**APUNTES**

**JAVASCRIPT**

**ÍNDICE**

[**1. Introducción 3**](#_cx0othhz5nvv)

[**1.1. Implementar JavaScript en documento XHTML 3**](#_pmrs5u91wxhv)

[**1.1.1. Implementar JavaScript internamente 3**](#_b1mdzjphfhke)

[**1.1.2. Implementar JavaScript externamente 3**](#_dshrs3kjlpur)

[**1.2. Etiqueta NoScript 4**](#_57zvemugor9s)

[**1.3. Sintaxis 5**](#_m5oxg7eliz21)

[**1.4. Variables 6**](#_puo21810bbvr)

[**1.4.1. Tipos de variables 6**](#_k2jx07b3sgtf)

[**1.5. Operadores 8**](#_691d65b4fjaz)

[**1.5.1 Incremento y Decremento 8**](#_uk0mb3fmkgfq)

[**1.5.2 AND y OR 8**](#_r9qyg429hyrx)

[**1.5.2 Relacionales 10**](#_qsygglyvqd7b)

[**1.6. Flujos 11**](#_v44ytda5s3en)

[**1.6.1 Condición IF…ELSE 11**](#_ex2uj43gg7cj)

[**1.6.2 Bucle FOR 12**](#_ddc8vpw1l7eu)

[**1.6.3 Bucle FOR…IN 12**](#_98hpuslaomfc)

[**1.7. Funciones útiles 13**](#_raxezto11qb0)

[**2. Programación avanzada 15**](#_68qcdvsukur)

[**2.1. Funciones 15**](#_qn1nnkb0260x)

[**2.2. Argumentos y Valores de retorno 17**](#_1kogq9uzb341)

[**3. DOM 18**](#_70dkkreemn1s)

[**3.1. Tipos de Nodos 18**](#_qvfl2mcp4mwq)

[**3.2. Acceso directo a los nodos 19**](#_5shjwdtdp6i)

[**3.3. Creación y Eliminación de nodos 20**](#_8ygb4o650vmy)

[**3.4. Acceso directo a atributos de XHTML 21**](#_ud2ru83cnwgo)

# 1. Introducción

## 1.1. Implementar JavaScript en documento XHTML

Para implementar JavaScript en un documento XHTML, se debe usar la etiqueta “<script>”, preferiblemente dentro de la cabecera del documento, aunque se podría usar en cualquier parte del mismo.

### 1.1.1. Implementar JavaScript internamente

Si se quiere implementar internamente en el mismo documento, dentro de la etiqueta <script>, se pondrá el siguiente código:

<html>

<head>

<script **type="text/javascript"**>

alert("Un mensaje de prueba");

</script>

</head>

<body>

<p>

Un p&aacute;rrafo aleatorio.

</p>

</body>

</html>

### 1.1.2. Implementar JavaScript externamente

Si se quiere implementar externamente desde un documento javascript .js externo, en el ejemplo anterior deberá integrar el siguiente añadido:

<head>

<script type="text/javascript" src="C:\Users\Equipo\Desktop\2º DAW\JAVASCRIPT\codigo.js">

alert("Un mensaje de prueba");

</script>

</head>

## 1.2. Etiqueta NoScript

La etiqueta “<noscript>” se usa en páginas en las que se necesita el uso de JavaScript para disfrutar al máximo la experiencia de dicha página. Si el usuario decide desactivarlo, al entrar en la página, le saltará un mensaje con esta etiqueta aplicada, para dar a entender de que debería activar JavaScript para esta página.

Para añadirla, en el <body>, añadimos la sentencia:

<html>

<head>

<script type="text/javascript"></script>

</head>

<body>

**<noscript>**

<p>Bienvenido a Mi Sitio</p>

<p>La página que estás viendo requiere para su funcionamiento el uso de JavaScript.

