena en la que se encuentra la cadena “Hola”.
- Reemplaza la cadena “Mundo” por la cadena “Youtube” y muestra el resultado.
- Extrae la segunda parte de la cadena y muéstrala por consola.
- Comprueba si la cadena empieza por “h”.
- Comprueba si la cadena empieza por “H”.
- Muestra por consola la letra “r” 10 veces.
- Utilizando TEMPLATE STRINGS declara 3 variables con tu nombre, apellidos y edad y muéstralas por consola introduciéndolas dentro de una cadena.

OBJETO MATH

Es un objeto que se utiliza para operaciones matemáticas más específicas.

Al ser un objeto también utiliza la nomenclatura del punto

Propiedades:

- Math.E - Math.Pi

Métodos:

- Math.abs(x) Devuelve el valor absoluto de x
- Math.ceil(x) Devuelve el entero más grande mayor o igual que un número.
- Math.floor(x) Devuelve el entero más pequeño menor o igual que un número.
- Math.pow(x, y) Devuelve la potencia de x elevado a y
- Math.random() Genera un número pseudoaleatorio entre 0 y 1
- Math.round(x) Devuelve el valor de un número redondeado al entero más cercano.
- Math.sign(x) Devuelve el signo de la x, que indica si x es positivo, negativo o cero.

EJERCICIO BÁSICO - MATH

- Genera un número aleatorio entre 0 y 100 redondeando y muéstralo por consola
- Muestra por consola el valor de PI.
- Genera un número aleatorio entre 5 y 10 redondeando y muéstralo por consola. *Para este caso lo recomendado es utilizar la fórmula (Math.random()* (max-min)+min).*
- Comprueba como funciona el método sign() para los siguientes valores: -5, 0, 5 .

FUNCIONES GLOBALES JAVASCRIPT

- **parseInt(valor):** convierte una cadena a entero. Devuelve un entero.
- **parseFloat(valor):** convierte una cadena a punto flotante. Devuelve un decimal.
- **String(valor):** convierte el valor de un objeto a cadena.
- **Number(valor):** convierte el valor de un objeto a un número.
- **isNaN():** determina si un valor es un número ilegal.

Ejemplos

```
var x1 = true;
var x2 = false;
var x3 = new Date();
var x4 = "999";
var x5 = "999 888";

var n =
Number(x1) + "<br>" +
Number(x2) + "<br>" +
Number(x3) + "<br>" +
Number(x4) + "<br>" +
Number(x5);
```

```
<!DOCTYPE html>
<html>
<body>

<p>Click the button to parse different strings.</p>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
function myFunction() {
  var a = parseFloat("10") + "<br>";
  var b = parseFloat("10.00") + "<br>";
  var c = parseFloat("10.33") + "<br>";
  var d = parseFloat("34 45 66") + "<br>";
  var e = parseFloat("   60   ") + "<br>";
  var f = parseFloat("40 years") + "<br>";
  var g = parseFloat("He was 40") + "<br>";

  var n = a + b + c + d + e + f + g;
  document.getElementById("demo").innerHTML = n;
}
</script>

</body>
</html>
```

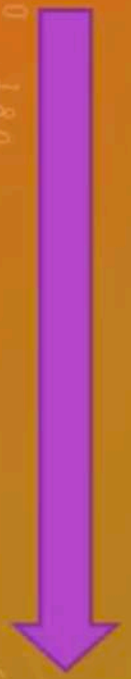
Listado completo

- https://www.w3schools.com/jsref/jsref_obj_global.asp

FLUJO DE UN PROGRAMA

El flujo de un programa siempre será de arriba abajo

```
let num = 2;

console.log(num);

num = 5;

console.log(num)

let word = 'Hola Mundo';

console.log(word);
```



Estructuras de control de flujo:

- Condicionales
  - Simples
  - Compuestos
  - Múltiples
- Bucles
  - Determinados
  - Indeterminados

ESTRUCTURA IF-ELSE

Sintaxis:

Condición simple:

- if(condición) //código a ejecutar

- if(condición){
  //código a ejecutar 1
  //código a ejecutar 2
  ...
  }

Condición compuesta:

- if(condición) //código a ejecutar
- else //código a ejecutar en caso contrario

Para comprobar la igualdad dentro de una condición se usa **=== si queremos que coincidan incluso en tipo.**

EJERCICIOS BÁSICOS - CONDICIONALES

EJERCICIO 1

- Declara una variable que se llame **num** e inicialízala a cero.
- Utilizando condicionales deberás comprobar si el valor de la variable **num** es mayor que cero, menor que cero o igual a cero. En cada uno de los casos deberás mostrar un mensaje utilizando TEMPLATE STRING mostrando el valor de la variable **num** e indicando si es mayor, menor o igual a cero.

EJERCICIO 2

- Declara dos variables una con valor 5 y otra con valor 1.
- Utilizando operadores lógicos && o || utilizada el adecuado para mostrar el mensaje de que ambos valores son mayores que cero.

EJERCICIO 3

- Dada una cadena, se mostrará un mensaje en el caso de que tenga mas de 4 letras, otro mensaje en el caso de que tenga menos de 4 letras y otro mensaje en el caso de tener 4 letras.

EJERCICIOS - CONDICIONALES

EJERCICIO 4

- Ordena 3 números de mayor a menor.
- Los números se introducirán por teclado. Para ello es necesario utilizar ***prompt*** para que pregunte los números.
- Muestra por consola los números ordenados de mayor a menor.

