2. SINTAXIS BÁSICA DE JAVASCRIPT

ES6

Desarrollo Web en entorno cliente

IES Clara del Rey

El primer programa

- Insertar código en la página HTML

```
<SCRIPT LANGUAGE="JavaScript">
Aquí va el código
</SCRIPT>
```

- Incluir ficheros externos

```
<script type="text/javascript" src="rutaArchivo1.js"/>
<script type="text/javascript" src="rutaArchivo2.js"/>
```

Comentarios

- De una línea

```
// Esto es un comentario en Javascript de una línea
```

- Bloque de líneas

```
/* Esto es un comentario en Javascript
de varias líneas */
```

Escribir desde programa

```
console.log ("Hola mundo!");
```

```
window.alert("Hola, mundo!")
```

(se puede obviar *window*)

```
document.write("Hola, mundo!")
```

```
document.getElementById("demo").innerText="Hola, mundo!";
```

```
document.getElementById("demo").innerHTML="Hola, mundo!";
```

Nombrado de variables

- **Javascript es un lenguaje sensible a mayúsculas y minúsculas**
- Caracteres alfanuméricos (números y letras),
- el carácter subrayado o guión bajo (_)
- y el carácter dólar $
- No pueden comenzar por un carácter numérico.
- No podemos utilizar caracteres raros como el signo + , un **espacio** o un signo -

Declaración de variables

- Mediante la palabra reservada **var** (en desuso desde 2015): Se usa cuando se trata de asignar un ámbito de toda una función, o ámbito global.
- **let** : restringe su existencia al ámbito donde ha sido declarada. Sólo existirán dentro del bloque que las contenga. Usar "let" siempre por regla general, hasta que por su ámbito restringido no nos venga bien.
- **const**: crea una variable donde su valor es inalterable a lo largo del código.

Con objetos, la constante es el objeto, no sus propiedades

```
let op1;
const x = 5;
let operando1, operando2;
```

- **"use strict"** (*Modo estricto*): obliga a que todas las variables sean declaradas

Declaración de variables

Incorrecto:

```
const MIVARIABLE;
MIVARIABLE = 10;
console.log(MIVARIABLE);
```

Intenta asignar un valor a una constante ya asignada

Correcto:

```
const MIVARIABLE = 10;
console. log(MIVARIABLE);
```

Tipos de datos

- **Numéricos (tipo “number”)**

let x = 3.14;
let y = 3; let y = 123e-5;

Numéricos especiales: **NaN** , **Infinity**

- **Enteros grandes (tipo “bigint”)**: Añadir una letra “n” al final del entero.

let x = 123456789012345678901234 5n;

- **Booleanos (tipo “boolean”)** : true, false, 1 y 0.
- **Cadenas (tipo “string”)**: cero o múltiples caracteres delimitados por comillas dobles o simples

let miTexto = “Pepe se va a pescar”;

Datos primitivos

- Averiguar el tipo de una variable: **typeof**: ***alert(typeof num);***
- **Undefined**

let noInicializada;

- **Objetos**

```
let persona = {
  nombre: 'Miguel Angel',
  apellidos: 'Alvarez Sanchez',
  profesion: 'Informático'
}
```

- **function**

```
let miFuncion = function() {
  // Hacer alguna cosa...
}
```

- **null** (p.ej objetos o arrays no inicializados)

Operadores

- Aritméticos: + - * / % ++ – **
- Asignación: = += -= *= /= %= **=
- Comparación: == === != !== > < >= <= ?
- Concatenación de string: + +=
- Lógicos: && || !
- typeof , instanceof
- Cadenas y números:

```
let x = 5 + 5;  // 10
let y = "5" + 5;  // 55
let z = "Hello" + 5;  //Hello5
```

Asignación de variables desde programa

```
let person = prompt("Tu nombre", "Nombre de alumno");
```

```
let respuesta;
respuesta = confirm("¿Acepta nuestras condiciones?");
alert("Su contestación es " + respuesta);
```

Conversiones de tipos

```
let num = "100"; // Es una cadena
let num2 = "100.13"; // Es una cadena
let num3 = 11  // Es un entero
let n = parseInt(num); // Almacena un entero truncado
let n2 = parseFloat(num); // Almacena un decimal
let n3 = num3.toString(); // Almacena una cadena
```

Template Strings (novedad ES6)

Las Template Strings son cadenas literales que habilitan el uso de expresiones incrustadas. Con ellas, es posible utilizar cadenas de caracteres de más de una línea, y funcionalidades de interpolación de cadenas de caracteres.