Si lo has deshabilitado intencionadamente, por favor vuelve a activarlo.</p>

**</noscript>**

</body>

</html>

Cabe destacar que dentro de la etiqueta <noscript> se pueden añadir elementos propios del XHTML con libertad.

## 1.3. Sintaxis

La sintaxis de JavaScript es parecida a lenguajes de programación como Java o C:

* **No se tienen en cuenta los espacios en blanco y nuevas líneas**
* **Se distinguen las mayúsculas y minúsculas**
* **No se define el tipo de las variables**
* **No es necesario terminar cada sentencia con ;**
* **Se pueden incluir comentarios**

Este último tiene dos maneras de comentar:

1. **//Comentario de línea**
2. **/\* Comentario de varias líneas\*/**

## 

## 

## 

## 

## 

## 

## **1.4. Variables**

Como norma general, las variables se definirán con la expresión **“var”. SÓLO** se definirá una vez por variable, ya que se podrá usar nuevamente poniendo el nombre atribuido a dicha variable.

Puede predefinirse: **“ var valor1; ”**

Puede inicializarse: **“ var valor1 = 1; ”**

Pueden usarse en una nueva variable: **“ var resultado = valor1 + valor2; ”**

* **Sólo puede estar formado por letras, números y los símbolos $ (dólar) y \_ (guión bajo).**
* **El primer carácter no puede ser un número.**

### 1.4.1. Tipos de variables

Todas las variables se definen con “var”, pero para identificar la variable que introducimos, dependiendo de lo que se escriba, se interpretará de una forma u otra:

* **Numéricas:** Al poner un número en la variable.
* **Cadenas de texto:** Al poner una cadena de texto en la variable. Puede poner el mensaje entre comillas simples o dobles. Hay algunas cadenas especiales según la situación:

| **Si se quiere incluir...** | **Se debe incluir...** |
| --- | --- |
| Una nueva línea | \n |
| Un tabulador | \t |
| Una comilla simple | \' |
| Una comilla doble | \" |
| Una barra inclinada | \\ |

* **Arrays:** Al poner diversas variables entre [ y ]. Puede ponerse tantos valores predefinidos como nombres de variables. Si queremos acceder a algún dato, haremos la siguiente expresión:

<!DOCTYPE html>

<html>

<head>

<script type="text/javascript">

var semana = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo"];

var diaAleatorio = semana[2];

alert("El día a mostrar es: " + diaAleatorio);

</script>

</head>

<body>

<noscript>

ES NECESARIO USAR JAVASCRIPT.

</noscript>

</body>

</html>

* **Booleano:** Al usar las expresiones “true” y “false”.

## 

## 1.5. Operadores

### 1.5.1 Incremento y Decremento

Para ello, se implementa ++ y –.

Si queremos incrementar o decrementar un valor antes de realizar alguna operación, se pondrá como prefijo de la variable, es decir, delante. En caso contrario, si lo queremos después, se colocará como sufijo.

<!DOCTYPE html>

<html>

<head>

<script type="text/javascript">

var numero1 = 5;

var numero2 = 5;

var resultado1 = ++numero1 + numero2;

alert(resultado1); // resultado1 = 11 ; numero1 = 6

var resultado2 = numero1++ + numero2;

alert(resultado2); // resultado2 = 10 ; numero1 = 6

</script>

</head>

<body>

<noscript> NECESITA JAVASCRIPT.</noscript>

</body>

</html>

### 1.5.2 AND y OR

Centrándonos en la parte booleana, con AND:

| **vvariable1** | v**variable2** | **variable1 && variable2** |
| --- | --- | --- |
| ttrue | ttrue | true |
| ttrue | ffalse | false |
| ffalse | ttrue | false |
| ffalse | ffalse | false |

Centrándonos en la parte booleana, con OR:

| **vvariable1** | **vvariable2** | **variable1 || variable2** |
| --- | --- | --- |
| ttrue | ttrue | true |
| ttrue | ffalse | true |
| ffalse | ttrue | true |
| ffalse | ffalse | false |

### 1.5.2 Relacionales

Centrándonos más en las cadenas de texto, seguimos una comparativa no intuitiva. Dará true si ambas cadenas son iguales y false si son distintas.