EJERCICIO 5

- Modifica el ejercicio anterior para que muestre los resultados en lugar de por consola los pinte en la página html. Para ello puedes crear en el HTML, un elemento div para pintar los números introducidos y otro elemento div para pintar el resultado ordenado. *Pista para este ejercicio: textContent y getElementById.*

ESTRUCTURA SWITCH

Se utiliza para elegir un camino de varios preestablecidos. Tenemos 2 tipos principales:

Sintaxis simple:

```
switch(evaluación) {
    case n1:
        //código
        break;
    case n2:
        //código
        break;
    default:
        //código
}
```

Sintaxis múltiple:

```
switch(evaluación) {
    case n1:
    case n2:
    case n3:
        //código
        break;
    default:
        //código
}
```

OPERADOR TERNARIO

Se utiliza cuando una condición será true o false, al igual que el if.

Su ejecución puede tener una o varias sentencias, en este caso irán separadas por comas y entre paréntesis.

(condición) ? true : false

(condición) ?

(primera sentencia,

segunda sentencia)

:

(primera sentencia,

segunda sentencia)

EJERCICIO – Operador ternario

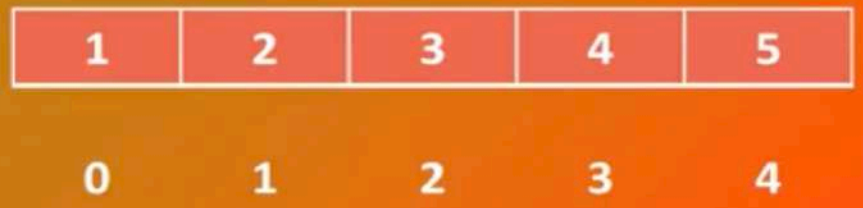
EJERCICIO

- Utilizando el operador ternario comparar si un número es par o impar mostrándolo por consola.

ARRAYS

- Son estructuras que nos permiten almacenar varios datos y agruparlos.
- Se pueden llenar con cualquier tipo de dato válido en JavaScript y deben ir separados por comas
- Se pueden mezclar tipos de datos, pero no es recomendable.
- Se declaran con llaves cuadradas o corchetes []
- Pueden declararse vacíos o con un contenido ya establecido
- Pueden añadirse o eliminarse elementos en el momento que queramos

```
let numero = 5

let array = []

let numeros = [1,2,3,4,5]
```



- Cada uno de los elementos podrá ser identificado por su índice, es decir su posición.
- Los índices empiezan a contar desde 0

EJERCICIO – Arrays

EJERCICIO

- Crea un Array que contenga los valores 1,2,3,4,5.
- Muestra por consola la posición en la que se encuentra el número 4.
- Muestra por consola la suma del segundo y tercer elemento del Array. Deberá ser 5.
- Crea un Array con 4 cadenas y muestra por consola el número de caracteres de la segunda palabra del Array. Utiliza TEMPLATE STRINGS para mostrar el mensaje de “La palabra XXXXXX tiene xx letras.

MÉTODOS BÁSICOS DE ARRAY

Mas adelante veremos métodos más avanzados de Arrays.

- **Array.isArray(variable a evaluar)** - Devuelve true si la variable es un array.
- **.shift()** - Elimina el primer elemento del array y devuelve ese elemento
- **.pop()** - Elimina el último elemento de un array y devuelve ese elemento
- **.push(element1, element2,...)** - Añade uno o más elementos al final del array y devuelve la nueva longitud.
- **.unshift(element1, element2,...)** - Añade uno o más elementos al comienzo del array y devuelve la nueva longitud.
- **.indexOf()** - Devuelve el primer índice del elemento que coincida con el valor especificado, o -1 si ninguno es encontrado.
- **.lastIndexOf()** - Devuelve el último índice del elemento que coincida con el valor especificado, o -1 si ninguno es encontrado.
- **.reverse()** - Invierte el orden de los elementos del array.
- **.join('separador')** - Devuelve un string con el separador que indiquemos, por defecto son comas.
- **.splice(a,b,items)** - Cambia el contenido de un array eliminando elementos existentes y/o agregando nuevos elementos. Donde: **a** - Indice de inicio, **b** - Número de elementos a borrar (opcional) e **items** - Elementos a añadir en el caso de que se añadan. (opcional)
- **.slice(a,b)** - Extrae elementos de un array desde el índice a hasta el índice b. Si no existe b lo hace desde a hasta el final, si no existe ni a ni b hace una copia del original.

EJERCICIO – Arrays y Métodos

EJERCICIO

- Crea un Array con nombre **numbers** que contenga los valores 1,2,3,4,5,6 y muestra por consola el número de elementos que contiene el array.
- Crea una variable con nombre **number** con valor 4. Muestra por consola si la variable **number** es un array o no. Muestra por consola si la variable **numbers** creada en el primer apartado es un array o no.
- Elimina el primero elemento del array y muestra por consola el elemento borrado y el resultado del array.
- Elimina el último elemento del array y muestra por consola el elemento borrado y el resultado del array.
- Añade al principio del array el elemento borrado anteriormente y muéstralo en consola.
- Invierte el orden de los elementos de un array y muéstralo como un String separado por espacios.
- Reemplaza los elementos 3 y 4 del array por 10, 23 y 54. Muéstralo por consola.
- Añade los elementos 12 y 14 delante del 10 utilizando un único método.

