Tema 6: JavaScript

1.- Introducción

El lenguaje JavaScript fue creado por Brendan Eich en 1995. Se desarrolló para el navegador Netscape 2. El primer Internet Explorer en ejecutarlo fue IE 4. La versión más reciente del lenguaje ha sido la ECMAScript 2023.

Se trata del lenguaje interpretado que se ejecuta en el navegador web. En su momento compitió con Visual Basic Script, que se ejecutaba en Internet Explorer, pero ahora es el único existente.

Aunque se parezca en el nombre, no tiene nada que ver con Java.

2.- Sintaxis básica de JavaScript

Generalidades

La sintaxis básica está basada en la de C, al igual que la de Java. Por eso es fácil familiarizarse con ella una vez conocemos Java. Algunas observaciones:

- Aunque no es imprescindible, las líneas de código deberían terminar en punto y coma.
- Una instrucción puede cortarse después de un operador. Así, la siguiente instrucción ocupa dos líneas y es correcta:

```
document.getElementById("contenedor").innerHTML =
      "¡Hola, Mundo!";
```

- Los bloques de código están encerrados entre {}.
- Los comentarios se escriben igual que en Java:
  - // Para comentarios de una línea.
  - /* */ Para comentarios de varias líneas.
- Podemos forzar al navegador a que nos obligue a tener una buena sintaxis escribiendo al principio del código JavaScript la línea:

```
"use strict"
```

Variables

JavaScript es un lenguaje débilmente tipado. Nos permite usar una variable sin haberla declarado antes. Sin embargo, deberíamos siempre declarar las variables con la palabra clave *let* o *const* (si se trata de una constante).

```
let precio = 10.25;
const IVA = 21;
```

En código antiguo veremos que también se usa la palabra clave *var*, pero está desaconsejado su uso si no es que nos vemos obligados a soportar navegadores antiguos.

Podemos declarar varias variables en una misma línea:

```
let nombre = "Carlos", apellidos = "Sogorb", dni = "10100100T";
```

La instrucción anterior también puede cortarse en varias líneas:

```
let nombre = "Carlos",
      apellidos = "Sogorb",
      dni = "10100100T";
```

Ámbitos de las variables

Al declarar una variable usando *let* o *const*, su ámbito es el bloque {} en el que se encuentra, igual que en Java.

Cuando usamos la palabra clave *var* para declarar una variable, su ámbito es global y puede ser usada incluso antes de ser declarada.

Tipos de datos en JavaScript

En JavaScript disponemos de estos tipos de datos:

- String
- Number
- Bigint
- Boolean
- Undefined
- Null
- Object

El tipo String

Es idéntico al de Java (una cadena de texto). Para delimitar el texto se pueden usar indistintamente comillas simples o dobles. Por ejemplo:

```
let color = "Rojo";
let nombre = 'Carlos';
```

Las cadenas (strings) pueden ir indistintamente entre comillas simples o dobles.

El tipo Number

En JavaScript sólo hay un tipo de dato numérico. Todos los números se almacenan como números decimales (de punto flotante), ocupando 64 bits. Por ejemplo:

```
let peso = 80;
let estatura = 1.75;
```

El tipo BigInt

Se trata de un tipo reciente (ES2020) que nos permite almacenar valores enteros muy grandes que, si los almacenásemos en un *Number*, perderían precisión. Un ejemplo de variable *BigInt* puede ser:

```
let x = BigInt(123456789123456789123456789);
```

El valor también puede indicarse escribiendo una "n" al final del número:

```
let y = 123456789123456789123456789n;
```

El tipo Boolean

Las variables booleanas son iguales que en Java. Pueden tomar los valores true o false:

```
let gratis = true;
let autorizado = false;
```

El tipo Undefined

Al declarar una variable sin darle un valor inicial, ésta tendrá el valor *undefined*.

El tipo Null

El valor *null* lo asignamos a una variable cuando queremos indicar que no tiene valor.

