

ISSD

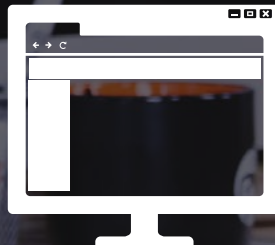
—Asc—
Analista de
Sistemas

JAVASCRIPT

I2

Informática 2

Módulo didáctico - 2015



Unidad 3



Clase 9



(U3) JavaScript

Luego de terminada esta clase ya podrás...

- | Conocer la importancia de JavaScript en el mundo del desarrollo web.
- | Interpretar e implementar el lenguaje propuesto a los ya aprendidos (HTML y CSS3)
- | Comprender de qué manera funcionan los eventos y cómo se asocia a estos JavaScript.
- | Utilizar en forma correcta la sintaxis del lenguaje y las distintas estructuras que presenta según los ejercicios expuestos.

A esta altura, ya dominás más o menos HTML. ¿Qué opinás de los límites del lenguaje? ¿Hasta dónde crees que se puede programar con HTML? No todo se puede realizar con HTML5. Para poder trabajar en algunos eventos y funciones del navegador, o bien para validar datos debés hacer uso de otro lenguaje potente para el desarrollo web: JavaScript. Las próximas clases van a ser bastante dinámicas, ya que este lenguaje te debe resultar familiar por la similitud de la sintaxis que posee con Java.

The screenshot shows a web browser window with the title 'Formulario'. The address bar displays the file path 'file:///D:/ITEHTML/14formularios/eje/fig1414.html'. The main content area features a heading 'Bienvenido' followed by a section titled 'Datos personales' containing a text input field with the placeholder 'Introduce tu nombre:'. Below this is a 'Mensaje:' label and a text area. A JavaScript alert dialog box is overlaid on the form, displaying the message 'Perfecto. La página ya se ha cargado.' and an 'Aceptar' button. At the bottom of the form, there is a button labeled 'Haz clic sobre mi' and a section titled 'Selecciona el color de tu icono:' with three radio button options: 'Rojo', 'Verde', and 'Azul'.



JavaScript

Una vez aprendido y asimilado **HTML+CSS3** vamos a desarrollar JavaScript para mejorar el aspecto y el control del navegador y de algunos eventos. Particularmente en esta clase veremos:

| Las características de JavaScript respecto a otros lenguajes.

| Su gran importancia para los desarrolladores web.

| También en esta clase vas a comenzar a interpretarlo e implementarlo.

| Y por último, aprenderás de qué manera funcionan los eventos y como se asocian JavaScript

Este lenguaje te tiene que ser familiar ya que comparte características y estructuras del lenguaje **Java** que estas viendo en otra materia, así que no debe causarte dificultad alguna. **JavaScript** conjuntamente con **HTML5** y **CSS3** se convierte en una de las potencias dinámicas más estudiadas a nivel de lenguajes Web. De más esta decir que para adquirir experiencia en este lenguaje debes practicar codificando los ejercicios.

¡Qué disfrutes la clase!



Desempeño de Exploración

Te invitamos a intercambiar opiniones, experiencias e inquietudes sobre JavaScript con tus compañeros de manera grupal. Luego compartiremos experiencias laborales referentes al desarrollo web y al uso de este lenguaje en el mercado actual.

¿Qué es Javascript?

Javascript es un lenguaje de programación utilizado para crear programas encargados de realizar acciones dentro del ámbito de una página web. Con Javascript podemos crear efectos especiales en las páginas y definir interactividades con el usuario. El navegador del cliente es el encargado de interpretar las instrucciones Javascript y ejecutarlas para realizar estos efectos e interactividades, de modo que el mayor recurso -y tal vez el único- con que cuenta este lenguaje es el propio navegador.

Javascript es un lenguaje con muchas posibilidades, permite la programación de pequeños scripts, pero también de programas más grandes, orientados a objetos, con funciones, estructuras de datos complejas,

etc. Toda esta potencia de Javascript se pone a disposición del programador.

Estructura Básica

Javascript es un lenguaje interpretado y orientado a html; por tanto, no necesitamos ningún compilador, nos basta y sobra con que nuestro navegador sea compatible con Javascript. Pues bien, la incrustación de un código de Javascript puede ser de dos formas:

- | Desde la misma página HTML.
- | Llamar el código desde un archivo externo.

Veamos esto con más detenimiento.

Hoy en día la mayoría de los navegadores de internet admiten Javascript, aunque en algunos casos el navegador tiene deshabilitado por defecto la ejecución de scripts.

Desde la página HTML

Para incrustar un código de Javascript desde la misma página HTML se toman en cuenta los siguientes puntos:

| El script se coloca dentro de las etiquetas **<head>** y **</head>**

| Las etiquetas para ejecutar este script son **<script>** y **</script>**. Dentro de estas etiquetas se incluye el atributo **"language=javascript"** el cual indica que nuestro script se basa bajo este lenguaje (también existen otros lenguajes, por ejemplo vbscript, JScript, etc.).

Veamos los ejemplos del cuadro: (1)

Desde un archivo externo

Para incrustar un código de Javascript desde un archivo externo debemos tener en cuenta estos puntos:

| Para llamar un archivo con el código en Javascript basta con colocar las mismas etiquetas de **< script >** y **< / script >** dentro de la cabecera (**head**) y colocar dos atributos

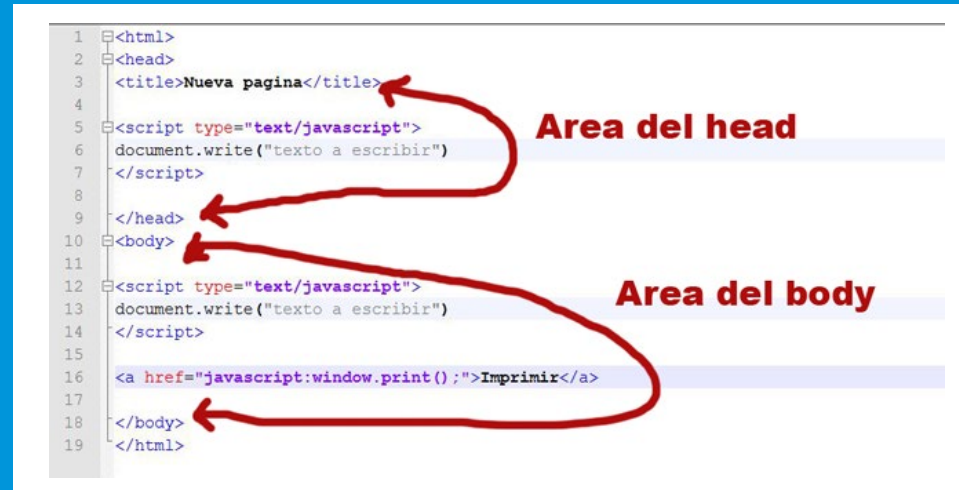
Ejemplo 01

```
< html >
< head >
< script language= "javascript" >
<!--
alert ("Mi primer programa en
Javascript"); //alert es una función
para mostrar en pantalla un mensaje
//-- >
< / script >
< / head >

< body >
< / body >
< / html >
```

Ejemplo 02

```
<html>
<head>
<title>Embeber JavaScript
en HTML5</title>
</head>
<body>
<script type="text/
javascript">
document.write('Hola
Mundo');
</script>
</body>
</html>
```



que son **type="text/javascript"** y **src="ruta del archivo"**.

| El código se guarda en un archivo con extensión .js; que es la extensión de los archivos Javascript y el código ya no se encierra entre los comentarios de HTML.

Veamos el siguiente ejemplo:

En un archivo llamado **codigo.js** usaremos la función alert ("Mi Primer programa en Java") como en el ejemplo anterior para mostrar en pantalla un mensaje. El archivo se ubica en la misma carpeta que nuestra página web.

```

< html >
< head >
< script type="text/javascript" src="codigo.js"
> < / script >
< / head >
< body >
< / body >
< / html >

```

Dependiendo de las necesidades o facilidades del programador, uno decidirá cuál es la mejor opción para sus aplicaciones web, ya sea desde un archivo externo o desde la misma página.

Observá además la imagen 1 para aclarar cuál es el area que corresponde al Head y cuál al Body.

Sintaxis

El lenguaje **Javascript** tiene una sintaxis muy parecida a la de **Java** por estar basado en él. También es muy parecida a la del **lenguaje C**, de modo que si conocés alguno de estos dos lenguajes te podrás manejar con facilidad con el código.

Comentarios en el código

El programador, a medida que desarrolla el script, va dejando frases o palabras sueltas, llamadas comentarios, que le ayudan a él o a cualquier otro a leer mas fácilmente el script a la hora de modificarlo o depurarlo. Existen dos tipos de comentarios en el lenguaje

| Uno de ellos, **la doble barra**, sirve para comentar una línea de código.

| El otro comentario lo podemos utilizar para comentar varias líneas y se indica con los signos `/*` para empezar el comentario y `*/` para terminarlo.

Veamos unos ejemplos:

```
<SCRIPT>
```

```
//Este es un comentario de una línea
```

```
/*Este comentario se puede extender  
por varias líneas.
```

```
Las que quieras*/
```

```
</SCRIPT>
```

Mayúsculas y minúsculas

En **Javascript** se han de respetar las mayúsculas y las minúsculas. Si nos equivocamos al utilizarlas el navegador responderá con un mensaje de error, ya sea de sintaxis o de referencia indefinida.

Un ejemplo claro podemos encontrarlo cuando tratemos con variables, puesto que los nombres que damos a las variables también son sensibles a las mayúsculas y minúsculas. A medida que se desarrolle el contenido de la clase veras como se implementa la sintaxis en el lenguaje.