BUCLES

- Se usan cuando queremos que un trozo de código se repita.
- Existen bucles determinados e indeterminados.
- Los determinados se usan cuando especificamos el número de veces que se va a repetir.
  - Imprimir números del 1 al 10
- Los indeterminados se usan cuando no sabemos el número de veces que se van a repetir.
  - Repetir mensaje de introducir contraseña
- La estructura de un bucle siempre es la misma
  - Bucle{

    Código a ejecutar

    }

BUCLE WHILE

- Es un bucle indeterminado ya que no sabemos cuantas vueltas dará durante su ejecución

- Su sintaxis se compone de una única parte
  - Condición de salida

```
while(condición){
        Código a ejecutar
}
```

BUCLE DO WHILE

- Es un bucle indeterminado ya que no sabemos cuantas vueltas dará durante su ejecución

- Su sintaxis se compone de dos partes
  - Código a ejecutar
  - Condición de salida

```
do{
    //Código a ejecutar
}while(condicion)
```

BUCLE FOR

- Es un bucle determinado ya que hay que especificar cuantas vueltas dará durante su ejecución

- Su sintaxis se compone de 3 partes
  - Iniciación de variable
  - Número de vueltas
  - Incremento o decremento

- for(let i=0;i<=10;i++){
  Código a ejecutar
  }

BUCLE FOR

- Durante su ejecución la variable i aumentará su valor en cada vuelta

```
for(let i=0;i<=3;i++){
        console.log(i)
}
```

1ª vuelta: i=0 - ¿i<=3? - 0 - i++

2ª vuelta: i=1 - ¿i<=3? - 1 - i++

3ª vuelta: i=2 - ¿i<=3? - 2 - i++

4ª vuelta: i=3 - ¿i<=3? -3 - i++

5ª vuelta: i=4 - ¿i<=3? - fin del bucle

BUCLE FOR OF / FOR IN

Simplifica el bucle for tradicional sin tener que darle un número de vueltas ni realizar un incremento

```
let names = ['Paco', 'José', 'Paula', 'María']

for(let name of names){
    console.log(name)
}
```

1ª vuelta: Paco
1ª vuelta: José
3ª vuelta: Paula
4ª vuelta: María

```
for(let name in names){
    console.log(name)
}
```

1ª vuelta: 0
2ª vuelta: 1
3ª vuelta: 2
4ª vuelta: 3

Devuelve el índice. Suele utilizarse más para recorrer Objetos que lo veremos más adelante.

EJERCICIOS BÁSICO – Bucles

EJERCICIOS

- Solicita por pantalla que se introduzca una contraseña hasta que se introduzca la contraseña correcta. La contraseña puede ser por ejemplo: **hola.**
- Implementa el mismo código utilizando "**do – while**". ¿Qué diferencia hay respecto al while?
- Utilizando el bucle **for** tradicional imprime los números del 1 al 10 de forma descendente.
- Dado el array [0,1,2,3,4,5]. Recorre el array con un bucle **for** tradicional e imprime el contenido con TEMPLATE STRING la siguiente cadena: "i vale ***XX*** y el valor de la posición en el array es ***XX***".
- Utilizando **for in** muestra por consola los nombres del *array[Marta, Inma, Joaquín, Javier].*
- Investiga cómo función dentro de los bucles for las palabras reservadas "*break*" y "*continue*". Realiza un ejemplo con el array de nombres anterior.

REALIZACIÓN DE BOLETÍN BÁSICO DE EJERCICIOS JAVASCRIPT

10 EJERCICIOS EN TOTAL

OBJETOS - INTRODUCCIÓN

Son estructuras de datos que representan propiedades, valores y acciones que puede realizar el objeto

Todos los objetos tienen propiedades o atributos y comportamientos o acciones representados por pares de clave (key) : valor (value)

```
const computer = {
    screenSize:17,
    model:'MacBook Pro'
}
```

```
const table = {
    material:'wood',
    width:160,
    height:110
}
```

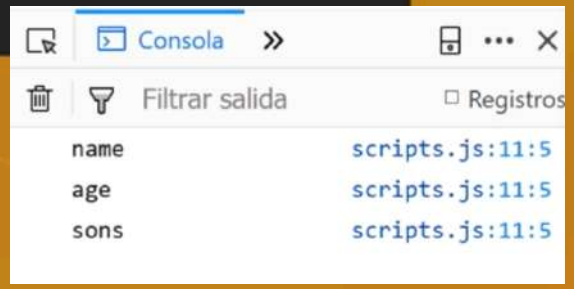
OBJETOS - INTRODUCCIÓN

Para acceder a las propiedades y acciones del objeto se utiliza la nomenclatura del punto

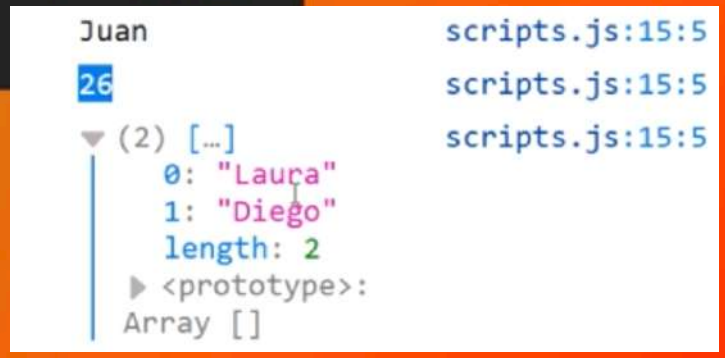
```
const person = {
    name: 'Juan',
    age: 26,
    sons: ['Laura', 'Diego']
}
```

```
console.log(person.name);
console.log(person.age);
console.log(person.sons[0]);
console.log(person.sons[1]);
```

```
for(const key in person){
    console.log(key);
}
```




```
for(const key in person){
    console.log(person[key]);
}
```