El tipo Object

Los objetos en JavaScript se escriben con llaves {} y tienen propiedades (atributos) y métodos. Las propiedades se escriben como pares nombre:valor, separados por comas. Por ejemplo:

```
const coche = {tipo:"Peugeot", modelo:"206", color:"plata"};
const cliente = {nombre:"Luis Pérez", apellidos: "Pérez Fernández",
      dni:"10200200H", visa:"1111-2222-3333-4444"};
```

No hay problema en declarar un objeto en varias líneas:

```
const cliente = {
      nombre:"Luis",
      apellidos: "Pérez Fernández",
      dni:"10200200H",
      visa:"1111-2222-3333-4444"
};
```

Hay dos formas de acceder a una propiedad de un objeto, usando el punto (.) o los corchetes ([]). Por ejemplo, para mostrar el nombre del cliente anterior podemos usar estas formas:

```
console.log(cliente.nombre)
console.log(cliente["nombre"])
```

Métodos de un objeto

Un método de un objeto es una función guardada como una propiedad. Por ejemplo:

```
const cliente = {
      nombre:"Luis",
      apellidos: "Pérez Fernández",
      dni:"10200200H",
      visa:"1111-2222-3333-4444",
      nombreCompleto: function(){
            return this.nombre + " " + this.apellidos;
      }
};
```

Para acceder a un método del objeto, usamos la notación de punto:

```
cliente.nombreCompleto();
```

Arrays en JavaScript

A diferencia de Java, los arrays se definen con corchetes []:

```
const coches = ["Renault", "Seat", "Volvo"];
```

Los arrays no tienen un número fijo de elementos. Por ejemplo, podemos añadir un elemento extra al array coches:

```
coches[3] = "Peugeot";
```

Para acceder a los elementos del array, ponemos su índice entre corchetes:

```
console.log(coches[3]);
```

JavaScript es un lenguaje con tipado dinámico

En JavaScript una misma variable puede ser usada para almacenar diferentes tipos de datos. Por ejemplo, con la siguiente instrucción declaramos una variable *x* sin tipo:

```
let x;       // Ahora x vale undefined
```

Ahora le asignamos un número, por lo que su tipo pasa a ser *Number*:

```
x = 10;       // Ahora x es de tipo Number
```

Más adelante podemos asignar a la variable *x* un *String*:

```
x = "Carlos";       // Ahora x es de tipo String
```

Operadores aritméticos

JavaScript usa los mismos operadores aritméticos que Java: +, -, *, / y %. Además añade el operador potencia, **.

```
3**2 = 3 elevado a 2 = 9
```

También podemos usar ++, --, +=, -=, etc.

Operadores de comparación

Los operadores de comparación en JavaScript son los mismos que en Java: ==, !=, >, <, etc. Además se añaden dos operadores nuevos:

=== Mismo valor y tipo

!== Distinto valor o distinto tipo

En contraposición a:

== Mismo valor

!= Distinto valor

Podemos usar operadores de comparación con Strings, que lo que hacen es comparar alfabéticamente. Por ejemplo:

```
let texto1 = "A";
let texto2 = "B";
let resultado = texto1 < texto2;
// resultado vale true.
```

Operadores lógicos

Los operadores lógicos son los mismos que en Java: &&, || y !.

Concatenación de variables

Se pueden concatenar variables con "+". Si se concatena un número con un String, el número se tratará como un String. Por ejemplo:

```
let x = "5" + 2 + 3;
El resultado es el String "523"
```

O también:

```
let x = 16 + 4 + "Paco";
El resultado es el String "20Paco"
```

Funciones en JavaScript

Las funciones en JavaScript siempre se declaran con la palabra *function* delante. No se especifican los tipos de los parámetros ni el tipo devuelto. Dentro de la función los parámetros se tratan como variables locales.

