JavaScript

4.0 Introducción

POR QUÉ

JavaScript es un lenguaje de secuencias de comandos interpretado. Una recopilación de código JavaScript no es más que una secuencia de líneas que a veces finalizan con un punto y coma. A esto le llamamos *secuencia de comandos.* Un JavaScript document puede contener todas las secuencias de comandos que quieras.

JavaScript es un lenguaje de *secuencias de comandos* que distingue entre las mayúsculas y minúsculas, lo que significa que no se compila (como lo hace Java), sino que se ejecuta al momento.

Una secuencia de comandos está formada por varias líneas juntas y los bloques de código se delimitan con llaves {}.

JavaScript y Java

Aunque parezca que JavaScript y Java estén relacionados, en realidad son muy diferentes. La diferencia principal es que JavaScript no es compatible con el paradigma orientado a objetos del desarrollo de software. Java es un lenguaje de escritura fuertemente tipado mientras que las variables de JavaScript están tipadas dinámicamente (más información en otros apartados). Básicamente, los objetos de JavaScript son dinámicos; es decir, la cantidad de métodos y miembros de datos de un objeto pueden cambiar mientras se ejecuta.

CÓMO

JavaScript se puede insertar en un elemento <style> o puede usarse en un archivo aparte. Puedes insertar el código, aunque no es demasiado útil, ya que debes escribirlo entero para cada uno de los botones. Hay una forma mejor de escribir las secuencias de comandos: con

funciones independientes en las etiquetas de cabecera (`<head>…</head>`). También puedes escribirlas en un archivo aparte.

INTÉNTALO
A continuación, te ofrecemos un ejemplo de una secuencia de comandos insertada para crear un elemento "Button" y abrir una ventana emergente al hacer clic en él.

```
<input type ="button" id="button1" value="Pulsar" onclick="alert('Has
hecho clic en el botón’);" />
```

El valor onclick se refiere a la acción que debe producirse al hacer clic en este botón.

> ADVERTENCIA: No debes usar atributos de evento, ya que mezclan el lenguaje HTML con JavaScript y hacen que los documentos sean más difíciles de leer y depurar y, a la vez, ocupen más.

A continuación, mostramos cómo escribir el mismo código JavaScript en un elemento de estilo.

```
<head>
  <script>
              function doSomething() {
                            alert('Has hecho clic en el botón’) ;
              }
      var button1 = document.getElementById('button1');
      button1.onclicked = doSomething;
  </script>
</head>

<input type ="button" id="button1" value="Pulsar" />
```

La funcionalidad de este código es muy parecida a la que hemos visto antes, pero se trata de una forma más modular de escribir código. No hace falta que te digamos que este es nuestro método preferido.

4.1 Acceso a elementos en JavaScript

POR QUÉ
Si tienes que encontrar un objeto asociado a un elemento de formulario en XHTML en JavaScript, puedes hacerlo de varias formas. El Document Object Model (DOM) consiste en usar matrices `Forms` y `Elements` del objeto `Document`, que hace referencia al objeto `Window`.
Para dejarlo claro: el objeto `Window` es la ventana en que se muestra el contenido XHTML.

Todos los objetos `Window` tienen una propiedad llamada `Document` que, a la vez, tienen una matriz `Forms` en que cada elemento representa un formulario del documento. El elemento de matriz `Forms` tiene como propiedad una matriz de `Element` con objetos que representan los elementos del formulario XHTML, como botones o menús, por ejemplo.

CÓMO

Los elementos de un documento son objetos que tienen tanto datos como operaciones. Los datos se conocen como propiedades *y* las operaciones, como métodos.
Mira el siguiente ejemplo:

```
<input type="text" name="age">
```

En este ejemplo, `type` y `name` son las propiedades del objeto mientras que "text" y "age" son los valores.

La dirección DOM es la del objeto JavaScript. Puedes acceder a elementos que estén en un nivel más profundo en el árbol DOM mediante la propiedad de punto. El nombre o ID único del elemento se usa para referirse al elemento en sí. Para hacerlo, se pueden usar métodos como getElementByName() o getElementById().

Mira el siguiente ejemplo:

```
var dom = document.getElementById("button1");
```

Esta línea de código se usa para acceder al elemento que tiene el nombre de identificación o ID único *button1*, que a la vez es parte del documento.