EJERCICIOS BÁSICO – Objetos

EJERCICIOS

- Crea un objeto **persona.** Persona está compuesto por un campo **nombre, edad, hijos. Hijos** será un array con los siguientes valores “Batman”, “Ironman”, “Yoda”, “Hulk”.
- Muestra por consola todos los nombres de los hijos.
- Usando Template String muestra por consola el siguiente mensaje: “Hola ***nombre***. Tienes **XX** años y tus hijos se llaman XXXX,XXXXX,XXXXX,XXXXXX

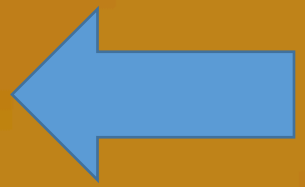
FUNCIONES- INTRODUCCIÓN

Son fragmentos de código que escribimos para ejecutar una tarea y no volver a escribir el mismo código más de una vez.

Nos ayudan a modularizar el código

Las funciones deben realizar una sola tarea

```
function nombreFuncion(){
    //Código a ejecutar
}

const nombreFuncion = () =>{
    //Código a ejecutar
}
```



Anteriormente. Todavía podéis encontrarlo así.

Con EMACSCRIPT 6

FUNCIONES- INTRODUCCIÓN

Pueden recibir parámetros

```
function nombreFuncion(parametro1, parametro2){
    //Código a ejecutar
}

const nombreFuncion = (parametro1, parametro2) =>{
    //Código a ejecutar
}
```

Pueden devolver valores

```
function nombreFuncion(parametro1, parametro2){
    return parametro1+parametro2
}

const nombreFuncion = (parametro1, parametro2) => parametro1+parametro2
```

Si es sólo una línea, con esta opción no hace falta return ni llaves

EJERCICIOS BÁSICO – Funciones

EJERCICIOS

- Crea una función **saludar** que imprima por pantalla el texto “Hola desde una función”. Crea un bucle que llame 5 veces a la función saludar.
- Crea una función **saludarUsuario** que reciba como parámetro el nombre de un usuario e imprima por pantalla el texto “Hola XXXXXXX desde una función”.
- Crea una función **suma** que reciba 2 parámetros (num1,num2). Si num1 es igual a 3, realizara la suma de num1 y num2 y devolverá el resultado. En caso contrario devolverá el valor de num1.

PROGRAMACIÓN ORIENTADA A OBJETOS

Es un paradigma de la programación que actualiza la forma de programar anterior.

Algunos de los conceptos fundamentales son:

- Clase
- Herencia
- Objeto
- Método
- Evento
- Etc

```
const persona = {
    nombre:'Juan',
    apellido: 'García',
    edad:27
}
```

```
const persona2 = {
    nombre:'Marta',
    apellido: 'Pérez',
    edad:35
}
```

CLASES - PROPIEDADES

Son plantillas que se utilizan para crear objetos iguales

Cuando creamos un objeto, a esa acción se le denomina INSTANCIAR un objeto

Necesitamos una función constructora. Se tiene que llamar constructor y se ejecuta cada vez que creemos un objeto

```
class Persona{

    constructor(nombre, apellido, edad){

        this.nombre = nombre
        this.apellido = apellido
        this.edad = edad
    }
}
```

this hace referencia al objeto

nombre del objeto = nombre del parámetro
apellido del objeto = apellido del parámetro
edad del objeto = edad del parámetro

Se le pueden asignar propiedades que no haya en los parámetros, pero siempre utilizando this para referenciar al objeto

```
this.datos = `${nombre} ${apellido} ${edad}`
```

CLASES - MÉTODOS

Los objetos pueden tener funciones asociadas a él. En ese caso se les denomina MÉTODOS

```
saludar() {
    return `Hola, me llamo ${this.nombre} y tengo ${this.edad} años`;
}
```

Los métodos deben ir dentro de la clase pero fuera del constructor

CREAR OBJETOS

Para crear un objeto utilizando la clase o plantilla se hace con la palabra reservada new y el nombre de la clase que queremos utilizar

```
const juan = new Persona('Juan', 'García', 23)
```

Una vez INSTANCIADO el objeto podremos acceder a sus propiedades y métodos utilizando la nomenclatura del punto o buscando su propiedad en el objeto

```
juan.nombre
juan.apellido
juan.edad
juan.datos
juan.saludar()
```

```
juan['nombre']
juan['apellido']
juan['edad']
juan['datos']
juan['saludar']()
```

Ejemplo de Clase con constructor, método y como se instancia un objeto de esa clase

```
class Persona{
    constructor(nombre, apellido, edad){
        this.nombre = nombre
        this.apellido = apellido
        this.edad = edad

        this.datos = `Me llamo ${nombre} ${apellido} y tengo ${edad} años`
    }

    saludar(){
        return `Hola, me llamo ${this.nombre} y tengo ${this.edad} años.`
    }
}

const juan = new Persona('Juan', 'García', 25)
const marta = new Persona('Marta', 'García', 35)
```

EJERCICIO – Clases

- Crea una clase Libro.
- La clase libro tendrá **título, autor, año y género**.
- Crea un método que devuelva toda la información del libro.
- Pide 3 libros y guárdalos en un array. Los libros se introducirán al arrancar el programa pidiendo los datos con prompt.
- Validar que los campos no se introduzcan vacíos.
- Validar que el año sea un número y que tenga 4 dígitos.
- Validar que el género sea: aventuras, terror o fantasía.
- Crea una función que muestre todos los libros.
- Crea una función que muestre los autores ordenados alfabéticamente.
- Crea una función que pida un género y muestre la información de los libros que pertenezcan a ese género usando el método que devuelve la información.