Por otra parte, al poner > o <, se basará en el orden alfabético inverso, siendo la z la mayor, y la a la menor.

**Ejemplo:**

**zoo** es **mayor que avión** porque z es mayor que a.

## 

## 

## 

## 

## 

## **1.6. Flujos**

Se trata de estructuras tales como **if**, **for**, **while**, etc… .

### 1.6.1 Condición IF…ELSE

La expresión a utilizar sería:

<html>

<head>

<script type="text/javascript">

var condicion = true;

var mensaje1 = "Bienvenido!";

var mensaje2 = "Adiós!"

if(condicion) {

alert(mensaje1);

} else {

alert(mensaje2);

}

</script>

</head>

<body>

</body>

</html>

Obviamente, en dicho flujo se podrá combinar con operadores para exprimir al máximo el uso del IF…ELSE.

### 

### 

### 

### 

### 1.6.2 Bucle FOR

Su estructura es idéntica a Java:

<html>

<head>

<script type="text/javascript">

var mensaje = "Hola, estoy dentro de un bucle";

for(var i = 0; i < 5; i++) {

alert(mensaje);

}

</script>

</head>

</html>

### 1.6.3 Bucle FOR…IN

Su estructura es parecida al FOR. Se expresará de la siguiente manera:

<html>

<head>

<script type="text/javascript">

var dias = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo"];

for(i in dias) {

alert("Hoy es: " + dias[i]);

}

</script>

</head>

<body></body>

</html>

La expresión “in” se refiere a que la variable “i” va a iterar desde el inicio a fin el array anteriormente definido, mostrando luego cada valor previamente añadido en dicho array. Este tipo de bucle se suele usar en estructuras más avanzadas al resultar muy útiles y cómodas para un código más limpio y eficiente.

## 1.7. Funciones útiles

* **var.concat();:** Para concatenar dos variables
* **var.toUpperCase();:** Para convertir toda la cadena de texto a mayúsculas
* **var.toLowerCase();:** Para convertir toda la cadena de texto a minúsculas
* **var.charAt(posicion);:** Para mostrar el carácter situado en la posición indicada
* **var.indexOf(‘’);:** Para devolver la posición en la que se encuentra el carácter indicado. Si se repite el carácter más de una vez, mostrará el más cercano a partir de la izquierda. Si no existe el carácter en la cadena de texto, devuelve -1
* **var.lastIndexOf(‘’);:** Devuelve la última posición en la que se encuentra el carácter indicado, en el caso de que se repita dicho carácter. Si no existe el carácter en la cadena de texto, devuelve -1
* **var.substring(inicio, final);:** Para sacar una porción de una cadena de texto entre dos posiciones, la del inicio y final. Si sólo se indica el inicio, sacará la porción desde la posición indicada hasta el final de la cadena. Sin embargo, si se indica la posición final, mostrará la porción entre esas dos posiciones, pero la posición inicial si la muestra, mientras que del final mostrará la anterior a la posición indicada
* **var.split(separador);:** Convierte una cadena de texto a array a partir de un separador, es decir, que según el separador indicado, dividirá elementos de una forma u otra para el array. Por ejemplo, si separamos por cada espacio en blanco, la cadena “Hola Mundo” se transformará en un array de “Hola”, “Mundo”, al indicar que se separen por cada espacio en blanco
* **var.join(separador);:** Funciona al contrario que un split. A partir de un array, todos sus elementos se transforman en una única cadena de texto dividida por el separador indicado
* **var.pop();:** Elimina el último elemento de un array y lo muestra en pantalla, disminuyendo en 1 la longitud del mismo
* **var.push();:** Añade un nuevo elemento al final de un array, aumentando en 1 la longitud del mismo
* **var.shift();:** Lo mismo que var.pop() pero en vez de eliminar el último elemento, elimina el primero
* **var.unshift();:** Lo mismo que var.push() pero en vez de añadir el último elemento, lo añade en el primero
* **var.reverse();:** Pone el array en reversa, es decir, si era “1, 2, 3”, ahora será “3, 2, 1”
* **isNaN(cálculo):** Se usa para que, en caso de salir un número no definido, avisarlo por pantalla. Se usa en un IF, por ejemplo
* **var.toFixed(número);:** Modifica un número, mostrando tantos decimales como se defina dentro del paréntesis de la función