INTÉNTALO

<u>Para contar la cantidad de casillas de verificación de un formulario:</u>

Esta secuencia de comandos crea una matriz implícita de tres elementos (*colores)* mediante una casilla de verificación. Todos los elementos tienen el mismo nombre e ID, con lo que es más fácil acceder a ellos en cualquier momento de la secuencia de comandos. Las direcciones específicas de los elementos se guardan usando índices de matrices cuya longitud va desde 0 hasta length-1.

Debido a que el tipo de entrada es una casilla de verificación, todos los elementos tienen además una propiedad `"checked"` *(marcada).* Esta `propiedad` se configura con el valor true si la casilla está marcada o false si no lo está.

```
<form id="FavoriteColor">
  <input type="checkbox" name="color"
      value="rojo" /> Rojo
  <input type="checkbox" name="color"
      value="azul" /> Azul
  <input type="checkbox" name="color"
      value="verde" /> Verde
</form>
```

```
var numChecked = 0’
var dom = document.getElementById(“FavoriteColor");
for (i = 0; i < dom.color.length; i++)
     if (dom.car[i].checked)
          numChecked++;
```

4.2 Variables, operadores y comentarios

4.2.1 Variables

POR QUÉ

Para ponerlo fácil: las variables contienen valores y, antes de usar una, tienes que declararla.
JavaScript es un lenguaje de tipado dinámico, lo que significa que las variables se pueden usar con cualquier finalidad, ya sea un valor o una referencia a un objeto.
Si un valor se asocia a una variable, adquiere el tipo de dicho valor automáticamente.
JavaScript se divide en dos tipos:
Tipos primitivos: Number, String, Boolean, Undefined y Null
Tipos de referencia: Objects, Arrays y Functions

Aspectos importantes sobre las variables:

- JavaScript distingue entre mayúsculas y minúsculas, mientras que HTML y CSS, no.
- Las palabras clave (como var, document, etc.) no se pueden usar como nombres de variables.
- El valor predeterminado de una variable (si no se especifica otro) es Undefined (sin definir).
- Las cadenas deben declararse entre comillas.

Alcance de una variable:

- Variables locales:
  Al declarar una variable en una función, solo puede accederse en dicha función y deja de ser válida cuando se cierra la función.
- Variables globales
  Al declarar una variable en el nivel superior de un archivo de secuencias de comandos, esta se vuelve accesible desde cualquier parte del programa JavaScript.
- Las variables que no se declaran se convierten en globales automáticamente.

CÓMO

Para asignar un valor a una variable, puede hacerse directamente (en cuyo caso el intérprete implícitamente declara que es una variable) o bien se puede añadir a una sentencia que empiece con la palabra clave `var`.

INTÉNTALO

```
var counter,
    pi = 3.14,
    flag = true,                    // Boolean variable
    shape = “circle";          //String variable
```

4.2.2 Operadores

POR QUÉ

En algunos valores (como suma, resta, etc.) se deben realizar varias funciones que se llevan a cabo mediante los operadores. Algunos de ellos, como +, también se pueden usar para concatenar las cadenas.

CÓMO

JavaScript cuenta con diferentes tipos de operadores.

- Aritméticos
  - Realizan las funciones aritméticas básicas.

+	Suma
-	Resta
*	Multiplicación
/	División
%	Función de residuo
++ ,--	Aumentar o disminuir una variable (operador unario)

+ también se puede usar como operador de concatenación

```
var a = "Hola," + “mundo"
```

Esto significa que *a* contiene *Hola, mundo.*

- Igualdad
  - Se usa para comparar 2 valores y tipos

=	Operador de asignación `var a = 2` Asigna el valor 2 a la variable a
= =	Operador de igualdad `var a = "2" == 2;` //true Esta acción comprueba los tipos y valores de las dos variables

- Comparación
    - Se usa para comparar 2 valores o para comparar Strings.

< , >	Menor que, mayor que
<= , >=	Menor que o igual a, mayor que o igual a

4.2.3 Comentarios

POR QUÉ

Los comentarios son líneas del código que se pueden leer y que el compilador no ejecuta.
No necesitan tener ningún formato específico; solo se añaden al código para que el programador identifique de forma más fácil la función del bloque de código.