ARRAY – MÉTODOS MAS AVANZADOS

Ya conocemos el concepto de función por tanto podemos ver métodos mas avanzados.

- **.from(iterable)** - Convierte en array el elemento iterable. Un elemento iterable es aquel que se puede recorrer, por ejemplo un String. Cuando trabajemos con el DOM este método es muy útil

- **.sort([callback])** - Ordena los elementos de un array alfabéticamente(valor Unicode), si le pasamos un callback los ordena en función del algoritmo que le pasemos. Callback es una función que la está ejecutando otra función, en cambio si las funciones pertenecen a un objeto se les denomina métodos.

```
console.log(numbers.sort())

console.log(numbers.sort((a,b)=>a-b))

/* function menorMayor(a, b) {
  if (a-b < 0) return -1;
  if (a-b > 0) return 1;
  if(a == b) return 0;
} */

/* function mayorMenor(a, b) {
  if (b-a < 0) return -1;
  if (b-a > 0) return 1;
  if(b == a) return 0;
} */
```

ARRAY – MÉTODOS MAS AVANZADOS

- **.forEach(callback(currentValue, [index]))** - ejecuta la función indicada una vez por cada elemento del array.

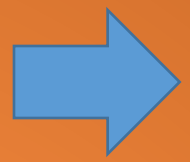
```
const numbers = [12, 25, 47, 84, 98]

numbers.forEach((number)=>console.log(number))
numbers.forEach((number, index) =>
    console.log(`${number} está en la posición ${index}
    `))
```

```
12
25
47
84
98
12 está en la posición 0
25 está en la posición 1
47 está en la posición 2
84 está en la posición 3
98 está en la posición 4
```

ARRAY – MÉTODOS MAS AVANZADOS

- **.some(callback)** - Comprueba si al menos un elemento del array cumple la condición.
- **.every(callback)** - Comprueba si todos los elementos del array cumplen la condición.

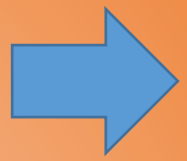
```
const words = ['HTML', 'CSS', 'JavaScript', 'PHP']

console.log(words.some(word => word.length>10));
console.log(words.every(word => word.length>3));
```



```
false
false
```

- **.map(callback)** - Transforma todos los elementos del array y devuelve un nuevo array.

```
const numbers = [12, 25, 47, 84, 98]

const numbers2 = numbers.map(number => number * 2)

console.log(numbers2);
```

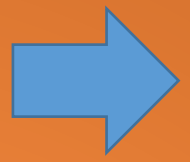


```
(5) […]
    0: 24
    1: 50
    2: 94
    3: 168
    4: 196
    length: 5
  ▶ <prototype>: Array []
```

ARRAY – MÉTODOS MAS AVANZADOS

- **.filter(callback)** - Filtra todos los elementos del array que cumplan la condición y devuelve un nuevo array.

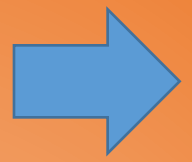
```
const numbers = [12, 25, 47, 84, 98];
const numbers2 = numbers.filter(number => number > 80);
console.log(numbers2);
```



```
▶ Array [ 84, 98 ]
```

- **.reduce(callback)** - Reduce todos los elementos del array a un único valor

```
const numbers = [1, 2, 3, 4, 5];
console.log(numbers.reduce((a,b)=>a+b));
```



```
15
```

Listado de métodos: https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Array

EJERCICIO – Arrays

- Convierte la cadena “Hola Mundo” en un array. Existen 2 métodos que permiten hacerlo. Hazlo con cada uno de ellos y asigna el resultado a 2 variables distintas.
- Crea un array que contenga las letras **b,c,z,a**. Muestra por consola el array ordenado.
- Crea un array que contenga los valores 1,8,100,300,3. Utilizando el mismo método anterior, muestra por consola el array ordenado. ¿Qué pasa? ¿por qué crees que es?.
- Crea un array de objetos de usuario. Concretamente serán 6 objetos. Los objetos estarán compuestos por **name** y por **online.** Online será true o false indicando si el usuario está online o no. Utilizando **reduce,** obtén en número de usuarios que están en línea.

SPREAD OPERATOR

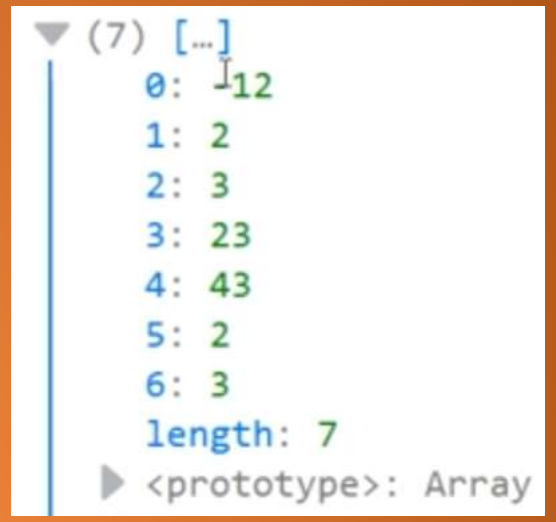
NUEVO EN EMACSCRIPT 6

Spread operator permite que una expresión sea expandida en situaciones donde se esperan múltiples argumentos (llamadas a funciones) o múltiples elementos (arrays literales).