Separación de instrucciones

Las distintas instrucciones que contienen nuestros scripts se han de separar convenientemente para que el navegador no indique los correspondientes errores de sintaxis. Javascript tiene dos maneras de separar instrucciones: la primera es a través del carácter punto y coma (;) y la segunda es a través de un salto de línea.

Un comentario es una parte de código que no es interpretada por el navegador y cuya utilidad radica en facilitar la lectura al programador.



Te voy a dar un ejemplo: no es lo mismo la función `alert()` que la función `Alert()`



¡Claro! Porque la primera muestra un texto en una caja de diálogo y la segunda (con la primera **A** mayúscula) simplemente no existe, a no ser que la definamos nosotros.

Diferencias entre Java y Javascript

Javascript se llamó así porque **Netscape**, que estaba aliado a los creadores de **Java** en la época, quiso aprovechar el conocimiento y la percepción que las personas tenían del popular lenguaje. Con todo, se creó un producto que tenía ciertas similitudes (como la sintaxis del lenguaje o el nombre) y se hizo entender que era un hermano pequeño y orientado específicamente para hacer cosas en las páginas web, pero también se hizo caer a muchas personas en el error de pensar que son lo mismo.

Javascript no tiene nada que ver con **Java**, salvo en sus orígenes, como se ha podido leer hace unas líneas. Actualmente son productos totalmente distintos y no guardan entre si más relación que la sintaxis idéntica y poco más. Algunas diferencias entre estos dos lenguajes son las siguientes:

| Diferencias entre Java y Javascript | | |
|-------------------------------------|--|--|
| Características | Java | Javascript |
| Compilador | Para programar en Java necesitamos un Kit de desarrollo y un compilador. | Javascript no es un lenguaje que necesite que sus programas se compilen, sino que éstos se interpretan por parte del navegador cuando éste lee la página. |
| Orientado a objetos | Java es un lenguaje de programación orientado a objetos. | Javascript no es orientado a objetos, esto quiere decir que podremos programar sin necesidad de crear clases, tal como se realiza en los lenguajes de programación estructurada como C o Pascal. |
| Propósito | Java es mucho más potente que Javascript, esto es debido a que Java es un lenguaje de propósito general, con el que se pueden hacer aplicaciones de lo más variadas. | Con Javascript sólo podemos escribir programas para que se ejecuten en páginas web. |
| Estructuras fuertes | Java es un lenguaje de programación fuertemente tipado, esto quiere decir que al declarar una variable tendremos que indicar su tipo y no podrá cambiar de un tipo a otro automáticamente. | Por su parte Javascript no tiene esta característica, y podemos meter en una variable la información que deseemos, independientemente del tipo de ésta. Además, podremos cambiar el tipo de información de una variable cuando queramos. |
| Otras características | Como vemos Java es mucho más complejo, aunque también más potente, robusto y seguro. Tiene más funcionalidades que Javascript y las diferencias que los separan son lo suficientemente importantes como para distinguirlos fácilmente. | |

Variables

Una **variable** es un espacio en la memoria donde se almacena un dato, un espacio donde podemos guardar cualquier tipo de información que necesitemos para realizar las acciones de nuestros programas.

Por ejemplo, si nuestro programa realiza sumas, será muy normal que guardemos en variables los distintos sumandos que participan en la operación y el resultado de la suma.

Los nombres de las variables han de construirse con caracteres alfanuméricos y el carácter subrayado (_). Aparte de esta, hay una serie de reglas adicionales para construir nombres para variables. La más importante es que tienen que comenzar por un carácter alfabético o el subrayado.

No podemos utilizar caracteres raros como el signo +, un espacio o un \$. También hay que evitar utilizar nombres reservados como variables, por ejemplo no podremos llamar a nuestra variable palabras como return o

for, que ya veremos que son utilizadas para estructuras del propio lenguaje.

Declaración de variables en Javascript

Declarar variables consiste en definir y de paso informar al sistema, que vas a utilizar una variable. Es una costumbre habitual en los lenguajes de programación el definir las variables que se van a usar en los programas y para ello, se siguen unas reglas estrictas.

Javascript cuenta con la palabra “var” que utilizaremos cuando queramos declarar una o varias variables. Como es lógico, se utiliza esa palabra para definir la variable antes de utilizarla.

var operando1
var operando2

También se puede asignar un valor a la variable cuando se está declarando:

var operando1 = 23
var operando2 = 33

* ! #



¿Vos te acordás qué es una variable?

¡Si! Una variable es un espacio en la memoria donde podemos guardar cualquier tipo de información que necesitemos para realizar las acciones de nuestros programas.
¡Es importante que vos también lo recuerdes porque vamos a usarlas seguido!



Además se permite declarar varias variables en la misma línea, siempre que se separen por comas.

var operando1,operando2

Tipos de variables

Aunque todas las variables de **JavaScript** se crean de la misma forma (mediante la palabra reservada **var**), la forma en la que se les asigna un valor depende del tipo de valor que se quiere almacenar (números, textos, etc.)

Numéricas

Se utilizan para almacenar valores numéricos enteros (llamados **integer** en inglés) o decimales (llamados **float** en inglés). En este caso, el valor se asigna indicando directamente el número entero o decimal. Los números decimales utilizan el carácter **.** (**punto**) en vez de **,** (**coma**) para separar la parte entera y la parte decimal:

```
var iva = 16;    // variable tipo entero
var total = 234.65; // variable tipo decimal
```

Cadenas de texto

Se utilizan para almacenar caracteres, palabras y/o frases de texto. Para asignar el valor a la variable, se encierra el valor entre comillas dobles o simples, para delimitar su comienzo y su final:

```
var mensaje = "Bienvenido a nuestro sitio web";
var nombreProducto = 'Producto ABC';
var letraSeleccionada = 'c';
```

En ocasiones, el texto que se almacena en las variables no es tan sencillo. Si por ejemplo el propio texto contiene comillas simples o dobles, la estrategia que se sigue es la de encerrar el texto con las comillas (simples o dobles) que no utilice el texto:

```
/* El contenido de texto1 tiene comillas
simples, por lo que se encierra con comillas
dobles */
var texto1 = "Una frase con 'comillas simples'
dentro";
/* El contenido de texto2 tiene comillas
dobles, por lo que se encierra con comillas
simples */
var texto2 = 'Una frase con "comillas dobles"
dentro';
```

Aunque Javascript no nos obligue a declarar explícitamente las variables, es aconsejable declararlas antes de utilizarlas. En lo que sigue veremos que se trata de una buena costumbre que te ayudará en tu trabajo.



| Si se quiere incluir... | Se debe incluir... |
|---|--|
| Una nueva línea | \n |
| Un tabulador | \t |
| Una comilla simple | \' |
| Una comilla doble | \" |
| Una barra inclinada | \\ |
| Ed mo nihilicus? quid co es hactur abervivis et L. | Serrio, demus; ium conscia sulicae clerimilicut oc, et; |

No obstante, a veces las cadenas de texto contienen tanto comillas simples como dobles. Además, existen otros caracteres que son difíciles de incluir en una variable de texto (tabulador, **ENTER**, etc.) Para resolver estos problemas, **JavaScript** define un mecanismo para incluir de forma sencilla caracteres especiales y problemáticos dentro de una cadena de texto.

El mecanismo consiste en sustituir el carácter problemático por una combinación simple de caracteres. Observá arriba la tabla de conversión que debés utilizar: **(1)**

De esta forma, el ejemplo anterior que contenía comillas simples y dobles dentro del texto se puede rehacer de la siguiente forma:

```
var texto1 = 'Una frase con \'comillas simples\' dentro';
var texto2 = "Una frase con \"comillas dobles\" dentro";
```

Este mecanismo de JavaScript se denomina **mecanismo de escape de los caracteres problemáticos**, y es habitual referirse a que los caracteres han sido escapados.

Arrays

En ocasiones, a los **arrays** se les llama **vectores**, **matrices** e incluso **arreglos**. No obstante, el término **array** es el más utilizado y es una palabra comúnmente aceptada en el entorno de la programación.

Un array es una colección de variables, que pueden ser todas del mismo tipo o cada una de un tipo diferente. Su utilidad se comprende mejor con un ejemplo sencillo: si una aplicación necesita manejar los días de la semana, se podrían crear siete variables de tipo texto:



Supongamos que tenemos que guardar el nombre de todos los países del mundo o las mediciones diarias de temperatura de los últimos 100 años ;se tendrían que crear decenas o cientos de variables!



Tranquilo...para eso justamente se pueden agrupar todas las variables relacionadas en un array.

```
var dia1 = "Lunes";  
var dia2 = "Martes";  
...  
var dia7 = "Domingo";
```

Aunque el código anterior no es incorrecto, sí que es poco eficiente y complica en exceso la programación. Si en vez de los días de la semana se tuviera que guardar el nombre de los meses del año, por ejemplo, se tendrían que crear gran cantidad de variables.

