1. LA ETIQUETA <SCRIPT>

El código **JavaScript** puede escribirse junto al resto del código de la página web, mezclado con el propio código HTML, o en un archivo independiente cuya extensión es **.js**.

Si decides escribir el código JavaScript en el documento HTML, puedes hacerlo en la sección de cabecera o en el cuerpo de la página, dependiendo de lo que desees hacer.

Así pues, si incorporas el código JavaScript en la sección de cabecera (**head**), este código se interpreta antes que cualquier otro elemento del cuerpo (**body**) de la página.

Fíjate en el siguiente código de una página web:

```
<!DOCTYPE HTML>
<html>
<head>
<style>
.imagen{
   border-style: none;
}
body{
   background-color: #ffffff;
}
</style>
<script type="text/javascript">
<!--
function swapImage(i, j) {
   document.images[i].src = "image" + j + ".gif";
}
// -->
</script>
<title>Otro ejemplo</title>
</head>
<body>
<a href="document.html" onmouseover="swapImage(0,2);"
                        onmouseout="swapImage(0,1);">
<img src="image1.gif" width="172" height="58"
 class="imagen">
</a>
</body>
</html>
```

El código JavaScript se introduce entre las etiquetas **<script>** y **</script>**. En este caso, observa cómo se escribe el código en la sección de cabecera.

Además, como el guión o secuencia de comandos puede ser escrito en varios lenguajes, debemos indicar que utilizamos JavaScript. Para ello aparece el atributo **type**: `type="text/javascript"`

Por otra parte, debemos ocultar el código JavaScript a aquellos navegadores que no sepan interpretarlo. Para ello se utilizan los comentarios.

En HTML los caracteres **<!--** se entienden como inicio de comentario y **-->** como fin de comentario. Es decir, todo lo que está incluido entre ambos es un comentario.

Por su parte, JavaScript utiliza **//** para indicar un comentario de una línea.

En JavaScript podemos escribir comentarios de una línea o que ocupen varias líneas:

Para los comentarios de una línea, se utiliza **//**

Para los comentarios de varias líneas, se utiliza **/*** al principio y ***/** al final del comentario.

Veamos qué ocurre con un navegador que no reconoce el código JavaScript:

1. Como no reconoce la etiqueta **<script>**, simplemente la ignora.

2. Encuentra el símbolo de inicio de comentario HTML (**<!--**), así que no interpreta el resto, hasta encontrar el símbolo -->.

3. Como no reconoce la etiqueta **</script>**, pasa a la línea siguiente.

Ahora veamos lo que ocurre con un navegador que sí que reconoce JavaScript:

1. Reconoce la etiqueta **<script>**, así que empieza a interpretar el código siguiente excepto los comentarios de HTML (**<!--**).

2. Interpreta el código siguiente hasta que llega a la línea que muestra un comentario de JavaScript (**//**), ignorando dicha línea.

3. Reconoce la etiqueta **</script>**, de forma que sabe que lo siguiente ya no es JavaScript.

Como puedes ver, el código funcionará correctamente tanto para uno como para el otro caso. La estructura que debes tener en cuenta es la siguiente:

```
<script>
<!--
 Código JavaScript
//-->
</script>
```

Podrás incorporar tantos guiones como desees en una página web. Lo único que tienes que hacer es repetir esta estructura las veces que haga falta.

Un último apunte por ahora: JavaScript es un lenguaje que diferencia entre minúsculas y mayúsculas, a diferencia del HTML. Ten cuidado con ello.

2. CONTENIDO ALTERNATIVO

Aunque en la actualidad vas a poder ejecutar código JavaScript en cualquier navegador, podemos crear contenido alternativo para aquellos navegadores que no lo saben interpretar o que tienen desactivada esta funcionalidad.

Los pasos a seguir son los siguientes:

1. Incluye la etiqueta **<noscript>** tras la etiqueta **</script>** de fin de guión.

2. Incluye el código HTML que desees para esta sección de la página.

3. Finaliza con **</noscript>**.

En muchos casos, el código HTML que se incluye para estos navegadores es justamente una indicación de que se hace necesario poder ejecutar código de script para disfrutar de la página web. Fíjate:

```
<noscript>
<p>Esta página requiere de JavaScript para su correcta
visualización.</p>
</noscript>
```

Puede que te estés preguntando cómo es posible que estos navegadores reconozcan la etiqueta **<noscript>** y no la etiqueta **<script>**. La respuesta es sencilla: no reconocen ninguna de las dos, por lo que ignoran ambas.