SINTAXIS: se pone **…nombredelarray** que queremso expandir.

```
/*
Spread Operator (operador de expansión)

Su sintaxis es ...

*/

const numbers = [-12, 2, 3, 23, 43, 2, 3]

console.log(...numbers)
```

Sin Spread Operator



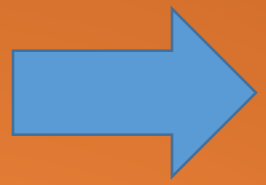
Con Spread Operator

SPREAD OPERATOR

NUEVO EN EMACSCRIPT 6

Enviar elementos de un array a una función

```
//Enviar elementos en un array a una función

const addNumbers = (a, b, c) => {
    console.log(a+b+c)
}

let numbersToAdd = [1,2,3]

addNumbers(numbersToAdd)
```

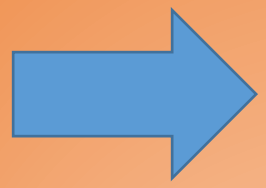


```
1,2,3undefinedundefined
```

Se debe a que la función **addNumbers** está cargando en el parámetro "**a**" todo el contenido del array, por tanto "**b**" y "**c**" serán "**undefined**".

¿Cómo podemos enviar todos los elementos del array a la función? **Usando Spread Operator**

```
addNumbers(...numbersToAdd)
```



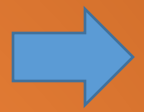
```
6
```

SPREAD OPERATOR

NUEVO EN EMACSCRIPT 6

Añadir un array a otro array

```
//Añadir un array a otro array
let users = ['javier', 'david', 'rosa', 'juan', 'mercedes']

let newUsers = ['marta', 'jaime', 'laura']

users.push(...newUsers)

console.log(users);
```



```
(8) […]
    0: "javier"
    1: "david"
    2: "rosa"
    3: "juan"
    4: "mercedes"
    5: "marta"
    6: "jaime"
    7: "laura"
    length: 8
```

Copiar arrays

```
//Copiar arrays
let arr1 = [1, 2, 3, 4,5]
let arr2 = [...arr1]
console.log(arr2);
```

Concatenar arrays

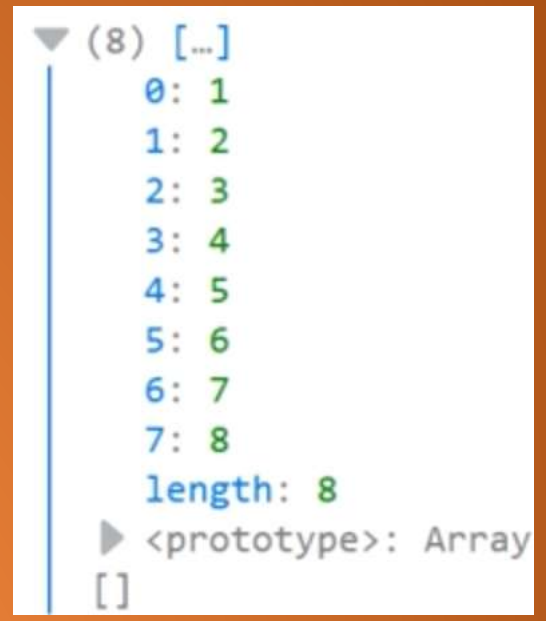
```
//Concatenar arrays
let arr1 = [1, 2, 3, 4, 5]
let arr2 = [6, 7, 8]

let arrConcat = [...arr1, ...arr2]
console.log(arrConcat);
```

SPREAD OPERATOR

NUEVO EN EMACSCRIPT 6

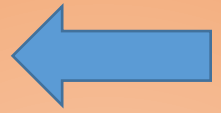
Enviar un número indefinido de argumentos a una función (parámetros REST)

```
// Enviar un número indefinido de argumentos a una función
(parámetros REST)

const restParms = (...numbers) => {
    console.log(numbers);
}

restParms(1,2,3,4,5,6,7,8)
```



Eliminar elementos duplicados

```
const numbers = [-12, 2, 3, 23, 43, 2, 3]

//Eliminar elementos duplicados
console.log([...new Set(numbers)])
```