En este tipo de casos, se pueden agrupar todas las variables relacionadas en una **colección de variables o array**. El ejemplo

anterior se puede rehacer de la siguiente forma:

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

Ahora, una única variable llamada **dias** almacena todos los valores relacionados entre sí, en este caso los días de la semana. Para definir un array, se utilizan los caracteres **[y]** para delimitar su comienzo y su final y se utiliza el carácter **,** (**coma**) para separar sus elementos:

```
var nombre_array = [valor1, valor2, ..., valorN];
```

Una vez definido un **array**, es muy sencillo acceder a cada uno de sus elementos. Cada elemento se accede indicando su posición dentro del array. La única complicación que es responsable de muchos errores cuando se empieza a programar, es que las posiciones de los elementos empiezan a contarse en el **0** y no en el **1**:

```
var diaSeleccionado = dias[0]; //  
diaSeleccionado = "Lunes"  
var otroDia = dias[5]; // otroDia =  
"Sábado"
```

En el ejemplo anterior, la primera instrucción quiere obtener el primer elemento del array. Para ello, se indica el nombre del array y entre corchetes la posición del elemento dentro del array. Como se ha comentado, las posiciones se empiezan a contar en el **0**, por lo que el primer elemento ocupa la posición **0** y se accede a él mediante `dias[0]`.

El valor `dias[5]` hace referencia al elemento que ocupa la sexta posición dentro del array **dias**. Como las posiciones empiezan a contarse en **0**, la posición **5** hace referencia al sexto elemento, en este caso, el valor **sábado**.

Booleanos

Las variables de tipo **boolean** o **booleano** también se conocen con el nombre de variables de tipo lógico. Una variable de tipo **boolean** almacena un tipo especial de valor que solamente puede tomar dos valores: **true** (verdadero) o **false** (falso). No se puede utilizar para almacenar números y tampoco permite guardar cadenas de texto.

Los únicos valores que pueden almacenar estas variables son **true** y **false**, por lo que

no pueden utilizarse los valores **verdadero** y **falso**. A continuación se muestra un par de variables de tipo booleano:

```
var clienteRegistrado = false;  
var ivaIncluido = true;
```

Estructuras de control

Si tenemos alguna experiencia en la programación sabremos que en los programas generalmente se necesitará hacer cosas distintas dependiendo del estado de nuestras variables o realizar un mismo proceso muchas veces sin escribir las mismas líneas de código una y otra vez. Para realizar cosas más complejas en nuestros scripts se utilizan las estructuras de control. Con ellas podemos realizar **tomas de decisiones** y **bucles**. A continuación veremos una simple reseña de estos controles ya que los debes tener familiarizados con **Algoritmos** y **estructura de datos**.

Toma de decisiones (estructura condicional)

Nos sirven para realizar unas acciones u otras en función del estado de las variables.

Un **array** es una colección de variables, que pueden ser todas del mismo tipo o cada una de un tipo diferente. Dentro de los tipos de variables encontramos las boolean, booleanas o de tipo lógico, que son las que almacenan un tipo especial de valor que solamente puede tomar dos valores: true (verdadero) o false (falso).

Es decir, tomar decisiones para ejecutar unas instrucciones u otras dependiendo de lo que esté ocurriendo en ese instante en nuestros programas. Por ejemplo, dependiendo si el usuario que entra en nuestra página es mayor de edad o no lo es, podemos permitirle o no ver los contenidos de nuestra página.

Estructura if

La estructura más utilizada en JavaScript y en la mayoría de lenguajes de programación es la estructura if. Se emplea para tomar decisiones en función de una condición. Su definición formal es:

```
if(condicion) {  
  ...  
}
```

Si la condición se cumple (es decir, si su valor es **true**) se ejecutan todas las instrucciones que se encuentran dentro de {...}. Si la condición no se cumple (es decir, si su valor es **false**) no se ejecuta ninguna instrucción contenida en {...} y el programa continúa ejecutando el resto de instrucciones del script.

Por ejemplo:

```
var mostrarMensaje = true;  
if(mostrarMensaje) {  
  alert("Hola Mundo");  
}
```

En el ejemplo anterior, el mensaje sí se muestra al usuario ya que la variable **mostrarMensaje** tiene un valor de **true** y por tanto, el programa entra dentro del bloque de instrucciones del **if**.

El ejemplo se podría reescribir también como:

```
var mostrarMensaje = true;  
if(mostrarMensaje == true) {  
  alert("Hola Mundo");  
}
```

En este caso, la condición es una comparación entre el valor de la variable **mostrarMensaje** y el valor **true**. Como los dos valores coinciden, la igualdad se cumple y por lo tanto la condición es cierta, su valor es **true** y se ejecutan las instrucciones contenidas en ese bloque del **if**.

Con las estructuras de control podés realizar tomas de decisiones y bucles



Las tomas de decisiones sirven para ejecutar unas instrucciones u otras dependiendo de lo que esté ocurriendo en ese instante en nuestros programas. La más usada es la Estructura if que se emplea para tomar decisiones en función de una condición.



La comparación del ejemplo anterior suele ser el origen de muchos errores de programación, al confundir los operadores `==` y `=`. Las comparaciones siempre se realizan con el operador `==`, ya que el operador `=` solamente asigna valores:

```
var mostrarMensaje = true;
// Se comparan los dos valores
if(mostrarMensaje == false) {
  ...
}
```

16

TSSD

```
// Error - Se asigna el valor "false" a la variable
if(mostrarMensaje = false) {
  ...
}
```

La condición que controla el `if()` puede combinar los diferentes operadores lógicos y relacionales mostrados anteriormente:

```
var mostrado = false;
if(!mostrado) { alert("Es la primera vez que se muestra el mensaje"); }
```

Los operadores **AND** y **OR** permiten encadenar varias condiciones simples para construir condiciones complejas:

```
var mostrado = false;
var usuarioPermiteMensajes = true;
if(!mostrado && usuarioPermiteMensajes) {
  alert("Es la primera vez que se muestra el mensaje");
}
```

La condición anterior está formada por una operación **AND** sobre dos variables. A su vez, a la primera variable se le aplica el operador de negación antes de realizar la operación **AND**. De esta forma, como el valor de **mostrado** es **false**, el valor **!mostrado** sería **true**.

Como la variable **usuarioPermiteMensajes** vale **true**, el resultado de **!mostrado && usuarioPermiteMensajes** sería igual a **true && true**, por lo que el resultado final de la condición del `if()` sería **true** y por tanto, se ejecutan las instrucciones que se encuentran dentro del bloque del `if()`.

En JavaScript disponemos de los operadores lógicos habituales en lenguajes de programación como son AND y OR. Como podés observar en los ejemplos, su sintaxis se basa en símbolos.



¡Hay que prestar atención a no confundir `==` con `=` porque implican distintas cosas!

* ! #



Algunas veces, las decisiones que se deben realizar suelen ser del tipo "si se cumple esta condición, hazlo; si no se cumple, haz esto otro", pero ¿cómo logro este resultado con la estructura if?

Justamente para eso existe una variante de la estructura if llamada if... else.



Estructura if...else

En ocasiones, las decisiones que se deben realizar no son del tipo "si se cumple la condición, hazlo; si no se cumple, no hagas nada". Normalmente las condiciones suelen ser del tipo "si se cumple esta condición, hazlo; si no se cumple, haz esto otro".

Para este segundo tipo de decisiones, existe una variante de la estructura **if** llamada **if...else**. Su definición formal es la siguiente:

```
if(condicion) {  
    ...  
}else {  
    ...  
}
```

Si la condición se cumple (es decir, si su valor es **true**) se ejecutan todas las instrucciones

que se encuentran dentro del **if()**. Si la condición no se cumple (es decir, si su valor es **false**) se ejecutan todas las instrucciones contenidas en **else { }**. Por ejemplo:

```
var edad = 18;  
if(edad >= 18) { alert("Eres mayor de edad"); }  
else { alert("Todavía eres menor de edad"); }
```

Si el valor de la variable **edad** es mayor o igual que el valor numérico **18**, la condición del **if()** se cumple y por tanto, se ejecutan sus instrucciones y se muestra el mensaje "**Eres mayor de edad**". Sin embargo, cuando el valor de la variable **edad** no es igual o mayor que **18**, la condición del **if()** no se cumple, por lo que automáticamente se ejecutan todas las instrucciones del bloque **else { }**. En este caso,

se mostraría el mensaje "**Todavía eres menor de edad**".

El siguiente ejemplo compara variables de tipo cadena de texto:

```
var nombre = "";  
if(nombre == "") { alert("Aún no nos has dicho tu nombre"); }  
else {  
    alert("Hemos guardado tu nombre");  
}
```

La condición del **if()** anterior se construye mediante el operador **==**, que es el que se emplea para comparar dos valores (no confundir con el operador **=** que se utiliza para

asignar valores). En el ejemplo anterior, si la cadena de texto almacenada en la variable **nombre** es vacía (es decir, es igual a "") se muestra el mensaje definido en el **if()**. En otro caso, se muestra el mensaje definido en el bloque **else { }**. La estructura **if...else** se puede encadenar para realizar varias comprobaciones seguidas:

```
if(edad < 12) {  
  alert("Todavía eres muy pequeño");  
}  
else if(edad < 19) {  
  alert("Eres un adolescente");  
}  
else if(edad < 35) {  
  alert("Aun sigues siendo joven");  
}  
else {  
  alert("Piensa en cuidarte un poco más");  
}
```

No es obligatorio que la combinación de estructuras **if...else** acabe con la instrucción **else**, ya que puede terminar con una instrucción de tipo **else if()**.

Bucles

Los bucles se utilizan para realizar ciertas acciones repetidamente. Son muy utilizados a todos los niveles en la programación. Con un bucle podemos por ejemplo imprimir en una página los números del 1 al 100 sin necesidad de escribir cien veces la instrucción imprimir. En javascript existen varios tipos de bucles, cada uno está indicado para un tipo de iteración distinto y son los siguientes:

- | For
- | While
- | Do While

A continuación describiremos cada uno de ellos:

Estructura for

La estructura for permite realizar este tipo de repeticiones (también llamadas bucles) de una forma muy sencilla. Su sintaxis es la siguiente:

```
for(inicializacion; condicion; actualizacion) {  
  ...  
}
```

Dijimos que con las estructuras de control podés realizar tomas de decisiones y bucles.