## 

## 

## 

## 

## 

## 

## 

# 

# **2. Programación avanzada**

## **2.1. Funciones**

Para definir una función en Javascript, se escribirá la siguiente expresión:

<html>

<head>

<script type="text/javascript">

function nuevaFuncion() {

}

</script>

</head>

<body></body>

</html>

Con esta función, podremos llamarla cuantas veces queramos.

Para poder utilizarla, la llamaremos con su nombre y dos paréntesis. En el siguiente caso:

<html>

<head>

<script type="text/javascript">

function resultadoSuma() {

var resultado = numero1 + numero2;

alert("El resultado es: " + resultado);

}

var numero1 = 3;

var numero2 = 4;

var resultado;

resultadoSuma();

</script>

</head>

<body></body>

</html>

De esta forma, se haría una función simple sin pasarle argumentos a la función directamente.

### 

### 

### 

### 

### 

### 

### 

### 

## 

## **2.2. Argumentos y Valores de retorno**

Al pasar argumentos, nos referimos a pasarle variables entre los paréntesis de la definición de la función. De esta forma, optimizamos aún más el código y lo dejamos más limpio y eficiente, como el siguiente ejemplo:

<html>

<head>

<script type="text/javascript">

function resultadoSuma(primero, segundo) {

var resultado = primero + segundo;

alert("El resultado es: " + resultado);

}

var numero1 = 3;

var numero2 = 4;

resultadoSuma(numero1, numero2);

</script>

</head>

<body></body>

</html>

Como se puede apreciar, en los paréntesis de la función se indica que entrarán en sus instrucciones dos variables las cuales se podrán usar cuando se indique dentro de la misma, como en este caso de suma. No importa si los nombres en los paréntesis son distintos. Lo que importa es pasar las variables que quieres usar en la función.

Si queremos usar el resultado, al final de la función se escribirá la sentencia “return”, la cual devuelve el valor fuera de la función para ser usada. Simplemente crearemos una variable y le asignamos la llamada de la función y listo.

# **3. DOM**

## **3.1. Tipos de Nodos**

* **Document:** Nodo raíz del que derivan el resto de nodos
* **Element:** Representa cada etiqueta creada. En ella pueden contener atributos
* **Attr:** Define el atributo de un elemento por cada atributo=valor
* **Text:** Contiene el texto encerrado en una etiqueta
* **Comment:** Representa los comentarios de una página XHTML

Hay otros tipos tales como: DocumentType, CDataSection, DocumentFragment, Entity, EntityReference, ProcessingInstruction y Notation.

## **3.2. Acceso directo a los nodos**

* **getElementsByTagName();:** Muestra todos los elementos y su contenido con el valor indicado entre paréntesis. Si pones “document.getElementsByTagName(“p”)”, es básicamente pedir que te muestre el contenido de cada elemento que esté entre <p> y </p>. Cabe recalcar que creando una variable, crearás un array de nodos de esa etiqueta.
* **getElementsByName();:** Muestra sólo las etiquetas cuyo atributo “name” tenga el valor indicado entre paréntesis. Por ejemplo, si en dicho paréntesis escribimos “especial”, debe buscar etiquetas con el siguiente contenido: <etiqueta name=”especial”> … </etiqueta>. Es como el anterior, pero a partir del valor del atributo “name”.
* **getElementById();:** A diferencia del anterior, el Id es único en cada etiqueta, por lo que al escribir dicha función, nos mostrará únicamente el contenido de aquella etiqueta cuyo atributo “Id” tenga el valor indicado en el paréntesis. Por ejemplo: document.getElementById(“cabecera”); buscará la etiqueta con dicho valor, como por ejemplo: <div id=”cabecera”> … </div>.