Se delimitan con el caracter de comillas o tildes invertidas (‘), en vez de comillas simples o dobles

Interpolación de variables con $\{variable\}$

```
let firstName = "John";
let lastName = "Doe";

let text = `Welcome ${firstName}, ${lastName}!`;
```

Saltos de línea en template string

Permite cadenas de varias líneas

```
let text =
`The quick
brown fox
jumps over
the lazy dog`;
```

Con interpolación de expresiones:

```
let a = 5;
let b = 10;
console.log(`Quince es ${a + b} y
no ${2 * a + b}.`);
// "Quince es 15 y
// no 20."
```

Estructuras de control: if / else

```
if (hour < 18) {
  greeting = "Good day";
}
```

```
if (hour < 18) {
  greeting = "Good day";
} else {
  greeting = "Good evening";
}
```

```
if (time < 10) {
  greeting = "Good morning";
} else if (time < 20) {
  greeting = "Good day";
} else {
  greeting = "Good evening";
}
```

Estructuras de control: switch

```
switch (new Date().getDay()) {
  case 6:
    text = "Today is Saturday";
    break;
  case 0:
    text = "Today is Sunday";
    break;
  default:
    text = "Looking forward to the Weekend";
}
```

default no tiene que estar al final del último *case*

Estructuras de control: for

```
for (let i = 0; i < 5; i++) {
  text += "The number is " + i + "<br>";
}

var i = 5;
for (var i = 0; i < 10; i++) {
  // some code
}
// Here i is 10

let i = 5;
for (let i = 0; i < 10; i++) {
  // some code
}
// Here i is 5
```

Estructuras de control: while / do while

```
while (i < 10) {
  text += "The number is " + i;
  i++;
}

do {
  text += "The number is " + i;
  i++;
}
while (i < 10);
```

Estructuras de control: for in

Itera sobre propiedades de un objeto o de un array

```
const person = {fname:"John", lname:"Doe", age:25};

let text = "";
for (let x in person) {
  text += person[x];
}
```

text = "John Doe 25"

```
const numbers = [45, 4, 9, 16, 25];

let txt = "";
for (let x in numbers) {
  txt += numbers[x];
}
```

Estructuras de control: for of

Itera en estructuras de datos como Arrays, Strings, Maps, NodeLists

Sobre una cadena (string):

```
let language = "JavaScript";

let text = "";
for (let x of language) {
text += x;
}
```

x contiene cada letra en cada iteración (‘J’,‘a’,‘v’...)

Instrucciones Break / continue

La instrucción **break** dentro de un bucle hace que este se interrumpa inmediatamente, continuando al final del bucle.

```
for (i=0;i<10;i++){
    document.write (i)
    escribe = prompt("¿Continuar?(yes/no)", "yes")
    if (escribe == "no")
        break
}
```

La instrucción **continue** en un bucle es el de hacer retornar a la secuencia de ejecución a la cabecera del bucle, permitiendo saltar instrucciones del propio bucle.

```
for (let i = 0; i < 10; i++) {
  if (i === 3) { continue; }
  text += "The number is " + i + "<br>";
}
```

Funciones

Conjunto de instrucciones que se agrupan bajo un nombre de función.

- Se ejecuta sólo cuando es llamada por su nombre en el código del programa.
- Ayudan a estructurar los programas para hacerlos su código más comprensible y más fácil de modificar.
- Permiten repetir la ejecución de un conjunto de órdenes las veces que sea necesario sin necesidad de escribir de nuevo las instrucciones.