Los bucles se utilizan para realizar ciertas acciones repetidamente. En JavaScript existen los tipos For, While y Do While, cada uno está indicado para un tipo de iteración distinto.



La idea del funcionamiento de un bucle for es la siguiente: **“mientras la condición indicada se siga cumpliendo, repite la ejecución de las instrucciones definidas dentro del for. Además, después de cada repetición, actualiza el valor de las variables que se utilizan en la condición”**.

| La **“inicialización”** es la zona en la que se establece los valores iniciales de las variables que controlan la repetición.

| La **“condición”** es el único elemento que decide si continua o se detiene la repetición.

| La **“actualización”** es el nuevo valor que se asigna después de cada repetición a las variables que controlan la repetición.

```
var mensaje = "Hola, estoy dentro de un bucle";  
for(var i = 0; i < 5; i++) { alert(mensaje);}
```

La parte de la inicialización del bucle consiste en:

```
var i = 0;
```

Por tanto, en primer lugar se crea la variable **i** y se le asigna el valor de **0**. Esta zona de inicialización solamente se tiene en consideración justo antes de comenzar a ejecutar el bucle. Las siguientes repeticiones no tienen en cuenta esta parte de inicialización. La zona de condición del bucle es **i < 5**.

Los bucles se siguen ejecutando mientras se cumplan las condiciones y se dejan de ejecutar justo después de comprobar que la condición no se cumple. En este caso, mientras la variable **i** valga menos de **5** el bucle se ejecuta indefinidamente.

Como la variable **i** se ha inicializado a un valor de **0** y la condición para salir del bucle es que **i** sea menor que **5**, si no se modifica el valor de **i** de alguna forma, el bucle se repetiría indefinidamente. Por ese motivo, es imprescindible indicar la zona de actualización, en la que se modifica el valor de las variables que controlan el bucle: **i++**

En este caso, el valor de la variable **i** se incrementa en una unidad después de cada repetición. La zona de actualización se ejecuta

La Estructura For permite realizar repeticiones (bucles) de manera sencilla y responde a la siguiente idea de funcionamiento: “mientras la condición indicada se siga cumpliendo, repite la ejecución de las instrucciones definidas dentro del for. Además, después de cada repetición, actualiza el valor de las variables que se utilizan en la condición”.



La "inicialización" es la zona en la que se establece los valores iniciales de las variables que controlan la repetición.



La "condición" es el único elemento que decide si continua o se detiene la repetición.



La "actualización" es el nuevo valor que se asigna después de cada repetición a las variables que controlan la repetición.

después de la ejecución de las instrucciones que incluye el **for**. Así, durante la ejecución de la quinta repetición el valor de **i** será **4**. Después de la quinta ejecución, se actualiza el valor de **i**, que ahora valdrá **5**. Como la condición es que **i** sea menor que **5**, la condición ya no se cumple y las instrucciones del **for** no se ejecutan una sexta vez.

Normalmente, la variable que controla los bucles **for** se llama **i**, ya que recuerda a la palabra índice y su nombre tan corto ahorra mucho tiempo y espacio. El ejemplo anterior que mostraba los días de la semana contenidos en un array se puede rehacer de forma más sencilla utilizando la estructura **for**:

```
var dias = ["Lunes", "Martes", "Miércoles",  
"Jueves", "Viernes", "Sábado", "Domingo"];  
for(var i=0; i<7; i++) { alert(dias[i]); }
```

Estructura **for...in**

Una estructura de control derivada de **for** es la estructura **for...in**. Su definición exacta implica el uso de objetos, que es un elemento de programación avanzada que no se va a estudiar. Por tanto, solamente se va a presentar la estructura **for...in** adaptada a su uso en arrays. Su definición formal adaptada a los arrays es:

```
for(indice in array) {  
  ...  
}
```

Si se quieren recorrer todos los elementos que forman un array, la estructura **for...in** es la forma más eficiente de hacerlo, como se muestra en el siguiente ejemplo:

```
var dias = ["Lunes", "Martes", "Miércoles",  
"Jueves", "Viernes", "Sábado", "Domingo"];  
  
for(i in dias) { alert(dias[i]);}
```

La variable que se indica como índice es la que se puede utilizar dentro del bucle **for...in** para acceder a los elementos del array. De esta forma, en la primera repetición del bucle la variable **i** vale **0** y en la última vale **6**. Esta estructura de control es la más adecuada para recorrer arrays (y objetos), ya que evita tener que indicar la inicialización y las

condiciones del bucle **for** simple y funciona correctamente cualquiera que sea la longitud del array. De hecho, sigue funcionando igual aunque varíe el número de elementos del array.

While

Este bucle ejecuta sus sentencias mientras una condición especificada sea evaluada como verdadera. Su sintaxis es la siguiente:

```
while (condición){  
    sentencia  
}
```

Si la condición llega a ser falsa, la **sentencia** dentro del bucle para la ejecución y el control pasa a la **sentencia** siguiente luego del bucle. La condición de prueba ocurre antes que la sentencia en el bucle sea ejecutada. Si la condición retorna verdadero, la **sentencia** es ejecutada y la condición es probada nuevamente. Si la condición retorna falso, la ejecución se para y el control es pasado a la siguiente **sentencia** después del bucle **while**. Para ejecutar múltiples sentencias, utilice los bloques de sentencias (`{ ... }`) para agruparlas.

Ejemplo 01: El siguiente bucle **while** iteraría mientras que **n** sea menor que tres:

```
n = 0; x = 0;  
while (n < 3) {  
    n++;  
    x += n;  
}
```

Con cada iteración, el bucle se incrementa por **n** y añade el valor a **x**. Sin embargo, **x** y **n** toman los siguientes valores:

- | Después de la primera pasada: **n** = 1 y **x** = 1
- | Después de la segunda pasada: **n** = 2 y **x** = 3
- | Después de la tercera pasada: **n** = 3 y **x** = 6

Después de completarse la tercera pasada, la condición **n** < 3 ya no es más verdadera, así que el bucle termina.

Ejemplo 02:

Evite bucles infinitos. Asegúrese que la condición dentro de un bucle eventualmente llegue a ser falsa; caso contrario, el bucle nunca terminará. La sentencia en el siguiente bucle **while** se ejecutará por siempre porque la condición nunca llega a ser falsa:

El bucle While en JavaScript se usa para ejecutar un pedazo de código mientras la condición siga siendo verdadera. Dicha condición es evaluada antes de ejecutar la sentencia.



Asegurate que la condición dentro de un bucle eventualmente llegue a ser falsa, en caso contrario ¡el bucle será infinito!



¿Por qué usaría el bucle do...while y no directamente while?

El bucle do...while se usa generalmente -igual que el bucle WHILE- cuando no sabemos cuántas veces se habrá de ejecutar el bucle, pero con la diferencia de que sabemos seguro que el bucle por lo menos se ejecutará una vez.



```
while (true) {  
    alert("Hola, mundo");  
}
```

Ejemplo 03:

En este ejemplo vamos a declarar una variable e inicializarla a 0. Luego iremos sumando a esa variable un número aleatorio del 1 al 100 hasta que sumemos 1.000 o más, imprimiendo el valor de la variable suma después de cada operación

```
var suma = 0  
while (suma < 1000){  
    suma += parseInt(Math.random() * 100)  
    document.write (suma + "<br>")  
}
```

do...while

Se utiliza generalmente cuando no sabemos cuántas veces se habrá de ejecutar el bucle, igual que el bucle WHILE, con la diferencia de que sabemos seguro que el bucle por lo menos se ejecutará una vez. La sintaxis es la siguiente:

```
do {  
    //sentencias del bucle  
} while (condición)
```

La sentencia se ejecuta al menos una vez antes de que la condición sea verificada. Para múltiples sentencias, utilice un bloque de sentencias para agruparlas ({ ... }). Sí la condición es verdadera, la sentencia

se ejecuta nuevamente. Al final de cada ejecución, la condición es verificada. Cuando la condición es falsa, la ejecución pasa el control a la siguiente sentencia después de **do...while**.

En el siguiente ejemplo, iteratúa (iteraciones, repeticiones) al menos una vez y se reiteratúa hasta que **i** ya no sea menor que **5**.

```
do {  
    i += 1; document.write(i);  
} while (i < 5);
```

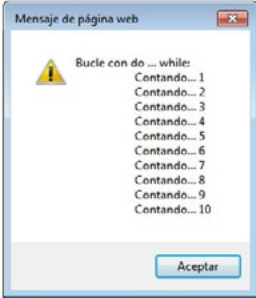
Con el siguiente código veremos, tal como lo muestra la imagen 1, cómo asignamos una función a un botón que tras pulsarlo nos lleva al desarrollo de la misma donde se implementa el **bucle do while()**

```
<!DOCTYPE HTML>
<html><head><title>Ejemplo de DO WHILE</title>
<meta char-set="utf-8">
<style type="text/css">
body {background-color:white; font-family: sans-serif;}
.boton{padding:15px; width: 200px; text-align:center; clear:both;
color: white; border-radius: 40px; background:
rgb(202, 60, 60);}
</style>
<script type="text/javascript">
function ejemploDoWhile() {
var contador = 0;var msg = "";
do { msg = msg + '\t Contando... ' + (contador+1) +
'\n';
contador +=1;
} while (contador < 10);
alert ('Bucle con do ... while: \n'+ msg);
}
</script>
</head>
<body>
<h1>Informatica 2</h1>
<h2>Estructura Do While en JavaScript</h2>
<h3 class="boton" onclick="ejemploDoWhile()">Pulsa
a </h3>
</body></html>
```

Informatica 2

Estructura Do While en JavaScript

Pulsa



Funciones en Javascript

A la hora de hacer un programa ligeramente grande existen determinados procesos que se pueden concebir de forma independiente, y que son más sencillos de resolver que el problema entero. Además, estos suelen ser realizados repetidas veces a lo largo de la ejecución del programa. Estos procesos se pueden agrupar en una función, definida para que no tengamos que repetir una y otra vez ese código en nuestros scripts, sino que simplemente llamamos a la función y ella se encarga de hacer todo lo que debe.