## **3.3. Creación y Eliminación de nodos**

Para ello, los pasos a seguir son:

* **Crear nodo “Element”:** Para ello, se crea con “.createElement();”
* **Crear nodo “Text”:** En él está el contenido del elemento. Para ello, se crea con “.createNodeText();”
* **Añadir el texto en el elemento:** Para ello, una vez creado ambos pasos, se usa la expresión “elemento.appendChild(contenido);”. Con esto, decimos que dentro de dicho elemento, ponemos a dicho contenido como hijo del elemento
* **Añadir el elemento como hijo de la página:** Por último, toca añadirlo a la página, usando la expresión “document.body.appendChild(elemento);”

Ahora, veremos cómo eliminar los elementos creados. Para ello, haremos uso de la siguiente expresión:

***elemento.parentNode.removeChild(elemento);***

Para poder usar esa expresión, guardamos en una variable dicho nodo, usando la función .getElementById();, para poder encontrarlo más fácil y además guardar sólo el nodo que queremos eliminar.

## **3.4. Acceso directo a atributos de XHTML**

Para acceder a los atributos de un elemento en XHTML, primero, como siempre, guardaremos en una variable el elemento usando la función “.getElementById(“elemento”);”. Una vez guardado en la variable, si queremos mostrar el contenido del atributo de dicho elemento, lo mostraremos con alert(), donde dentro del paréntesis pondremos la variable seguido de “.attrname”, para coger el atributo. Por ejemplo:

<html>

<head>

</head>

<body>

<a id="enlace" href="youtube.com">

¡Suscribete!

</a>

</body>

<script type="text/javascript">

var elemento = document.getElementById("enlace");

alert(elemento.id);

</script>

</html>

Se pone el script al final ya que al ponerlo primero, no deja que cargue todo el DOM, por lo que mostraría un valor nulo. Hay que asegurarse de dejar que cargue todo el código para luego ejecutar el código javascript.

En cambio, si queremos mostrar el atributo de CSS, simplemente pondremos al lado de esa variable donde se guardó el elemento “.style.estilousado”, por ejemplo:

<html>

<head>

</head>

<body>

<div id="divBordeado" style="border:solid; border-color: red; margin: 100px;">

Div con borde

</div>

</body>

<script type="text/javascript">

var elemento = document.getElementById("divBordeado");

alert(elemento.style.margin);

</script>

</html>

En este ejemplo, ponemos un bordeado de color rojo y con un margen general de 100 px a un div. Si queremos que nos muestre el valor dado al margen de dicho div, ponemos la sentencia mostrada en la captura de pantalla.

Cabe recalcar que si el atributo es compuesto, es decir, que tiene palabras con guiones, a la hora de mostrarlo en pantalla se debe poner de la siguiente manera:

font-weight -> fontWeight

margin-top -> marginTop

…

Por último, el único atributo que no se escribe igual a la hora de llamarlo es “class”. Se debe escribir “className” en vez de “class”, ya que al poner este último indicará que es undefined, es decir, indefinido, mientras que poniendo “className”, nos mostrará el valor asignado.

<html>

<head>

</head>

<body>

<div id="divBordeado" class="divNormalBordeado" style="border:solid; border-color: red; margin-top: 150px;">

Div con borde

</div>

</body>

<script type="text/javascript">

var elemento = document.getElementById("divBordeado");

alert(elemento.style.marginTop);

alert(elemento.className);

</script>

</html>