CÓMO

Los comentarios de una sola línea deben escribirse con //
Los comentarios de varias líneas deben escribirse entre /* y */

INTÉNTALO

```
var a = 2 ;                    //Esto es un comentario. Debe tener una sola línea.
var b = "Hello" ;          /*Esto es un comentario de más de una línea.
Puede separarse en varias líneas.*/
```

4.3 Funciones, objetos y eventos

4.3.1 Funciones

POR QUÉ

Una función es un módulo de código independiente que lleva a cabo una acción determinada. Esta opción mejora la reutilización del código, especialmente si tienes que realizar una acción determinada varias veces, como hacer una suma o pedirle al usuario que introduzca datos.

CÓMO

La definición de una función consta de una *cabecera* de función y de un conjunto de sentencias compuestas que se conoce como *cuerpo*.
La cabecera de la función empieza con la function *de palabra reservada (palabra clave).*
También tiene un nombre único que describe el objetivo de la función. El cuerpo está formado por sentencias que se encargan de llevar a cabo las acciones determinadas por la función.
Las funciones de JavaScript son objetos, por lo que las variables que hagan referencia a ellos se pueden considerar referencias de objetos.
Las funciones también pueden contener parámetros, que son las variables de una cabecera de

función que, a la vez, puede contener valores. Al llamar a una función, los *argumentos* pueden pasarse a la función, lo que significa que los valores pueden pasarse a la función como datos de entrada. Luego, esto puede usarse en la función. Es importante resaltar que los parámetros de función se pasan mediante el método de valor por llamada. Es decir, solo el valor (no la referencia de objeto) se pasa a la función como datos de entrada.

```
function functionName(var1, var2) {
              /* aquí va el código específico de la función
              …
              */
              Return returnVal; // valor de resultado opcional
}
```

INTÉNTALO
Escribe una función para aceptar dos cadenas del usuario y devolverlas como una cadena concatenada.

```
function returnConcatString(var1 var2) {
              var result = "";                          //iniciar
una cadena en blanco de salida
              for (var i = 0; i < arguments.length; i++) {
                            result = result + " " + arguments[i];
              }
      return result;
}

var b = returnConcatString("Hola,", "mundo");
```

b contiene *Hola, mundo.*

4.3.2 Objetos

POR QUÉ
Un objeto es un conjunto de propiedades. Si el valor de una propiedad es una función, se conoce como método.
Un objeto consiste básicamente en dos elementos: una propiedad y un método.
Para hacerlo más fácil, todos los objetos del mismo tipo tienen las mismas propiedades, pero los valores de propiedad pueden ser distintos.

CÓMO
Un objeto se crea definiendo las propiedades y los métodos que tiene asociados. Para acceder a las propiedades y métodos, usa el operador de punto (.)
Accede a las propiedades de un objeto de la siguiente forma:

```
objectName.propertyName ;
        objectName[“propertyName"] ;
```

Accede a los métodos de una propiedad de la siguiente forma:

```
objectName.methodName() ;
```

Declara un objeto vacío mediante un constructor de la siguiente forma:

```
var car = new Object () ;
```

Una vez hayas creado el objeto, se pueden inicializar las propiedades.

```
car.type = "Sedan";
car.color = “red";
```

INTÉNTALO

Crea un objeto para un tipo de coche con varias propiedades, como el tipo, el color y el modelo.

```
var car = {type: 'Sedan’,
                          color: 'red’,
                          Model: 'Fiat};
```

4.3.3 Eventos

POR QUÉ

Un evento es una acción de usuario o una notificación de que se ha producido alguna acción. El código JavaScript interactúa con las páginas HTML mediante los eventos. Por ejemplo, un evento puede ser hacer clic en un botón o mover el ratón por encima de un texto.

CÓMO

Un gestor de eventos es una secuencia de comandos a la que se llama implícitamente cuando se produce un evento. Todos los elementos tienen un atributo de evento asociado, aunque varios elementos pueden compartir atributos de evento.
Es decir, un evento como `onclick` tiene las mismas acciones en un enlace (etiqueta de enlace) que en un cuadro de texto.