Así que podemos ver una función como una serie de instrucciones que englobamos dentro de un mismo proceso. Este proceso se podrá luego ejecutar desde cualquier otro sitio con

solo llamarlo. Por ejemplo, en una página web puede haber una función para cambiar el color del fondo y desde cualquier punto de la página podríamos llamarla para que nos cambie el color cuando lo deseemos.

Las funciones se utilizan constantemente, no sólo las que escribís vos, sino también las que ya están definidas en el sistema, pues todos los lenguajes de programación suelen tener un montón de funciones para realizar procesos habituales, como por ejemplo obtener la hora, imprimir un mensaje en la pantalla o convertir variables de un tipo a otro.

¿Cómo se escribe una función?

Una función se debe definir con una sintaxis especial que vamos a conocer a continuación.

```
function nombrefuncion () {  
    instrucciones de la función  
    ...  
}
```

Primero se escribe la palabra **function**, reservada para este uso. Seguidamente se escribe el nombre de la función, que como los nombres de variables puede tener números, letras y algún carácter adicional como el guión bajo. A continuación se colocan entre llaves las distintas instrucciones de la función. Las llaves en el caso de las funciones, no

son opcionales, además es útil colocarlas siempre como se ve en el ejemplo, para que se reconozca fácilmente la estructura de instrucciones que engloba la función.

Veamos un ejemplo de función para escribir en la página un mensaje de bienvenida dentro de etiquetas <H1> para que quede más resaltado.

```
function escribirBienvenida(){
  document.write("<H1>Hola a todos</H1>")
}
```

Las etiquetas H1 no se muestran en la página, sino que son interpretadas como el significado de la misma en este caso que escribimos un encabezado de nivel 1. Como estamos escribiendo en una página web, al poner etiquetas HTML se interpretan como lo que son. Como veras la sintaxis **document.write ()** te debe resultar familiar por **AED1**.

¿Cómo llamar a una función?

Para ejecutar una función la tenemos que invocar en cualquier parte de la página. Con eso conseguiremos que se ejecuten todas las

instrucciones que tiene la función entre las dos llaves. Para ejecutar la función utilizamos su nombre seguido de los paréntesis. Por ejemplo, así llamaríamos a la función `escribirBienvenida()` que acabamos de crear.

`escribirBienvenida()`

Luego veremos que existen muchas cosas adicionales que debemos conocer de las funciones, como el paso de parámetros o los valores de retorno. Pero antes vamos a explicar dónde debemos colocar las funciones Javascript.

Ámbito de una variable

Se le llama **ámbito de las variables** al lugar donde estas están disponibles.

Por lo general, cuando declaramos una variable hacemos que esté disponible en el lugar donde se ha declarado, esto ocurre en todos los lenguajes de programación y como **Javascript** se define dentro de una página web, las variables que declaremos en la página estarán accesibles dentro de ella.

Podemos considerar una función como una serie de instrucciones que englobamos dentro de un mismo proceso. Este proceso se podrá luego ejecutar desde cualquier otro sitio con solo llamarlo. Las funciones se utilizan constantemente, no sólo las que escribís vos, sino también las que ya están definidas en el sistema, pues todos los lenguajes de programación suelen tener un montón de funciones para realizar procesos habituales.



Se le llama “**ámbito de las variables**” al lugar donde estas están disponibles. Como Javascript se define dentro de una página web, las variables que declaremos en la página estarán accesibles dentro de ella.

¡Claro! es por eso que la propia página donde se define es el ámbito más habitual de una variable y le llamaremos a este tipo de variables globales a la página.



En **JavaScript** no podremos acceder a variables que hayan sido definidas en otra página. Por tanto, la propia página donde se define es el ámbito más habitual de una variable y le llamaremos a este tipo de variables globales a la página. Veremos también que se pueden hacer variables con ámbitos distintos del global, es decir, variables que declaremos y tendrán validez en lugares más acotados.

Variables globales

Como hemos dicho, las **variables globales** son las que están declaradas en el ámbito más amplio posible, que en Javascript es una página web. Para declarar una variable global

a la página simplemente lo haremos en un script, con la palabra var.

```
<SCRIPT>  
    var variableGlobal  
</SCRIPT>
```

Variables locales

También podremos declarar variables en lugares más acotados, como por ejemplo una función. A estas variables les llamaremos **locales**. Cuando se declaren variables locales sólo podremos acceder a ellas dentro del lugar donde se ha declarado, es decir, si la habíamos declarado en una función solo podremos acceder a ella cuando estemos en esa función. Las variables pueden ser

locales a una función, pero también pueden ser locales a otros ámbitos, como por ejemplo un bucle. En general, son ámbitos locales cualquier lugar acotado por llaves.

```
<SCRIPT>  
function miFuncion (){  
    var variableLocal  
}  
</SCRIPT>
```

En el ejemplo hemos declarado una variable dentro de una función, por lo que esa variable sólo tendrá validez dentro de la función. Se pueden ver cómo se utilizan las llaves para acotar el lugar donde está definida esa función o su ámbito.

No hay problema en declarar una variable local con el mismo nombre que una global, en este caso la variable global será visible desde toda la página, excepto en el ámbito donde está declarada la variable local ya que en este sitio ese nombre de variable está ocupado por la local y es ella quien tiene validez. En resumen, la variable que tendrá validez en cualquier sitio de la página es la global. Menos en el ámbito donde está declarada la variable local, que será ella quien tenga validez.

```
<SCRIPT>
var numero = 2
function miFuncion (){
  var numero = 19
  document.write(numero) //imprime 19
}
document.write(numero) //imprime 2
```

Funciones y argumentos

Podemos desear que una función ejecute unos enunciados en los que opere con una serie de valores que no hayamos definido dentro de la misma, sino que los reciba de otra función o enunciado. Esos valores son

los **argumentos**, que se especifican entre los paréntesis que van tras el nombre de la función, y se separan por comas:

```
function nombre_de_la_
función(argumento1,argumento1,...){
  ...enunciados a ejecutar...
}
```

Los argumentos se nombran como las variables:

```
function sumar(x,y){
  var total = x + y;
  alert(total);
}
```

Si después se ejecuta unas líneas como las siguientes...

```
sumar(1,2);
sumar(3,5);
sumar(8,13);
```

...la función **sumar** total adquiere sucesivamente los valores de **3, 8 y 21**, que es lo que mostrarían tres alertas.

Recordá que podés hacer variables con ámbitos distintos del global, variables que declararemos y tendrán validez en lugares más acotados. A estas les llamaremos variables locales. Cuando se declaren variables locales sólo podremos acceder a ellas dentro del lugar donde se han declarado. Las variables pueden ser locales a una función, pero también pueden ser locales a otros ámbitos, como por ejemplo un bucle.



Podemos desear que una función ejecute unos enunciados en los que opere con una serie de valores que no hayamos definido dentro de la misma, sino que los reciba de otra función o enunciado.

Esos valores ¡son los argumentos!



Funciones para cadenas de texto

A continuación se muestran algunas de las funciones más útiles para el manejo de cadenas de texto.

Función `length`

Esta función calcula la longitud de una cadena de texto, es decir, el número de caracteres que la forman.

```
var mensaje = "Hola Mundo";  
var numeroLetras = mensaje.length; //  
numeroLetras = 10
```

Función `+`

La función `+` se emplea para concatenar varias cadenas de texto

```
var mensaje1 = "Hola";  
var mensaje2 = " Mundo";  
var mensaje = mensaje1 + mensaje2; //  
mensaje = "Hola Mundo"
```

Función `concat()`

Para concatenar cadenas de texto, además del operador `+` también se puede utilizar la función **`concat()`**.

```
var mensaje1 = "Hola";  
var mensaje2 = mensaje1.concat(" Mundo"); //  
mensaje2 = "Hola Mundo"
```

Variables numéricas

Por otro lado, las cadenas de texto también se pueden unir con variables numéricas:

```
var variable1 = "Hola ";  
var variable2 = 3;  
var mensaje = variable1 + variable2; //  
mensaje = "Hola 3"
```

Cuando se unen varias cadenas de texto es habitual olvidar añadir un espacio de separación entre las palabras:

```
var mensaje1 = "Hola";  
var mensaje2 = "Mundo";  
var mensaje = mensaje1 + mensaje2; //  
mensaje = "HolaMundo"
```

Los espacios en blanco se pueden añadir al final o al principio de las cadenas y también se pueden indicar forma explícita:

```
var mensaje1 = "Hola";  
var mensaje2 = "Mundo";  
var mensaje = mensaje1 + " " + mensaje2; //  
mensaje = "Hola Mundo"
```

toUpperCase()