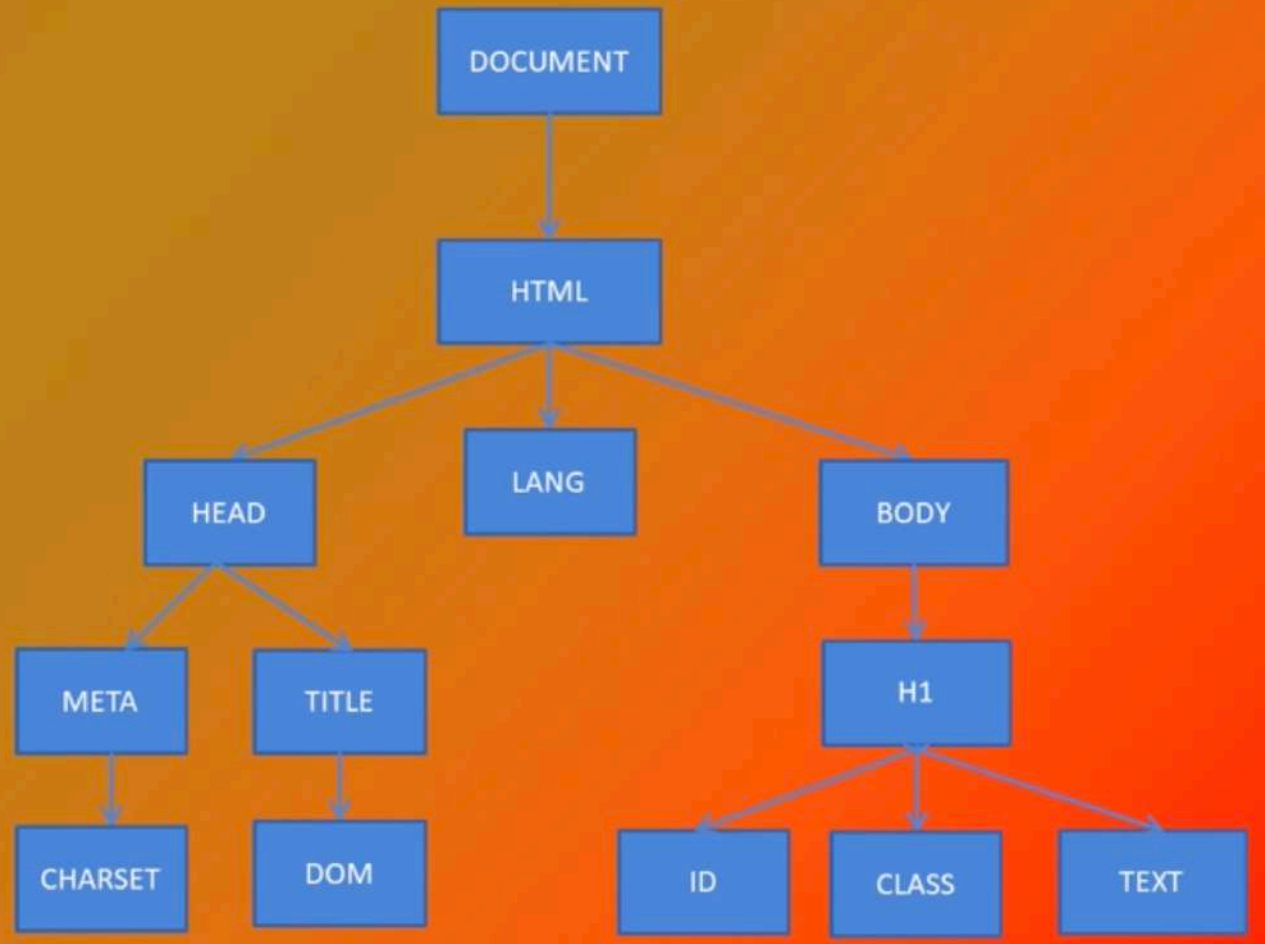
Set nos convierte el Array en un objeto **sin duplicados**. Para volver a convertirlo a Array es necesario poner los [ ] que aparecen

DOM (DOCUMENT OBJECT MODEL)

Es toda la estructura HTML del documento.
No es JavaScript, es una API (Aplication Programing Interface)

```html
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>DOM</title>
</head>
<body>
    <h1 id="title" class="title">DOM Document Object Model</h1>
</body>
</html>
```