Lo que sí que reconocen es el código situado tras la etiqueta **<noscript>** y, como en este caso no se utilizan comentarios, lo muestran en la página web.

Aquellos navegadores que interpretan JavaScript simplemente ignoran el contenido alternativo.

No siempre desearás crear contenido alternativo. En muchos casos, la mejor solución es ignorar dicho código.

3. VARIABLES

Ahora que ya sabemos cómo insertar código JavaScript en la página web, empezamos a estudiar algunos conceptos fundamentales de programación.

Aun teniendo experiencia en programación, debes conocer las particularidades de JavaScript, por lo que deberías seguir las explicaciones con atención.

En el caso de que no tengas conocimientos de programación, estas lecciones serán de gran ayuda.

Y sin lugar a dudas, uno de los conceptos fundamentales en programación es el de **variable**. Una variable es una ubicación temporal de memoria donde el programador guarda cierto valor que le interesa retener durante la ejecución del *script* o guión.

Estudia el siguiente código:

Aunque no es obligatorio finalizar las instrucciones con un punto y coma en JavaScript, es una costumbre hacerlo.

```
<!DOCTYPE HTML>
<html>
<head>
<title>Título de la página</title>
</head>
<body>
<script type="text/javascript">
<!--
var Variable1 = "Esta es una cadena de texto";
document.write(Variable1);
document.write("<br>");
var Variable2 = 255;
document.write(Variable2 + 5 + "45");
document.write("<br>");
Variable2 = "Otra cadena de texto.";
document.write(Variable2);
//->
</script>
</body>
</html>
```

Para poder utilizar variables, es necesario **declararlas**. Al declararlas, les das el nombre de la variable e incluso se le puede dar un valor inicial, que podrá cambiar posteriormente.

Se utiliza la palabra clave **var** y seguidamente el nombre de la variable. Adicionalmente, se puede indicar un valor inicial al declararla:

```
var Variable1 = "Esta es una cadena de texto";
```

En este caso estamos guardando la cadena *Esta es una cadena de texto* en una variable con el nombre **Variable1**, de forma que puedes utilizar esta cadena siempre que la necesites.

Se utiliza el operador de asignación = para establecer el valor que debe tomar. Como es texto, la cadena debe aparecer entre comillas dobles (").

A este respecto debe indicarse que se puede incluir cadenas de texto tanto entre comillas simples (') como entre comillas dobles ("), pero si eliges iniciar la cadena de una forma, tienes que acabarla de esa misma forma.

Además, si en el interior de la cadena necesitas incluir las comillas, utiliza el otro tipo de comillas para indicar el inicio y fin de la cadena: *'Esta cadena incluye comillas dobles " entre sus caracteres'*.

Es importante tener claro que el valor de la variable puede cambiar a lo largo del *script* (obsérvalo para el caso de **Variable2** en el ejemplo anterior).

4. TIPOS DE DATOS

El concepto de **tipo de datos** es sencillo: determina el conjunto de valores que puede tomar una variable y el conjunto de operaciones en las que puede formar parte.

En JavaScript, el tipo de datos de una variable se establece al asignarle un nombre, por lo que no es posible indicarlo expresamente en la declaración de la variable.

Por ejemplo, pensemos en una variable que almacena texto. Esa variable no debería poder tomar un valor numérico y no debería formar parte en una operación de multiplicación.

Es conveniente dar un valor inicial a las variables. Ten en cuenta que, si no lo haces, se corre el riesgo de utilizar una variable en una operación o expresión y que se produzca un error.

Sin embargo, JavaScript realiza conversiones automáticas de tipo, de forma que una variable puede llegar a tomar valores de distintos tipos a lo largo del *script*. Esto no es una buena técnica, pero en ocasiones es necesario. Fíjate:

```
var Variable2 = 255;
Variable2 = "Otra cadena de texto.";
```

Observa cómo primero hemos asignado un valor numérico (255) y más tarde una cadena de texto. JavaScript realizará la conversión de tipo de forma automática.