Esta función transforma todos los caracteres de la cadena a sus correspondientes caracteres en mayúsculas:

```
var mensaje1 = "Hola";  
var mensaje2 = mensaje1.toUpperCase(); //  
mensaje2 = "HOLA"
```

toLowerCase()

toLowerCase() transforma todos los caracteres de la cadena a sus correspondientes caracteres en minúsculas:

```
var mensaje1 = "Hola";  
var mensaje2 = mensaje1.toLowerCase(); //  
mensaje2 = "hola"
```

charAt(posicion)

charAt(posicion) obtiene el carácter que se encuentra en la posición indicada:

```
var mensaje = "Hola";  
var letra = mensaje.charAt(0); // letra = H  
letra = mensaje.charAt(2);    // letra = l
```

indexOf(caracter)

indexOf(caracter) calcula la posición en la que se encuentra el carácter indicado dentro de la cadena de texto. Si el carácter se incluye varias veces dentro de la cadena de texto, se devuelve su primera posición empezando a buscar desde la izquierda. Si la cadena no contiene el carácter, la función devuelve el valor -1:

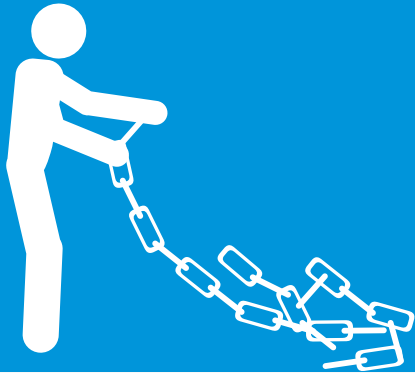
```
var mensaje = "Hola";  
var posicion = mensaje.indexOf('a'); // posicion  
= 3  
posicion = mensaje.indexOf('b');    // posicion  
= -1
```

lastIndexOf(caracter)

lastIndexOf(caracter) calcula la última posición en la que se encuentra el carácter indicado dentro de la cadena de texto. Si la

JavaScript incorpora una serie de herramientas y utilidades (llamadas funciones y propiedades) para el manejo de las variables. De esta forma, muchas de las operaciones básicas con las variables, se pueden realizar directamente con las utilidades que ofrece JavaScript.





JavaScript ofrece numerosas funciones predefinidas que facilitan el trabajo con cadenas de texto. Entre las posibilidades que ofrecen estas funciones tenemos el extraer un carácter, extraer un fragmento de cadena, separar una cadena en múltiples cadenas indicando un separador, etc.

cadena no contiene el carácter, la función devuelve el valor -1:

```
var mensaje = "Hola";  
var posicion = mensaje.lastIndexOf('a'); //  
posicion = 3  
posicion = mensaje.lastIndexOf('b');    //  
posicion = -1
```

lastIndexOf()

La función **lastIndexOf()** comienza su búsqueda desde el final de la cadena hacia el principio, aunque la posición devuelta es la correcta empezando a contar desde el principio de la palabra.

substring(inicio, final)

substring (inicio, final), extrae una porción de una cadena de texto. El segundo parámetro es opcional. Si sólo se indica el parámetro inicio, la función devuelve la parte de la cadena original correspondiente desde esa posición hasta el final:

```
var mensaje = "Hola Mundo";  
var porcion = mensaje.substring(2); // porcion  
= "la Mundo"  
porcion = mensaje.substring(5);    // porcion =  
"Mundo"  
porcion = mensaje.substring(7);    // porcion =  
"ndo"
```

Si se indica un **inicio** negativo, se devuelve la misma cadena original:

```
var mensaje = "Hola Mundo";  
var porcion = mensaje.substring(-2); // porcion  
= "Hola Mundo"
```

Cuando se indica el **inicio** y el **final**, se devuelve la parte de la cadena original comprendida entre la posición inicial y la inmediatamente anterior a la posición final (es decir, la posición **inicio** está incluida y la posición **final** no):

```
var mensaje = "Hola Mundo";  
var porcion = mensaje.substring(1, 8); //  
porcion = "ola Mun"  
porcion = mensaje.substring(3, 4);    // porcion  
= "a"
```


Si se indica un **final** más pequeño que el **inicio**, JavaScript los considera de forma inversa, ya que automáticamente asigna el valor más pequeño al **inicio** y el más grande al **final**:

```
var mensaje = "Hola Mundo";  
var porcion = mensaje.substring(5, 0); //  
porcion = "Hola "  
porcion = mensaje.substring(0, 5); // porcion  
= "Hola "
```

split(separador)

La función **split(separador)** convierte una cadena de texto en un **array** de cadenas de texto. La función parte la cadena de texto determinando sus trozos a partir del carácter **separador** indicado:

```
var mensaje = "Hola Mundo, soy una cadena  
de texto!";  
var palabras = mensaje.split(" ");  
// palabras = ["Hola", "Mundo,", "soy", "una",  
"cadena", "de", "texto!"];
```

Con esta función se pueden extraer fácilmente las letras que forman una palabra:

```
var palabra = "Hola";  
var letras = palabra.split(""); // letras = ["H", "o",  
"l", "a"]
```

Funciones para arrays

A continuación te vamos a presentar algunas de las funciones más útiles para el manejo de arrays.

length

La función **length**, calcula el número de elementos de un array

```
var vocales = ["a", "e", "i", "o", "u"];  
var numeroVocales = vocales.length; //  
numeroVocales = 5
```

concat()

La función **concat()**, se emplea para concatenar los elementos de varios arrays

```
var array1 = [1, 2, 3];  
array2 = array1.concat(4, 5, 6); // array2 = [1,  
2, 3, 4, 5, 6]  
array3 = array1.concat([4, 5, 6]); // array3 = [1,  
2, 3, 4, 5, 6]
```

JavaScript proporciona una serie de funciones predefinidas para el manejo de arrays. Entre ellas podemos citar length, concat, join, pop, push, shift, unshift, reverse.

¡Vamos a estudiarlas!



* ! #



length

concat()

join(separador)

pop()

push()

shift()

unshift()

reverse()

join(separador)

La función **join(separador)** es la contraria a **split()**. Une todos los elementos de un array para formar una cadena de texto. Para unir los elementos se utiliza el carácter separador indicado

```
var array = ["hola", "mundo"];  
var mensaje = array.join(""); // mensaje =  
"holamundo"  
mensaje = array.join(" "); // mensaje = "hola  
mundo"
```

pop()

La función **pop()**, elimina el último elemento del array y lo devuelve. El array original se modifica y su longitud disminuye en 1 elemento.

```
var array = [1, 2, 3];  
var ultimo = array.pop();// ahora array = [1, 2],  
ultimo = 3
```

push()

La función **push()**, añade un elemento al final del array. El array original se modifica y aumenta su longitud en 1 elemento. (También es posible añadir más de un elemento a la vez)

```
var array = [1, 2, 3];  
array.push(4); // ahora array = [1, 2, 3, 4]
```

shift()

La función **shift()** elimina el primer elemento del array y lo devuelve. El array original se ve modificado y su longitud disminuida en 1 elemento.

```
var array = [1, 2, 3];  
var primero = array.shift(); // ahora array = [2,  
3], primero = 1
```

unshift()

La función **unshift()** añade un elemento al principio del array. El array original se modifica y aumenta su longitud en 1 elemento. También es posible añadir más de un elemento a la vez.

```
var array = [1, 2, 3];  
array.unshift(0); // ahora array = [0, 1, 2, 3]
```

reverse()

La función **reverse()**, modifica un array colocando sus elementos en el orden inverso a su posición original:

```
var array = [1, 2, 3];
array.reverse(); // ahora array = [3, 2, 1]
```

Funciones números

A continuación se muestran algunas de las funciones y propiedades más útiles para el manejo de números.

NaN

Las siglas **NaN** provienen del inglés y significan **"Not a Number"**. JavaScript emplea el valor **NaN** para indicar un valor numérico no definido (por ejemplo, la división 0/0).

```
var numero1 = 0;
var numero2 = 0;
alert(numero1/numero2); // se muestra el valor NaN
```

isNaN()

Esta función permite proteger a la aplicación de posibles valores numéricos no definidos

```
var numero1 = 0;
var numero2 = 0;
if(isNaN(numero1/numero2)) {
    alert("La división no está definida para los
```

```
números indicados");
}else {
    alert("La división es igual a => " +
numero1/numero2);
}
```

Infinity

Infinity, hace referencia a un valor numérico infinito y positivo (también existe el valor **-Infinity** para los infinitos negativos)

```
var numero1 = 10;
var numero2 = 0;
alert(numero1/numero2); // se muestra el valor Infinity
```

toFixed(dígitos)

toFixed(dígitos) devuelve el número original con tantos decimales como los indicados por el parámetro **dígitos** y realiza los redondeos necesarios. Se trata de una función muy útil por ejemplo para mostrar precios.

```
var numero1 = 4564.34567;
numero1.toFixed(2); // 4564.35
numero1.toFixed(6); // 4564.345670
numero1.toFixed(); // 4564
```

En JavaScript existen funciones y propiedades que son muy útiles para el manejo de números. Las más importantes son NaN, isNaN(), Infinity y toFixed (dígitos). ¡Vamos a aprenderlas!



| | |
|--------------------|--|
| <i>onclick</i> | Al dar un click con el botón izquierdo del mouse sobre un elemento |
| <i>ondblclick</i> | Dar doble click sobre un elemento |
| <i>onmousedown</i> | Cualquier botón del mouse es presionado sobre un elemento |
| <i>onmouseup</i> | Se libera un botón del mouse sobre un elemento |
| <i>onmouseover</i> | Solo situar el ratón sobre un elemento |
| <i>onmousemove</i> | Mover el mouse |
| <i>onmouseout</i> | Es retirado el ratón de un elemento |

Eventos

Eventos del Mouse

Los eventos del mouse que se muestran en el cuadro de arriba (1) reaccionan a diferentes acciones del usuario y pueden ser insertados en cualquier elemento HTML de bloque.