Ejemplo

En este ejemplo usamos una función *producto* que recibe dos números y devuelve el resultado de multiplicarlos:

```
function producto(num1, num2) {
            return num1 * num2;
      }

console.log(producto(10, 12));
```

3. Instrucciones condicionales y bucles

Condicionales if, else if, else

La sintaxis es exactamente igual que en Java. Por ejemplo:

```
if(color == "verde"){
      console.log("Adelante");
}else if(color == "naranja"){
      console.log("Precaución");
}else{
      console.log("Deténgase")
}
```

Condicionales switch

Funcionan igual que en Java, sólo con una observación: usan la comparación estricta (===). Los valores deben ser del mismo tipo.

Por ejemplo:

```
let nombre, mes = 1;

switch (mes) {
      case 1:
            nombre = "Enero";
            break;
      case 2:
            nombre = "Febrero";
            break;
      default:
            nombre = "Algún mes entre marzo y diciembre";
}

console.log(nombre);
```

Bucles for

Usan la misma sintaxis que Java.

Ejemplo

Con el siguiente bucle for mostramos los 10 primeros múltiplos de 7.

```
for (let i = 1; i <= 10; i++) {
      console.log(i * 7);
}
```

Bucles for in

Los bucles "for in" pueden usarse para recorrer todas las propiedades de un objeto. La sintaxis es:

```
for (let clave in objeto) {
      ...
}
```

Ejemplo

Mostramos los valores de las propiedades de una variable objeto llamada *cliente*.

```
const cliente = {
      apellidos: "Sogorb Valls",
      visa: "1111-2222-3333-4444"
};

for(let campo in cliente){
      console.log(campo);
}
```

Bucles for of

El bucle "for of" itera sobre los valores de un objeto iterable, como un array o un String. La sintaxis es:

```
for (let variable of array) {
      ...
}
```

En cada iteración del bucle, la variable toma el valor de una posición del array, empezando desde el principio y llegando hasta el final.

Ejemplo

El siguiente código muestra por la consola los colores almacenados en el array.

```
const colores = ['Azul', 'Verde', 'Amarillo', 'Rojo', 'Naranja'];

for (let color of colores) {
      console.log(color);
}
```

Recorrer un array con el método Array.forEach()

Los arrays tienen un método llamado *forEach()*. Este método llama a una función una vez para cada elemento del array. Este tipo de función se denomina función de *callback*.

Ejemplo

Modificamos el ejemplo anterior para mostrar los colores del array colores usando el *forEach()*.

```
const colores = ['Azul', 'Verde', 'Amarillo', 'Rojo', 'Naranja'];

colores.forEach(mostrar);

function mostrar(value) {
      console.log(value);
}
```

Bucles while y do-while

La sintaxis de los bucles *while* y *do-while* es idéntica a la de Java.

Ejemplo

Mostramos los 10 primeros múltiplos de 5 usando *while*:

```
let i = 1;

while (i < 10) {
      console.log(i * 5);
i++;
}
```

Y usando *do-while*:

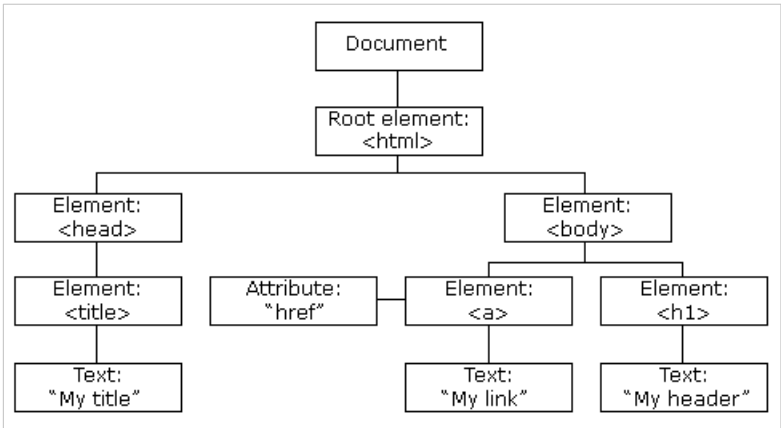
```
let i = 1;

do {
      console.log(i * 5);
      i++;
} while (i <= 10);
```

Instrucciones break y continue

Estos comandos hacen lo mismo que en Java.