En JavaScript podrás tener los siguientes tipos de datos:

- **Números**: cualquier número entero o real (fíjate que no hay un tipo distinto para cada caso).
- **String**: una serie de caracteres encerrados entre comillas simples (') o dobles (").
- **Boolean**: un valor lógico (verdadero o falso).
- **Null**: significa que la variable no tiene valor.

5. OPERADORES

Conocemos cómo declarar variables y qué tipos de datos podemos utilizar. Sin embargo, las variables en sí son bastante inútiles si no forman parte de expresiones más complejas.

Por ejemplo, podremos realizar operaciones matemáticas con dos variables numéricas, conseguir una cadena de texto a partir de otras, etc.

Uno de los elementos que permiten crear expresiones más complejas son los **operadores**. En JavaScript se puede utilizar un gran número de operadores divididos según su carácter:

Operadores matemáticos:

+ Suma
- Diferencia
*** Multiplicación**
/ División
% Módulo
++ Incremento
-- Decremento

El operador **%** devuelve el resto de una división. Es decir, se realiza la división y se guarda el resto obtenido. Por ejemplo, **`5 % 3 = 2`**

Operadores de comparación:

== Igual a
!= Distinto a
< Menor que
<= Menor o igual que
> Mayor que
>= Mayor o igual que

Operadores de asignación: para establecer el valor de las variables.

= Asignación
+= Se asigna la suma o concatenación. Por ejemplo `variable +=5;` es igual a `variable = variable + 5;`
-= Se asigna la resta.
***= Se asigna la multiplicación.**
/= Se asigna la división.
%= Se asigna el módulo.

Operadores con cadenas

+ Concatenación:

Inserta la segunda cadena de texto al final de la primera. Por ejemplo:

```
"Hasta " + "luego"
```

produce la cadena "Hasta luego" (observa que la primera cadena incluye un espacio al final).

Operadores lógicos, operadores para fechas (Date), etc.

El operador **+** es un caso especial, ya que puede producir resultados distintos dependiendo de que los operandos sean de tipo numérico o cadenas de texto.

Sin embargo, es importante entender que esto no es lo normal. Un operador espera que sus operandos sean de un determinado tipo o que, por lo menos, puedan convertirse a dicho tipo.

Por ejemplo, si deseas realizar una multiplicación, es lógico pensar que una expresión del tipo **`4 * "hola"`** no tenga sentido. Sin embargo, una expresión como **`4 * "22"`** daría el resultado **88**, ya que la cadena **`"22"`** puede convertirse a un valor numérico válido.

Volvamos al operador **+**. Este operador puede actuar tanto con operandos numéricos como con operandos de texto. Por ejemplo, si escribes una expresión del tipo **`2 + 2`**, donde ambos valores son numéricos, está claro que el resultado será la suma de ambos, es decir, el valor **4**.

Sin embargo, ¿qué pasa si alguno de los operandos es una cadena de texto? En la expresión **`25 + 5 + "45"`** los dos primeros operandos son numéricos, por lo que el resultado será equivalente a **`30 + "45"`**, que produce la cadena de texto **"3045"**.

Esto nos hace ver que cuando el operador + se utiliza con alguna cadena de texto, entonces el resultado final es otra cadena. Por ello hay que tener cuidado a la hora de utilizar este operador.

Por otra parte, cuando se utiliza más de un operando en una expresión, es necesario conocer la precedencia con que se aplican. Por ejemplo, si escribes una expresión del tipo `(4 + 5 * 2)`, ¿qué se realiza primero la suma o la multiplicación?

La respuesta está clara: primero se multiplica `5 * 2` y al resultado se le suma **4**, dando el resultado final **14**. ¿Qué ocurre, sin embargo, cuando se aplica el mismo operador más de una vez en la expresión? En este caso se aplica de izquierda a derecha.

A continuación puedes encontrar la precedencia de los operadores en JavaScript. No es necesario recordar que utilizando paréntesis podemos forzar la evaluación de cierta parte de una expresión antes que otras:

MÁS PRECEDENCIA

`( )`

`!, -, ++, --`

`*, /, %`

`+, -`

`<, <=, >, >=`

`==, !=`

`=, +=, -=, *=, /=, %=`

MENOS PRECEDENCIA

Los operadores con la misma precedencia se evalúan de izquierda a derecha.

Un símbolo = es una asignación y dos símbolos == es una comparación de igualdad.