Estos son algunos ejemplos de códigos que se pueden emplear en cualquiera de los eventos de ratón, es posible insertarlos en cualquier elemento de bloque HTML.

```
<a href="#" onclick="FUNCIÓN">VÍNCULO</a>
<div onclick="FUNCIÓN">TEXTO</div>
<h2 onclick="FUNCIÓN">Encabezado</h2>
<pre onclick="FUNCIÓN">Código</pre>
```

Los eventos del mouse pueden utilizarse para la creación de una infinidad de efectos, desde diferentes estilos del puntero, mensajes, menús personalizados para el clic derecho, etc., pero no se deben usar de forma indiscriminada ya que pueden hacer saltar molestas alertas, principalmente con el evento onmouseover. En los vínculos se usa a menudo el evento onclick para mostrar un cuadro de dialogo con un mensaje de información o una confirmación, antes de enviar al usuario al destino solicitado, por ejemplo, el primer vínculo solo muestra un mensaje, el segundo ofrece información y da la alternativa de permanecer en la misma página:

```
<a href="http://google.com"
onclick="alert('Este link te dirige a Google.
com')">Google.com</a>

<a href="#" onclick="javascript:if(confirm('Este link te dirige a Google.com'))
{pa-rent.location='http://google.com'}
else{''}>Google</a>
```

Eventos del teclado

Los siguientes eventos de teclado se pueden insertar en casi cualquier elemento HTML.

| | |
|-------------------|--|
| onkeypress | Presionar una tecla del teclado y soltarla |
| onkeydown | Presionar una tecla del teclado y soltarla |
| onkeyup | Liberar una tecla |

| |
|-------------------|
| onkeypress |
| onkeydown |
| onkeyup |

(1)

```
<input type="text" value="onkeypress"
onkeypress="alert('Haz soltado una tecla.')">
<input type="text" value="onkeydown"
onkeydown="alert('Haz presionado una
tecla.')">
<input type="text" value="onkeyup"
onkeyup="alert('Haz liberado una tecla.')">
```

Eventos onfocus y onblur

Los eventos **onfocus** y **onblur** se utilizan en los elementos HTML: **a**, **textarea**, **area**, **button**, **label**, **input** y **select** para desencadenar una acción cuando tengan el focus (es decir, la atención y reciben lo que se escriba), o cuando lo pierdan.

| **onfocus** Un elemento obtiene el focus

| **onblur** Un elemento pierde el focus

```
<input type="text" value=""
onfocus="value='Tengo el focus';style.
backgroundColor='yellow';"
onblur="value='Perdi el focus';style.
backgroundColor='#ddd';">
```

Eventos onload y onunload

Los eventos **onload** y **onunload** se utilizan para definir cuando una página o imagen terminan de cargar completamente y cuando el usuario abandona la página respectivamente. Se pueden usar solo en las etiquetas **body** e **img** y en el objeto **window**

| **Onload**: Una página o imagen termina de cargar

| **Onunload**: El usuario abandona la página

El siguiente código permite mostrar un mensaje de bienvenida cuando el usuario carga la página y muestra otro de despedida cuando la abandona.

```
<body onload="saludo()"
onunload="despedida()">
```

```
<script type="text/javascript">
function saludo() {alert('Bienvenido a la página
de Javascript')}
function despedida() {alert('Gracias por tu
visita')}
</script>
```

Los eventos onload y onunload se pueden insertar de dos formas:

1 | Insertándolo en la etiqueta <body>:

```
<body onload="funcion()">
```

2 | En un script:

```
<script type="text/javascript">
window.onunload=funcion();
</script>
```

Importancia y usos prácticos de Onload

Onload es ampliamente utilizado en las páginas de internet por dos motivos:

1 | Para cargar elementos pesados como imágenes, flash o videos, pero solo después que la página se muestre completamente en el navegador, de esa forma se hace las páginas muy rápidas.

2 | Para cargar elementos que se desea que no sean rastreados por los buscadores, como iframes con publicidad (marcos), vínculos a sitios de afiliados y otros por alguna razón. De

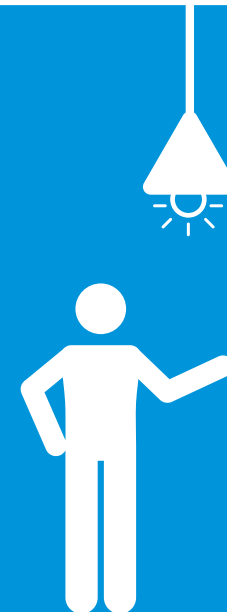
esa forma estos elementos son escritos solo después que la página concluya de cargar completamente.

Con el siguiente script se escribe un vínculo en una página usando Onload. El contenedor se puede usar en la zona exacta de la página donde se necesita escribir el link.

```
<div id="lk"></div>

<script type="text/javascript">
function esclinks(){
document.getElementById('lk').innerHTML='<a
href="http://sitioweb">Entra a mi sitio</a>'; }
window.onload=function(){
esclinks();}
</script>
```

Observá como el nombre de cada evento se construye mediante el prefijo on, seguido del nombre en inglés de la acción asociada al evento. Así, por ejemplo, el evento de pinchar un elemento con el ratón se denomina onclick y el evento asociado a la acción de mover el ratón se denomina onmousemove. ¡Tal vez esto te ayude a recordar los nombres!



Evento onselect

El evento onselect permite realizar una acción cuando el usuario seleccione algún texto, se puede utilizar en los elementos HTML `<input type="text">` y `<textarea>`. Código usado en el ejemplo:

```
<input type="text" value="onselect"
onselect="alert('Haz seleccionado texto')">
```

Evento onchange

El evento **onchange** ejecuta una acción al cambiar el contenido de una entrada de texto, puede usarse en las etiquetas `<input type="text">`, `<select>` y `<textarea>`

```
<input type="text" value="onchange" on
change="style.backgroundColor='red';">
```

Existen mas eventos pero los iremos viendo y comentando a lo largo de los ejemplos y ejercicios expuestos

Cuadros de alertas

Cuando utilizamos alertas, se abre una pequeña ventana o cuadro de dialogo donde aparece el resultado de la variable y el texto que se adicione. Para crear una alerta podemos recurrir a diferentes métodos. A continuación los abordaremos.

Mediante un link

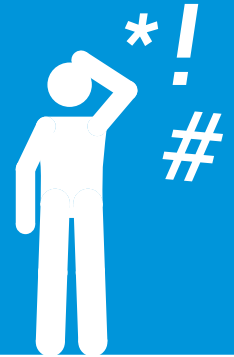
Para construir un alerta mediante un link debes utilizar el siguiente código:

```
<a href="javascript:alert(VARIABLE);">Nombre
Link</a>
<a href="javascript:alert('texto'+VARIABLE);">N
ombreLink</a>
<a href="javascript:alert('texto'+VARIABLE+'tex
to');">NombreLink</a>
```

Para impedir que al presionar el link el navegador salga de la página actual, se utiliza `void 0` al final, por ejemplo:

```
<a href="javascript:alert(VARIABLE);void
0">NombreLink</a>
```

A veces es necesario mostrar mensajes de aviso al usuario, ya sea para informarle o advertirle algo, pero... ¿cómo hacerlo?



Para crear alertas podés recurrir a tres métodos diferentes: podés crearlo mediante un link, a través de un botón o por eventos.



Mediante un botón

Para construir un alerta mediante un botón debes utilizar el siguiente código:

```
<input type="button" value="NombreBoton"
on-click="alert(VARIABLE);" />
<input type="button" value="NombreBoton" on
click="alert('texto'+VARIABLE);" />
<input type="button" value="NombreBoton" on
click="alert('texto'+VARIABLE+'texto');" />
```

Mediante eventos

De forma similar se pueden ejecutar funciones mediante cualquier evento (acciones que pueden ser detectadas por Javascript). El más utilizado de ellos es **onclick**, pero existen varios, todos ellos se pueden utilizar en cualquier elemento HTML.

Algunos ejemplos:

```
<a href="javascript:FUNCIÓN;">NombreLink</a>
<input type="button" value="NombreBoton"
onclick="FUNCIÓN;" />
<div onclick="FUNCIÓN;">Texto</div>
```

Objetos

Cada vez que se carga una página en el navegador, el intérprete de JavaScript crea automáticamente una serie de objetos que pueden ser usados al programar en JavaScript. Entre los distintos objetos generados automáticamente podemos destacar los siguientes:

| **Window**: es el objeto principal del cual "cuelgan" los demás como si fueran propiedades suyas. Se crea un objeto de este tipo para cada ventana que pueda ser lanzada desde una página Web.

| **Navigator**: contiene información acerca del navegador, tal como nombre, versión, tipos MIME soportados por el cliente y plugs-in instalados.

| **Location**: con información acerca del URL que estemos visualizando.