4. EL DOM. Selección y acceso a elementos

El DOM (Document Object Model) a la interfaz es una interface que trata un documento HTML o XML como una estructura de árbol donde cada nodo es un objeto representando una parte del documento.



Los métodos del DOM permiten acceso desde programa al árbol. Con ellos se puede cambiar la estructura, el estilo o el contenido de un documento. Los nodos pueden tener manejadores de eventos (listeners) asignados. Una vez que un evento se produce, los manejadores de eventos se ejecutan.

El objeto *document* del DOM

El objeto *document* es el propietario de todos los otros objetos en nuestra página web. Representa nuestra página.

Si queremos acceder a cualquier elemento en un página web, siempre empezaremos accediendo al objeto *document*.

Acceder a un elemento

Podemos acceder a cualquier elemento de la página con el métod *document.getElementById()*. A este método le pasamos el id del elemento que queramos seleccionar.

Una vez seleccionado un elemento, podemos leer o modificar el contenido del elemento con la propiedad *innerHTML*.

Ejemplo

```
<body>
    <h1>Ejemplo de contenido dinámico</h1>
    <p id="explicacion">Este es un párrafo cuyo contenido va a cambiarse
            usando JavaScript.</p>

    <script>
        document.getElementById("explicacion").innerHTML = "Este texto se ha
            escrito mediante JavaScript.";
    </script>
</body>
```

Otros métodos de acceso

Seleccionar elementos por nombre de etiqueta

Podemos seleccionar todas las etiquetas de un tipo con el método *getElementsByTagName()*. Por ejemplo, podemos seleccionar todas las etiquetas "p" de una página:

```
document.getElementsByTagName("p")
```

Seleccionar elementos por nombre de clase

Podemos seleccionar todos los elementos de una clase con el método *getElementsByClassName()*. A este método le proporcionamos el nombre de clase entre paréntesis. Por ejemplo, si queremos seleccionar todos los elementos cuya clase sea "resaltado", ejecutaremos:

```
document.getElementsByClassName("resaltado");
```

Seleccionar elementos mediante selectores CSS

La manera más completa para seleccionar elementos es mediante un selector CSS, usando los dos métodos *querySelector()* y *querySelectorAll()*.

La siguiente orden selecciona el primer div con la clase "resultado":

```
document.querySelector("div.resultado")
```

Esta orden selecciona todos los divs con la clase "resultado":

`document.querySelectorAll("div.resultado")`: Selecciona todos los div con la clase "resultado".

5. Modificación de un atributo HTML

Podemos modificar un atributo de un elemento HTML accediendo al elemento y añadiendo un punto y el nombre del atributo que queremos modificar.

```
document.getElementById(id).atributo = nuevo_valor
```

Ejemplo

Aquí podemos ver un trozo de código en el que, mediante JavaScript, se modifica la imagen que muestra una etiqueta <img<, cambiando su atributo *src*.

```
<body>

    <img id="imagen" src="imgs/flores/geranio.jpg">

    <script>
        document.getElementById("imagen").src = "imgs/flores/azucena.jpg";
    </script>

</body>
```

6. Modificación de un estilo CSS

Podemos modificar un estilo CSS de un elemento HTML accediendo al elemento y añadiendo un punto y la palabra "style" y otro punto y la propiedad que queremos modificar.

```
document.getElementById(id).style.propiedad = nuevo_estilo
```

Ejemplo

Aquí podemos ver un trozo de código en el que, mediante JavaScript, se modifica el color del encabezado de la página.

```
<body>

    <h1 id="encabezado">Este título cambia de color con JavaScript.</h1>

    <script>
        document.getElementById("encabezado").style.color = "red";
    </script>

</body>
```

7. Eventos en JavaScript

Asignar eventos usando el DOM

Podemos asignar un evento a un elemento del documento seleccionándolo y añadiendo un punto y el tipo de evento. Por ejemplo, si queremos que al hacer clic sobre un botón se ejecute una función *mostrar_mensaje()*, escribiremos esto:

```
document.getElementById("boton").onclick = mostrar_mensaje;
```

Observa que al final del nombre de la función no se ponen paréntesis.