Otros operadores que se utilizan mucho son los que permiten realizar comparaciones como > (mayor que), < (menor que), == (igual a), >= (mayor o igual que), <= (menor o igual que), etc.

Además, ya has utilizado el operador más importante, que es el de asignación. Con el operador = estás estableciendo el valor a una variable. ¡No lo confundas con el operador de igualdad!

6. CUADROS DE DIÁLOGO

Antes de estudiar las estructuras de control que nos permiten modificar el flujo secuencial de un script, vamos a introducir los **cuadros de diálogo**.

Utilizando los cuadros de diálogo que incorpora el lenguaje JavaScript, podrás interaccionar con el usuario, de forma que este facilite información para continuar con la ejecución del script.

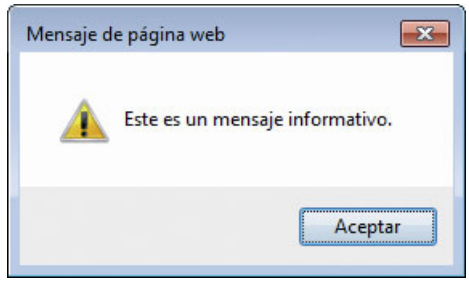
En otros casos, el cuadro de diálogo simplemente servirá para informar de cierta circunstancia al usuario, como puede ser el suceso de un error o algún mensaje de advertencia.

En JavaScript existen tres cuadros de diálogo predefinidos:

- **alert(mensaje)**: sirve para enviar un mensaje de información al usuario.

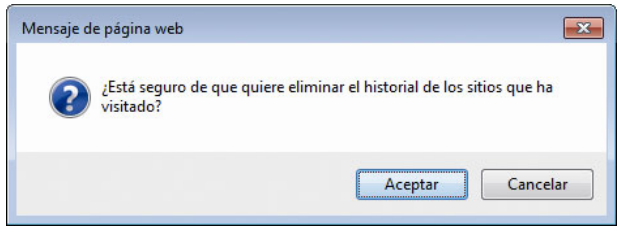
Es el más sencillo de todos, ya que lo único que se tiene que indicar es el mensaje que aparecerá. El cuadro de diálogo incluye dicho mensaje y un botón para aceptarlo, que es la única acción que puede realizar el usuario.

```
alert("Este es un mensaje informativo.");
```



Gracias al título del cuadro de diálogo, se puede diferenciar entre lo que es un cuadro propio del sistema operativo y otro correspondiente a una página web. Incluso en otros navegadores se indica que el cuadro proviene de una aplicación JavaScript.

- **confirm(mensaje)**: sirve para que el usuario confirme la realización de alguna tarea. A diferencia del cuadro de diálogo anterior, ahora aparecen dos botones (uno para aceptar y otro para cancelar).



Por ejemplo con el código:

```
variable = confirm("¿Está seguro de que quiere eliminar
el historial de los sitios que ha visitado?");
```

se obtiene en **variable** el botón que el usuario ha utilizado para cerrar el cuadro de diálogo. Esta variable tendrá el valor **true** (verdadero) si el usuario pulsa el botón para aceptar el cuadro y **false** (falso) en el caso de que lo haya cancelado.

- **prompt(mensaje[, valor por defecto]**: sirve para recabar información por parte del usuario.

La primera cadena indica el mensaje que aparecerá en el cuadro de diálogo. La segunda cadena es opcional y expresa el valor inicial que aparecerá en el campo donde el usuario introduce la información requerida.

Cuando en la sintaxis de una función una parte de la misma aparece entre corchetes **[ ]**, debes entender que esa parte es opcional, es decir, que puede aparecer o no.

Por ejemplo**,** prompt(mensaje[, valor_defecto]) quiere decir que la función **prompt** necesita del primer parámetro y, opcionalmente, del segundo (de ahí que la coma que los separa también es opcional y solo aparecerá cuando indiquemos el segundo parámetro).

Por ejemplo con el código:

```
variable = prompt("Introduzca su nombre:", "anónimo");
```

se obtiene en **variable** el valor introducido por el usuario. Si el usuario pulsa en el botón ***Cancelar***, entonces el resultado que se recibe es el valor **null**; si pulsa en ***Aceptar***, lo que se recibe es la cadena que ha escrito (o la inicial si no ha escrito nada).