La sintaxis de JavaScript

JavaScript puede ser añadido a un documento HTML por medio de la etiqueta `<script>`, añadiendo el código directamente o por medio de un archivo externo.

Etiqueta script con código

Ejemplo de documento HTML con código JavaScript dentro de la etiqueta `script`

```
<!DOCTYPE html>
<html lang="en">
<head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-
scale=1.0">
        <title>Document</title>
</head>

<body>
        <h1>Hola mundo Javascript</h1>

        <!-- Código JavaScript -->
        <script>
        console.log('Hola mundo!')
        </script>
</body>
</html>
```

Etiqueta script con archivo

```
<!DOCTYPE html>
<html lang="en">
<head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-
```

```
scale=1.0">
        <title>Document</title>

        <!-- Código JavaScript -->
        <script>
        console.log('Hola mundo!')
        </script>

</head>

<body>
        <h1>Hola mundo Javascript</h1>

        <!-- JavaScript -->
        <script src="js/holamundo.js">
        </script>
</body>
</html>
```

Sintaxis JavaScript

A continuación se detallan los diferentes elementos de la sintaxis de JavaScript para la realización de programas informáticos.

1. Variables

Las **variables** permiten trabajar con información de diferentes tipos de datos:

- números
- textos
- caracteres
- boolean (verdadero o falso)
- estructuras de datos
- objetos

Para crear las **variables** utilizamos las palabras reservadas: `var`, `let`, `const`. Siendo let y const la manera recomendada desde la versión ES6.

Para escribir **comentarios** se usa doble barra `//` o barra y asterisco `/* */`.

```
// Number
var length = 50;
var length2 = 50.43;

// String (Cadenas de texto)
var nombre1 = "Rodrigo";
var nombre2 = "Juan";

// Booleans (true o false) (verdadero o falso)
var hasLastName = true;
var hasCar = false;

// Arrays              0        1       2        3
var productNames = ["Balón", "Silla", "Mesa", "Taza"]

// Objetos
var product1 = {
 name: "Balón Liga 2005",
 description: "Balón de Caucho Liga 2005 material importado
Noruega",
 price: 4.99,
 weight: 0.4,
 quantity: 4,
 hasOffer: true
}
```

Ejemplos `let` y `const`:

```
// ES6
var hola;
let adios;
let month = 'January';
const monday = 'MONDAY';
```

Para conocer el **tipo de dato** de una variable en particular se utiliza la palabra reservada `typeof`:

```
var miNumero = 5;
var miBoolean = false;
var miString = "Hola"
var miArray = ["Hola", "Adiós"]

var miVariable; // al no asignar valor será undefined
console.log(typeof miNumero) // number
console.log(typeof miBoolean) // boolean
console.log(typeof miString) // string
console.log(typeof miArray) // object
console.log(typeof miVariable) // undefined
```

2. Operadores

Los operadores son símbolos que permiten realizar operaciones con las variables, como por ejemplo operaciones aritméticas y lógicas.

```
// El operador suma (+)
var resultado1 = 5 + 4;

// El operador resta (-)
var resultado2 = 5 - 4;

// El operador multiplicación (*)
var resultado3 = 5 * 5;

// El operador exponente (**) (A partir de ES6)
var resultado4 = 5 ** 3; // Equivalente a  Math.pow(5,3)

var resultado4_2 = Math.pow(5, 3);

// Operador incremento
var resultado5 = 5;
resultado5++; // suma 1 a resultado5
resultado5++; // suma 1 a resultado5

// Operador decremento
var resultado6 = 5;
resultado6--; // resta 1 a resultado6
resultado6--; // resta 1 a resultado6
```

```
// Operador división
var resultado7 = 10 / 2;
var resultado8 = 20 / 5;

// Operador módulo (resto de la división)
var resultado9 = 10 % 2;
```

3. Funciones

Las **funciones** permiten reutilizar un código desarrollado. Se crean utilizando la palabra reservada `function` y un identificador:

```
// Función que imprime un saludo por consola
function function1() {
    console.log('Hi from function1');
}

function1(); // Para ejecutar una función escribimos su nombre y
paréntesis
function1(); // Podemos ejecutar una misma función más de una vez
function1(); // Podemos ejecutar una misma función más de una vez
```