Ejemplo

Hacemos que al pulsar un botón se muestre un mensaje con el texto "¡Bien pulsado!".

```
<body>

    <h1>Evento onclick</h1>

    <button id="boton">Púlsame</button>

    <script>
        document.getElementById("boton").onclick = mostrar_mensaje;

        function mostrar_mensaje() {
            alert("¡Bien pulsado!");
        }
    </script>
</body>
```

Asignar eventos directamente en un elemento

Una manera más antigua de asignar elementos es usando el nombre del evento como atributo en la etiqueta. En este caso, al nombre de la función que queramos ejecutar sí le añadiremos paréntesis.

Ejemplo

El ejemplo anterior podría escribirse así:

```
<body>

    <h1>Evento onclick</h1>

    <button id="boton" onclick="mostrar_mensaje()">Púlsame</button>

    <script>
        function mostrar_mensaje() {
            alert("¡Bien pulsado!");
        }
    </script>
</body>
```

Añadir un manejador de eventos

Otra forma de gestionar eventos en JavaScript es con el método addEventListener(). A este método le pasaremos como parámetros el tipo de evento y el nombre de la función que lo manejará. Por ejemplo:

```
document.getElementById("boton").addEventListener("click",
      mostrar_mensaje);
```

Observar que el tipo de evento no lleva la palabra "on" delante.

Igualmente, podemos eliminar un manejador de eventos con el método *removeEventListener()*.

Observación

Si necesitamos pasar algún parámetro a la función manejadora del evento, lo podemos hacer indicando una función "anónima" que llama a la función manejadora y le pasa los parámetros.

```
element.addEventListener("click", function(){ miFuncion(p1, p2); });
```

Ejemplo

El ejemplo anterior podría escribirse así:

```
<body>

    <h1>Evento onclick</h1>

    <button id="boton">Púlsame</button>

    <script>
        document.getElementById("boton").addEventListener("click",
            mostrar_mensaje);

        function mostrar_mensaje() {
            alert("¡Bien pulsado!");
        }
    </script>
</body>
```

8. Algunos tipos de eventos

Los eventos onmouseover y onmouseout

Estos eventos se producen cuando el usuario pone el puntero del ratón encima de un elemento (onmouseover), o lo quita (onmouseout).

Ejemplo

Hacemos que, al pasar el puntero por encima de una imagen se le ponga borde rojo. Y al quitarlo, se le quite el borde. Usamos la variable *this*, que apunta al elemento que ha lanzado el evento.

```
<body>

    <h1>Eventos onmouseover y onmouseout</h1>

    <img src="imgs/planetas/saturno.png" alt="Saturno"
      onmouseover="poner_borde(this)" onmouseout="quitar_borde(this)">

    <script>
        function poner_borde(x) {
            x.style.border = "5px solid red";
        }

        function quitar_borde(x) {
            x.style.border = "none";
        }
    </script>

</body>
```

Los eventos onfocus y onfocusout

Se producen cuando un elemento recibe el foco (onfocus) o lo pierde (onfocusout).