DOM Document Object Model

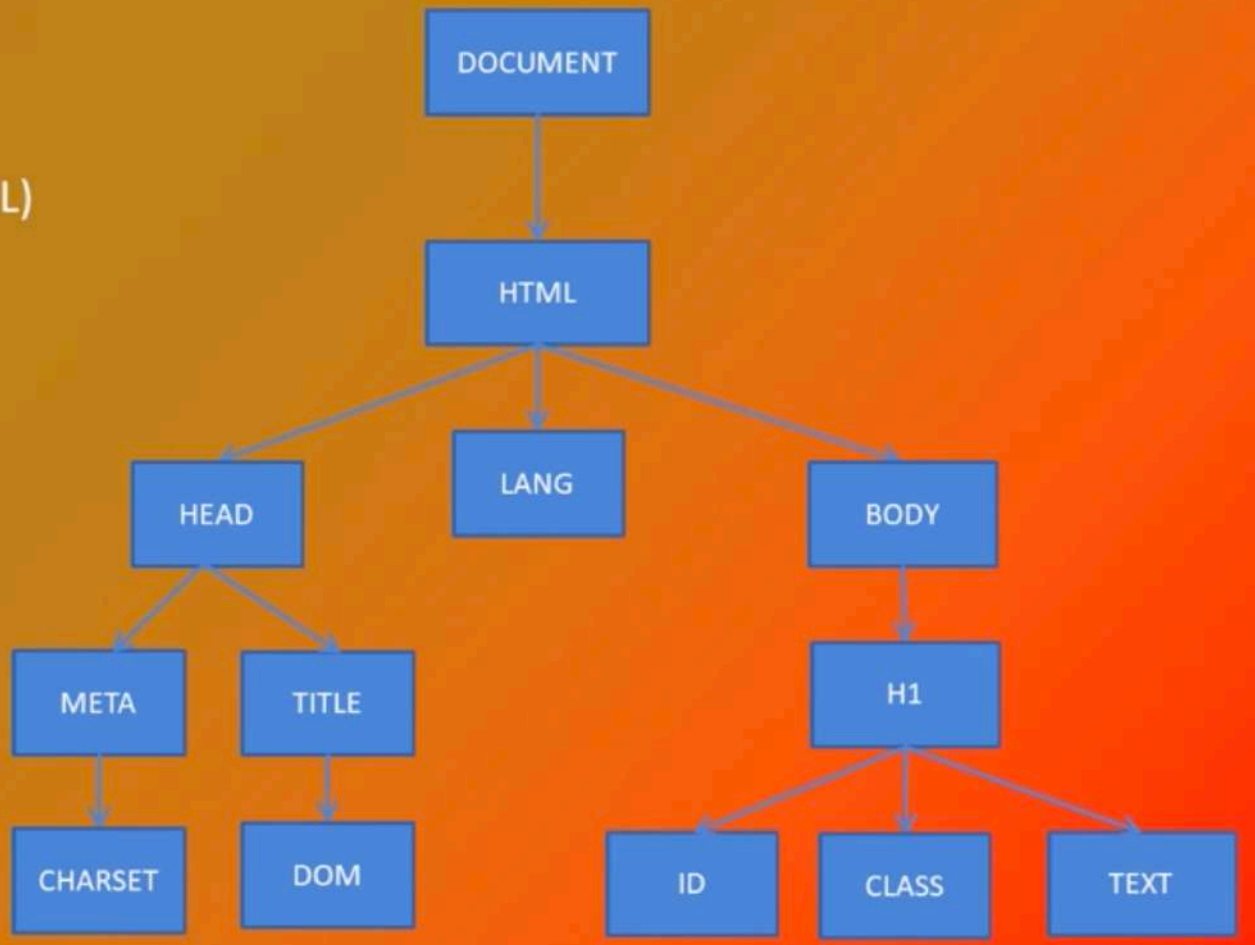


NODOS

Cada parte del arbol del documento es un NODO

Hay varios tipos de nodos, los más utilizados son:

- Element node – 1 (Cualquier etiqueta HTML)
- Text node – 3 (El contenido de la etiqueta)
- Comment node – 8 (Cualquier comentario en HTML)



Acceder a elementos/nodos

Acceso a un nodo mediante getElementById():

```
<html>
    <body>
        <p id="miParrafo">Esto es un parrafo</p>
    </body>
</html>
```

document.getElementById("miParrafo")

Si queremos obtener el valor abría que poner **document.getElementById("miParrafo").value**

Con **textContent** puedo modificar el valor del texto que se muestra en la página HTML. Por ejemplo:

const parrafo=document.getElementById("miParrafo");
parrafo.textContent='Texto modificado';

Acceder a elementos/nodos

Acceso a una colección de nodos mediante getElementsByTagName():

```
<p id="miParrafo">Esto es un parrafo</p>
<p>Otro parrafo</p>
<p>Tercer parrafo</p>
<table title="una tabla">
    <tr>
        <td>Una columna</td>
        <td>Otra Columna</td>
    </tr>
</table>
```

document.getElementsByTagName("td"), devuelve una colección de celdas de la tabla.

Acceder a elementos/nodos

Acceso a hijos de un nodo dado:

```
<p id=“miParrafo”>Esto es un parrafo</p>
<p>Otro parrafo</p>
<p>Tercer parrafo</p>
<table id=”miTabla” title=“una tabla”>
    <tr>
        <td>Una columna</td>
        <td>Otra Columna</td>
    </tr>
</table>
```

document.getElementById(“miTabla”).childNodes, devuelve una colección de nodos hijos de la tabla (tr,td).

Crear un nuevo Nodo

```
<p id="miParrafo">Esto es un parrafo</p>
<p>Otro parrafo</p>
<p>Tercer parrafo</p>
<table id="miTabla" title="una tabla">
    <tr>
    <td>Una columna</td>
    </tr>
</table>
```

Codigo Javascript:

```
laTabla = document.getElementById("miTabla");
miFila = document.createElement("tr"); //Creo fila
miCol = document.createElement("td"); //Creo columna
miTexto = document.createTextNode("El texto de la nueva columna");
miCol.appendChild(miTexto); //Agrego texto a la columna
miFila.appendChild(miCol); // Agrego columna a la fila
laTabla.appendChild(miFila); //Agrego fila a la tabla
```

Modificar un Nodo

```
<p id="miParrafo">Esto es un parrafo</p>
<p>Otro parrafo</p>
<p>Tercer parrafo</p>
<table id="miTabla" title="una tabla">
<tr>
<td>Una columna</td>
</tr>
```

Codigo Javascript:

```
miTabla = document.getElementById("miTabla");
viejoNodo = miTabla .getElementsByTagName("td")[0];
nuevoNodo = document.createElement("tr");
nuevoNodo.appendChild(document.createTextNode("Otro texto");
miTabla.replaceChild(viejoNodo, nuevoNodo );
```

Eliminar un Nodo

```
<p id=“miParrafo”>Esto es un parrafo</p>
<p>Otro parrafo</p>
<p>Tercer parrafo</p>
<table id=“miTabla” title=“una tabla”>
    <tr>
    <td>Una columna</td>
    </tr>
</table>
```

Codigo Javascript:

```
nodoABorrar = document.getElementsByTagName(“p”)[2];
document.removeChild(nodoABorrar);
```

EJERCICIOS DOM

Ejercicio1: Crea un formulario con la siguiente apariencia:

Es importante que sepas como añadir, modificar y borrar nodos del DOM

- Al introducir un nuevo elemento en el input y pulsar el botón **Añadir elemento**, este se añadirá a la lista.
- Al pulsar el botón **Borrar**, se eliminará el primer elemento de la lista.

Ejercicio2 y 3: Realizar el ejercicio 12 y 13 que se encuentran en el siguiente enlace:
https://uniwebsidad.com/libros/javascript/capitulo-5/ejercicios-sobre-dom