A continuación, se muestran unos cuantos eventos con sus atributos de etiqueta:

EVENTO	ATRIBUTO DE ETIQUETA
blur	onblur
click	onclick
dblclick	ondblclick

focus	onfocus
keydown	onkeydown
keypress	onkeypress
keyup	onkeypress
load	onload
mousedown	onmousedown
mousemove	onmousemove
mouseover	onmouseover
reset	onreset
select	onselect
submit	onsubmit
unload	onunload

INTÉNTALO

Crea un mensaje de alerta cuando se haya cargado el cuerpo del documento.

```
<html>
<head>
              <title> load.html </title>
              <script type = “text/javascript"  src = “load.js">
              </script>
</head>
<body onload = load_greeting();">
              <p> La página se ha cargado. </p>
</body>
</html>

//load.js
function load_greeting() {
              alert("Este es un mensaje emergente de salutación:
¡Hola! Te doy la bienvenida a mi página.");
}
```

4.4 Cadenas, cifras, funciones matemáticas y fechas

4.4.1 Cadenas

POR QUÉ

Las cadenas de JavaScript no se almacenan ni se tratan como matrices de caracteres, sino que son valores escalares unitarios.

CÓMO

Las cadenas pueden tener varias operaciones o funciones para llevar a cabo.
Los métodos de cadena pueden usarse con valores primitivos de cadena como si los valores fueran objetos.
Por ejemplo, el número de caracteres de una cadena se almacena en la propiedad `length` (longitud).

En la tabla de abajo se muestran algunos de los métodos de cadena más conocidos.

Método	Parámetro	Resultado
length	Ninguno	Devuelve como resultado el número de caracteres de un objeto String
charAt	Un número	Devuelve como resultado el carácter de un objeto String en una posición concreta
indexOf	Cadena de un carácter	Devuelve como resultado la posición del parámetro de un objeto String
substring	Dos cifras	Devuelve como resultado la cadena secundaria del objeto String desde la posición del primer parámetro hasta el segundo
toLowerCase	Ninguno	Convierte las mayúsculas en minúsculas
toUpperCase	Ninguno	Convierte las minúsculas en mayúsculas

Además de estos métodos, también se usan unas secuencias de caracteres conocidas como caracteres de escape.

\n	Línea nueva
\t	Tabulación (8 espacios)

\r	Retorno de carro

INTÉNTALO

```
var str = “Hello World" ;

str.charAt(2);                              // el resultado es l
str.indexOf('o’);                 // el resultado es 4
str.substring(2,4);                 // el resultado es llo
str.toLowerCase();                // el resultado es hello world
```

4.4.2 Cifras

POR QUÉ

El objeto Numbers incluye un conjunto de propiedades útiles con valores constantes.

CÓMO

La tabla siguiente muestra las propiedades de las cifras.

Propiedad	Significado
MAX_VALUE	La cifra más grande que se puede representar
MIN_VALUE	La cifra más pequeña que se puede representar
NaN	No es una cifra
POSITIVE_INFINITY	Un valor especial para representar el infinito
NEGATIVE_INFINITY	Un valor especial para representar el infinito negativo
PI	El valor de pi (3,14)

4.4.3 Función Math

POR QUÉ

El objeto Math tiene métodos de funciones trigonométricas, como seno y coseno. También tiene métodos para otras funciones matemáticas habituales.

CÓMO

La función Math tiene funciones trigonométricas como sin (sine) y cos (cosine), así como otras funciones matemáticas como `floor` (que trunca una cifra), `round` (que redondea la cifra al valor entero más próximo) o `max` (que devuelve como resultado la mayor cifra de entre dos), entre otras.

El objeto Math se usa para hacer referencia a todas las funciones matemáticas.

INTÉNTALO

```
Math.floor(4.2343);                    //El resultado es 4
```

4.4.4 Función Date

POR QUÉ

La función Date se usa para representar la fecha y hora.

CÓMO

El objeto Date se usa para hacer referencia al objeto Date.
Se puede crear con la siguiente sentencia:

```
var today = new Date() ;
```

A continuación, se muestran algunos de los métodos más usados para el objeto Date.