Ejemplo

En un formulario, hacemos que la casilla de texto que tiene el foco cambie su color de fondo a amarillo. Y recupere el fondo blanco al perder el foco.

```
<body>
    <h1>Eventos onfocus y onfocusout</h1>

    <form action="">

        <label for="apellidos">Apellidos</label><br>
        <input type="text" name="apellidos" id="apellidos"
            onfocus="resaltar(this)" onfocusout="resetear(this)"><br><br>

        <label for="nombre">Nombre</label><br>
        <input type="text" name="nombre" id="nombre" onfocus="resaltar(this)"
            onfocusout="resetear(this)"><br><br>

        <input type="submit" value="Enviar">

    </form>

    <script>
        function resaltar(x) {
            x.style.backgroundColor = "yellow";
        }

        function resetear(x) {
            x.style.backgroundColor = "white";
        }
    </script>
</body>
```

El evento oninput

El evento oninput sucede cuando cambia el contenido de un elemento en un formulario.

Ejemplo

Mientras se escribe en un input text, se ejecuta una función que convierte el texto introducido a mayúsculas.

```
<body>
    <h1>Evento oninput</h1>

    <form action="">
        <label for="nombre">Nombre</label><br>
        <input type="text" name="nombre" id="nombre"
            oninput="mayusculas(this)"><br><br>

        <input type="submit" value="Enviar">
    </form>

    <script>
        function mayusculas(x) {
            x.value = x.value.toUpperCase();
        }
    </script>
</body>
```

El evento onchange

Este evento es similar al oninput, sólo que éste se lanza cuando tras modificar un elemento, pierde el foco.

Ejemplo

Cuando se escribe en un input text, y el usuario quita el foco de la casilla, se ejecuta una función que convierte el texto introducido a mayúsculas.

```
<body>
    <h1>Evento onchange</h1>

    <form action="">
        <label for="nombre">Nombre</label><br>
        <input type="text" name="nombre" id="nombre"
            onchange="mayusculas(this)"><br><br>

        <input type="submit" value="Enviar">
    </form>

    <script>
        function mayusculas(x) {
            x.value = x.value.toUpperCase();
        }
    </script>
</body>
```

9. Agregar y eliminar elementos del DOM

Crear un elemento y agregarlo al documento

Podemos crear un elemento nuevo con el método *createElement()*. Por ejemplo, podemos crear un elemento <p> con la instrucción:

```
const parrafo = document.createElement("p");
```

Luego, podemos añadir este elemento nuevo a otro mediante el método *appendChild()*.

```
document.getElementById("contenedor").appendChild(parrafo);
```

También podríamos añadirlo al final del documento con:

```
document.body.appendChild(parrafo);
```

Ejemplo

Agregamos a un div (con el id "contenedor") un nuevo párrafo.

```
<body>
    <h1>Agregar elementos a un elemento del documento</h1>

    <div id="contenedor">

    </div>

    <script>
        const parrafo = document.createElement("p");
        parrafo.innerText = "Este párrafo se ha añadido al div 'contenedor'
            mediante JavaScript.";
        parrafo.style.color = "salmon";

        document.getElementById("contenedor").appendChild(parrafo);
    </script>

</body>
```

Ejemplo

Agregamos al final del documento un nuevo párrafo, generado mediante JavaScript.

```
<body>
    <h1>Agregar elementos al documento</h1>

    <img src="imgs/planetas/saturno.png" alt="Saturno"
      onmouseover="poner_borde(this)" onmouseout="quitar_borde(this)">

    <script>
        const parrafo = document.createElement("p");
        parrafo.innerText = "Este párrafo se ha añadido al final del documento
            con JavaScript.";
        parrafo.style.color = "salmon";

        document.body.appendChild(parrafo);
    </script>

</body>
```

Eliminar un elemento del documento

Podemos eliminar un elemento del DOM accediendo a él y aplicando el método *remove()*.

Ejemplo

En este ejemplo, al clicar en un párrafo cuyo id es "eliminar" lo quitamos del documento.

```
<body>
    <h1>Eliminar elementos del documento</h1>

    <p id="eliminar" onclick="eliminar(this)">Este párrafo será eliminado
      mediante JavaScript</p>

    <script>
        function eliminar(x){
            x.remove();
        }
    </script>

</body>
```