Las funciones pueden tener **parámetros**, es decir, recibir uno o más datos como entrada:

```
function function2(name) {
    console.log('Hola ' + name);
    console.log('Adiós ' + name);
}

function2('Mike');
```

A su vez, las funciones pueden **retornar un dato**, por ejemplo el resultado de una operación:

```
function toFahrenheit(celsius) {
    return celsius * 9 / 5 + 32;
}
```

```
function toCelsius(fahrenheit) {
    return (5 / 9) * (fahrenheit - 32);
}

function suma(num1, num2){
    return num1 + num2;
}


console.log(toFahrenheit(30)); // 30°C son 86°F
console.log(toCelsius(86)); // 86°F son 30°C
console.log(toCelsius(86));
console.log(toCelsius(40));
console.log(suma(5, 5)); // 10
```

Las variables creadas dentro de una función son visibles dentro de la misma, pero no fuera:

```
function printName(){
    let age = 99; // variable visible solo dentro de la función
}

console.log(age) // Error, la variable no existe fuera de la función
```

4. Estructuras de control

4.1. Estructuras de control condicional

Las **estructuras de control condicional** permiten crear bifurcaciones en el código, permitiendo ejecutar código de manera condicional en base a ciertas condiciones.

```
// IF ELSE
var check2 = false;

if (check2) { // SI SE CUMPLE LA CONDICIÓN
        console.log('Condición check2 verdadera')
```

```
} else { // SI NO SE CUMPLE LA CONDICIÓN
        console.log('Condición check2 falsa')
}
```

Se pueden comprobar **múltiples condiciones** haciendo uso de estructuras `if`, `else if`, `else`:

```
if (check) {
    // Este código se ejecuta solo si la condición 1 se cumple
        console.log('Condición 1 verdadera')

} else if (check2) {
    // Este código se ejecuta solo si la condición 2 se cumple
        console.log('Condición 2 verdadera')

} else {
    // Sse ejecuta solo si la condición 1 y la condición 2 NO se
cumplen
        console.log('Ninguna condición se cumplió, ejecutamos el
código dentro de else.')
}
```

4.2. Estructuras de control iterativo

Las estructuras de control iterativo permiten repetir un fragmento de código más de una vez, en base a condiciones programadas.

Las estructuras for y while permiten iterar el código:

```
// Estructura for
for (let i = 0; i < 5; i+=2) {
        console.log("El valor de i es: " + (i+1));
}

// estructura while
var cont = 20;
while (20 < 10) {
        console.log("Iteración while valor cont: " + cont);
```

```
        cont++; // Incremento
}
```

Las palabras reservadas break y continue permiten alterar la iteración de un bucle for o while:

```
// Ejemplo break: se rompe el bucle y se deja de iterar
for (let i = 0; i < 10; i++) {
        if (i === 5) {
                break; // hace salirse del bucle
        }
        console.log("El valor de i es: " + i)
}


// Ejemplo continue: se salta a la siguiente iteración
for (i = 0; i < 10; i++) {
        if (i % 2 == 0) {
                continue; // salta a la siguiente iteración
        }
        console.log("Ejemplo 2 - El valor de i es: " + i)
}
```

5. Arrays

Una variable permite trabajar con un valor al mismo tiempo, si se quiere trabajar con más de un valor a la vez existen las estructuras de datos como los Arrays.

Un array es una estructura de datos que puede contener más de un dato al mismo tiempo, por ejemplo una lista de nombres.

```
var names = ["Bob", "John", "Raúl", "Wally", "Diego"]
```

Para acceder a los elementos del array se utiliza un índice, es decir, la posición del elemento teniendo en cuenta que empiezan en 0:

```
var firstName = names[0];  // Bob
var secondName = names[1]; // John
```

Utilizando estructuras de control se puede iterar una estructura de datos:

```
for (i = 0; i < names.length; i++) {
        names[i] = names[i] + " Editado"; // modifica cada nombre
        console.log("names[" + i + "]: " + names[i])
}
```