Método	Devuelve
toLocaleString	Una cadena de la información de Fecha
getDate	Día del mes
getMonth	Mes del año (de 0 a 11)
getDay	Día de la semana (de 0 a 6)
getFullYear	Año
getTime	La cantidad de milisegundos desde el 01/01/1970
getHours	La cifra que representa las horas (de 0 a 23)
getMinutes	La cifra que representa los minutos (de 0 a 59)
getSeconds	La cifra que representa los segundos (de 0 a 59)

4.5 Matrices

POR QUÉ

Una matriz es un conjunto de valores con el mismo nombre de referencia. Cada elemento de matriz es accesible desde su índice, cuya longitud va desde 0 hasta length-1.
Es importante resaltar que en JavaScript, a diferencia de otros lenguajes de programación, las matrices pueden ser de tipo mixto, lo que significa que no es necesario que todos los elementos de una matriz sean del mismo tipo.

CÓMO

Los objetos Array, a diferencia de lo que pasa con la mayoría de objetos de JavaScript, se pueden crear de dos formas.
La primera y la más habitual es usando un operador *nuevo*, como al crear la mayoría de objetos.

```
var list = new Array (1, 2, "three", "fo ur") ;
```

La segunda, usando un valor de matriz literal.

```
var list_2 = [1, 2, "three", "four"];
```

El índice más bajo de cualquier valor de matriz siempre es cero.
Se puede acceder a los elementos Array usando el valor numérico del índice como secuencia de comandos secundaria para el nombre de matriz.
La longitud de una matriz es la cantidad de elementos que contiene. Así, pues, el último índice de una matriz siempre es length-1.

Las matrices tienen varios métodos asociados. A continuación, se muestran algunos de los más comunes.

Método	**Finalidad**
join	Convierte todos los elementos de un objeto Array en una cadena y los concatena
reverse	Invierte el orden de los elementos de un objeto Array
sort	Convierte todos los elementos de un objeto Array en una cadena y los ordena alfabéticamente
concat	Añade los parámetros actuales del método concat al final del objeto Array
slice	Devuelve como resultado parte de un objeto Array que se haya definido como parámetros
pop	Quita un elemento del otro extremo de un objeto Array
push	Añade un elemento al otro extremo de un objeto Array

INTÉNTALO
Crea un objeto de matriz y muestra los valores. Luego, añade más elementos a la matriz y muestra los valores nuevos.

```
//array_example.js
var list = [2, 4, 6];
```

```
for (var i = 0; i < list.length; i++) {
              document.write(list[i] + "  ");
}
document.write("<br />");

var new_list = list.concat(8, 10);

for (var j = 0; j < new_list.length; j++) {
              document.write(new_list[j] + "  ");
}
```

El resultado sería
2 4 6
2 4 6 8 10

4.6 Booleans, condiciones y bucles

4.6.1 Booleanos

POR QUÉ

Un Boolean es un tipo de datos de JavaScript que solo puede tener dos valores:
True y False.

CÓMO

Los objetos Boolean se declaran como todos los otros objetos de JavaScript.

```
var bool_t = new Boolean() ;
        bool_t = true;
```

4.6.2 Condiciones

POR QUÉ

Las sentencias condicionales son importantes porque determinan el flujo de ejecución de las sentencias. Siempre usan parámetros que devuelven un valor boolean como resultado. Si el valor boolean es cierto, se ejecuta el bloque de sentencias inmediatamente posteriores a la condición.
Dependen de los operadores lógicos.

CÓMO

Las expresiones en que se puede basar el control del flujo de sentencias incluyen valores primitivos y expresiones relacionales y compuestas. El resultado de una expresión de control es True o False.
JavaScript también tiene los operadores AND, OR y NOT para las operaciones Boolean. Son: && (AND), || (OR) y ! (NOT).

Existen dos tipos de sentencias condicionales:

1. Sentencias de selección

   Dependen de la construcción if-then e if-then-else.

   Esta es la sintaxis:

```
if (condición) {
                /* si el resultado de la condición es
cierto,
   ejecuta las siguientes sentencias
                …
                */  }
                else {
                /* si el resultado de la condición es
falso,
   evalúa estas sentencias
                …
                */  }
```

2. La sentencia switch

   Se usa cuando se deben comprobar varias condiciones y solo una puede coincidir.

Esta es la sintaxis:

```
switch (expresión) {
                                case value_1:
 //sentencias
                                case value_2:
 //sentencias
   …
   case value_n:                          //sentencias
   [default:                            //sentencias]
}
```

Se evalúa la expresión de la construcción switch y el valor se compara con los valores de los casos de la construcción. Si uno de ellos coincide, el control se transfiere a las sentencias siguientes al valor del caso. Luego, se sigue ejecutando lo que queda de la construcción.
En la mayoría de casos, solo se debe ejecutar una sentencia de caso y, a continuación, el control debe transferir las sentencias externas a la construcción de switch.
En este caso, se debe usar una sentencia *break*.
La sentencia *break* transfiere el control al exterior de la sentencia compuesta en que aparece.