Ejemplo básico de sintaxis de función

```
// Llamada a la función, el valor de retorno
// se guarda en x
let x = myFunction(4, 3);

function myFunction(a, b) {
// La función devuelve el producto de a por b
  return a * b;
}
```

Error habitual en la llamada a funciones

```
<SCRIPT>
miFuncion()
</SCRIPT>
....
<SCRIPT>
function miFuncion(){
    //hago algo...
    document.write("Esto va bien")
}
</SCRIPT>
```

¿Es el orden del script el correcto?

Parámetros de funciones

- Los parámetros son los valores de entrada que recibe una función.
- Los parámetros pueden recibir cualquier tipo de datos, numérico, textual, boleano o un objeto.
- No se especifica el tipo del parámetro. Hay que tener un cuidado especial al definir las acciones que se realizan dentro de la función y al pasarle valores.
- Una función puede recibir tantos parámetros se quiera
- Se colocan los nombres de los parámetros separados por comas, dentro de los paréntesis.
- **Los parámetros de las funciones se pasan por valor**. Estamos pasando valores, no variables.

Múltiples parámetros

```
function escribirBienvenida(nombre,colorTexto){
    document.write("<FONT color='" + colorTexto + "'>")
    document.write("<H1>Hola " + nombre + "</H1>")
    document.write("</FONT>")
}

var miNombre = "Pepe"
var miColor = "red"
escribirBienvenida(miNombre,miColor)
```

Paso de parámetros por valor

```
function pasoPorValor(miParametro){
    miParametro = 32
    document.write("he cambiado el valor a 32")
}
var miVariable = 5
pasoPorValor(miVariable)
document.write (`el valor de la variable es: ${miVariable}`
```

¿Qué se imprime en la última sentencia?

Funciones flecha (arrow functions)

Es una alternativa compacta al uso de funciones tradicionales.

- Deben ser utilizadas solo en algunos contextos donde sean útiles.
- No son adecuadas para ser utilizadas como métodos.

Sintaxis general:

```
let mifuncion = () => {
  //código de la función
}

mifuncion();
```

Funciones flecha (cont)

Con varios parámetros

```
let saludo = (nombre, tratamiento) => {
  alert('Hola ' + tratamiento + ' ' + nombre)
}

//invocamos
saludo('Miguel', 'sr.');
```

Con un parámetro

```
let cuadrado = numero => {
  return numero * numero;
}
```

Parámetros con valores por defecto en funciones

```
function saludar(nombre = 'Miguel Angel') {
  console.log('Hola ' + nombre);
}

saludar();
```

Se puede seguir invocando a la función enviando un valor diferente como parámetro. Sirve para tener un parámetro definido si no se le pasa parámetro.

Orden en los parámetros predefinidos

El orden de los valores predeterminados a los parámetros importa

```
function potencia(base, exponente = 2) {
  let resultado = 1;
  for (let i = 0; i < exponente; i++) {
    resultado *= base;
  }
  return resultado;
}
console.log(potencia(3)); // indica 9 como resultado
console.log(potencia(3, 3)); // indica 27 como resultado

function potenciaMal(exponente = 2, base) {
  .......
}
console.log(potenciaMal(3)); // esto muestra NaN
```

Operador Rest

El operador rest sirve para obtener cualquier número de parámetros de una forma estructurada, **mediante un array de valores.**

Se escribe mediante tres puntos seguidos, como los puntos suspensivos de la escritura tradicional.

```
function max(...numeros) {
  console.log(numeros);
}
max(1, 2, 6);
max();
max(1, 5, 6, 7, 10001);
max("test", 4, true, 2000, "90");
```

Siempre te llega un array como valor del parámetro. Se usan los métodos conocidos para recorrido o manipulación del array.

Operador Rest con más parámetros

El operador rest nos ofrece en el array solamente los parámetros a los que no les hemos asignado un nombre

```
function enLista(buscar, ...nombres) {
  for(let i = 0; i < nombres.length; i++) {
    if(buscar === nombres[i]) {
      return true;
    }
  }
  return false;
}

console.log(enLista('Miguel', 'Diego', 'Noe', 'Miguel'));
//true
console.log(enLista('Miguel', 'Diego', 'Noe', 'Manolo'));
//false
```

Documentación

- w3schools
- Desarrollo Web. Manual de Javascript
- Desarrollo Web. Manual de ES6