INTÉNTALO

Escribe una construcción if-else sencilla para comparar dos cifras y que muestre el número mayor.

```
//if_then.js
```

```
var a = 5;
var b = 8;

if (a > b)
   document.write("a es mayor que b <br/> ");
else
   document.write("a es menor que b <br/> ");
```

4.6.3 Bucles

POR QUÉ

Los bucles se usan mayormente para realizar la misma tarea en varios elementos de una matriz o lista. Lo que hacen básicamente es hacer bucles en un bloque de código varias veces.

CÓMO

Los bucles suelen tener una expresión de control seguida de un bloque de código entre paréntesis. Existen dos tipos de bucles:

1. Bucles While

   Los bucles While ejecutan una serie de líneas de código sin parar hasta que la expresión de control ya no se muestra como cierta. Esta es la sintaxis:

```
while (sentencia condicional)
        {
                        /* conjunto de sentencias
                        …
                        */
        }
```

2. Bucles For

Los bucles For constan de un valor inicial, uno condicional y un paso de incremento. El valor inicial de la variable se incrementa cada vez que se ejecuta el bloque de sentencias. Cuando el valor condicional deja de ser cierto, el bucle se detiene. Esta es la sintaxis:

```
for (expresión inicial; expresión de con trol; expresión de
incremento)
{
                        /* conjunto de sentencias
                        …
                        */
        }
```

Los dos tipos de bucles pueden detenerse cuando sea necesario con la sentencia break.

INTÉNTALO

Escribe un código JavaScript para saber el producto de las cifras de 1 a 50.

```
//loops_while.js

var i = 1;
var sum = 0;

while ( i <= 50) {
sum = sum * i ;
}

document.write("El producto es" + sum);

//loops_for.js
var sum = 0;
for (var i = 0; i < 51; i++)
sum = sum * i;

document.write("El producto es" + sum);
```

4.7 Regular Expressions (Expresiones regulares, RegEx)

POR QUÉ

JavaScript tiene funciones muy potentes para hacer coincidir patrones según regular expressions. Aunque se puede usar la función de `búsqueda` de objetos String, RegEx tiene una funcionalidad mucho más amplia.

CÓMO

Los caracteres "normales" son aquellos que no consideramos metacaracteres, es decir, que no tienen ningún significado especial según la situación. Estos son algunos de los caracteres especiales o metacaracteres:

`\ | ( ) [ ] { } ^ $ * + ? .`

Los metacaracteres se hacen coincidir en las expresiones regulares usando una barra inversa justo delante de ellos (\).

El punto (.) coincide con cualquier carácter, excepto con la línea nueva. Como se indica más arriba, para hacer coincidir un punto de una cadena, debe ir precedido por una barra inversa.
Ejemplo:
`/snow./` coincidirá con "snowy", "snowed", "snows" etc.

Si la expresión regular empieza con un carácter circunflejo (^), niega o invierte el conjunto especificado.
Ejemplo:

[^abc] coincidirá con todos los caracteres excepto "a", "b" y "c"

El guion (-) indica un intervalo.

La siguiente tabla te muestra más detalles sobre las clases predefinidas.

Nombre	Patrón equivalente	Coincide con
`\d`	`[0-9]`	Un dígito
`\D`	`[^0-9]`	No es un dígito
`\w`	`[A-Za-z_0-9]`	Un carácter alfanumérico
`\W`	`[^A-Za-z_0-9]`	Un carácter que no es alfanumérico
`\s`	`[ \r\t\n\f]`	Un carácter de espacio
`\S`	`[^ \r\t\n\f]`	Un carácter que no es un espacio

INTÉNTALO

[\d\.\d\d] //Coincide con un dígito seguido de un punto y 2 dígitos
[\D\d\D] //Coincide con un único dígito
[\d+\.\d*] //Coincide con un dígito o más seguidos de un punto y 0 dígitos o más
[\w\w\w] //Coincide con los 3 caracteres adyacentes a la palabra