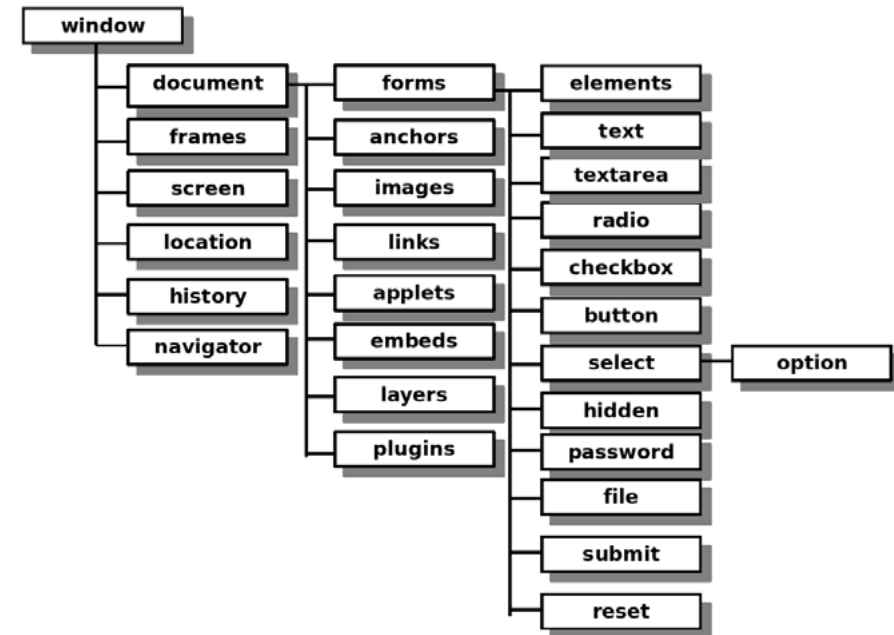
| **History**: historial en el que se guardan los URL ya visitados.

| **Document**: es el contenedor de todos los objetos que se hayan insertado en la página web: enlaces, formularios, imágenes o marcos.

Podemos ver una página web como una colección de objetos. Por ejemplo, para JavaScript un formulario es un objeto, una imagen es un objeto, etc. Los objetos tienen propiedades, métodos y eventos asociados y se organizan conforme a una jerarquía que veremos enseguida.

| **Frames:** Matriz conteniendo los distintos objetos frame que puede contener una ventana. Cada frame es a su vez otra ventana.

El cuadro (1) muestra la jerarquía de objetos creados por el navegador al cargar una página Web:



Document Object

Al ser cargada una página web por el navegador, Windows la considera como un objeto, por lo que las siguientes variables devuelven información como si la página fuera un objeto.

| | |
|--------------------------|---|
| document.title | Muestra el título de la página actual |
| document.URL | Muestra la dirección URL de la página actual |
| document.referrer | Muestra la dirección URL de la página que dirigió a la actual |
| document.lastModified | Fecha de la última modificación de la página |
| document.domain | Muestra el nombre del domino del sitio web |
| document.cookie | Muestra las cookies guardadas por este sitio web en tu equipo |
| document.links.length | Muestra el número de links en la página |
| document.links[0]. | Muestra el nombre del primer link |
| innerHTML | Muestra el nombre del segundo link |
| document.links[1]. | Número de anchors en la página |
| innerHTML | Número de formas en la página |
| document.anchors.length | Muestra el nombre de la primera forma |
| document.forms.length | Número de imágenes en la página |
| document.forms[0].name | Muestra la identidad (ID) de la primera imagen |
| document.images.length | Permite identificar un elemento en una página por su identidad (ID), |
| document.images[0].id | entonces ejecuta una acción, en este caso se emplea: |
| document.etElementById() | Document.getElementById('test').innerHTML='Hola';void 0 (test es un DIV o contenedor con dicha identidad en la página, innerHTML='Hola' escribe el texto indicado y void 0 se utiliza para evitar que el navegador cargue una nueva página). |
| document.write() | Escribe texto, código o el resultado de una variable en una página, se usa de las siguientes formas: document.write('texto') document.write(variable) document.write('texto'+variable) document.write('texto'+variable+'texto') |

| | |
|--------------------------------------|--|
| <code>navigator.appCodeName</code> | Devuelve el código del nombre del navegador web con que se carga la página |
| <code>navigator.appName</code> | Devuelve el nombre del navegador |
| <code>navigator.appVersion</code> | Versión del navegador |
| <code>navigator.cookieEnabled</code> | Comprueba si están habilitadas las cookies en el navegador (true=Si, false=No) |
| <code>navigator.platform</code> | Plataforma del navegador |
| <code>navigator.userAgent</code> | Agente de usuario enviado por el navegador al servidor |
| <code>navigator.javaEnabled</code> | Se comprueba si está habilitada Java en el navegador (true=Si, false=No) |

```
Navigator appCodeName: Mozilla
Navigator appName: Netscape
Navigator appVersion: 5.0 (Windows NT 6.1; WOW64; Trident/7.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0E; .NET4.0C; InfoPath.3; rv:11.0) like Gecko
Navigator language: es-ES
Navigator platform: Win32
Navigator userAgent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0E; .NET4.0C; InfoPath.3; rv:11.0) like Gecko
```

Navigator Object

Las variables que podemos observar arriba **(1)** devuelven información del navegador usado para cargar la página.

Un ejemplo de ello podés observarlo en el siguiente código y en la imagen **2**.

```
<HTML>
```

```
<HEAD>
```

```
<title>Ejemplo de JavaScript</title>
```

```
</HEAD>
```

```
<BODY>
```

```
<script LANGUAGE="JavaScript">
```

```
<!--
```

```
document.write("Navigator
<b>appCodeName</b>: " + navigator.
appCodeName + "<br>");
```

```
document.write("Navigator <b>appName</
b>: " + navigator.appName + "<br>");
```

```
document.write("Navigator <b>appVersion</
b>: " + navigator.appVersion + "<br>");
```

```
document.write("Navigator <b>language</
b>: " + navigator.language + "<br>");
document.write("Navigator <b>platforma</
b>: " + navigator.platform + "<br>");
document.write("Navigator <b>userAgent</
b>: " + navigator.userAgent + "<br>");
```

```
//-->
```

```
</script>
```

```
</BODY>
```

```
</HTML>
```

Location Object

| | |
|--------------------------|--|
| <i>location.host</i> | Devuelve el nombre del host de una dirección web |
| <i>location.hostname</i> | Similar al anterior |
| <i>location.href</i> | Devuelve la dirección URL completa |
| <i>location.pathname</i> | Devuelve solo la ruta relativa en el servidor a la pagina |
| <i>location.port</i> | Devuelve el número del puerto usado |
| <i>location.protocol</i> | Muestra el protocolo usado (http, https, file, ftp, etc.) |
| <i>location.reload()</i> | Vuelve a cargar la página (window.location.reload(true)) |
| <i>location.href</i> | Encadenando location.href se refresca el contenido de la página (vuelve a cargarla), en este caso se usa: location.href=location.href |

History Object

| | |
|--------------------------|--|
| <i>history.length</i> | Devuelve la cantidad de direcciones URL en la lista del historial |
| <i>history.back()</i> | Página anterior en el historial |
| <i>history.forward()</i> | Página siguiente en el historial |
| <i>history.go()</i> | Carga una página determinada del historial, en este caso se emplea: history.go(0) por lo que recarga la página actual, similar a usar la tecla F5 |

Screen Object

| | |
|---------------------------|---|
| <i>screen.height</i> | Devuelve la altura total de la pantalla en pixeles |
| <i>screen.width</i> | Devuelve el ancho total de la pantalla en pixeles |
| <i>screen.availHeight</i> | Devuelve la altura de la pantalla disponible en pixeles |
| <i>screen.availWidth</i> | Devuelve el ancho de la pantalla disponible en pixeles |
| <i>screen.colorDepth</i> | Profundidad de color de la pantalla para mostrar imágenes |
| <i>screen.pixelDepth</i> | Resolución del color en bits por pixel de la pantalla |

Podemos observar un ejemplo a través del siguiente código y su imagen correspondiente (1).

```
<HTML>
<HEAD>
  <title>Ejemplo de JavaScript</title>
</HEAD>
<script LANGUAGE="JavaScript">
<!--
  img1 = new Image();
  img1.src = "foto1.jpg";
  img2 = new Image();
  img2.src = "foto2.jpg";
  function cambia(nombre,imagen)
  {
    nombre.src = imagen.src
  }
  function dobleancho()
  {
    imagen1.width=imagen1.width*2;
  }
  function doblealto()
  {
    imagen1.height=imagen1.height*2;
  }
  function mitadancho()
```

```
{
  imagen1.width=imagen1.width/2;
}
function mitadalto()
{
  imagen1.height=imagen1.height/2;
}
//-->
</script>
<BODY>
<a href="" onmouseover="cambia(imagen1,imagen2)" onmouseout="cambia(imagen1,img2)">
</a><br><br>
<a href="javascript:dobleancho()">Doble ancho</a><br>
<a href="javascript:doblealto()">Doble Alto</a><br>
<a href="javascript:mitadanch()">Mitad ancho</a><br>
<a href="javascript:mitadalto()">Mitad Alto</a><br>
</BODY>
</HTML>
```



Doble ancho
Doble Alto
Mitad ancho
Mitad Alto



Doble ancho
Doble Alto
Mitad ancho
Mitad Alto

Window Object ▼

| | |
|------------------------|---|
| window.onload | Ejecuta una función o varias inmediatamente después de que termine la carga de la página por completo, en este caso se emplea una alerta que se muestra al entrar a la página. El código usado es: <script type="text/javascript"> window.onload=alert('Hola, Bienvenido a Javascript') </script> Este código se debe insertar en el final de la página, justo antes del cierre de la etiqueta </body> |
| window.parent.location | Devuelve la dirección URL de la página actual |
| window.parent.location | Conduce a una dirección web, en este ejemplo: window.parent.location='http://norfipc.com' |
| window.alert() | Muestra una ventana de alerta con un mensaje: window.alert('Mensaje') |
| window.confirm() | Confirmación, muestra un cuadro de dialogo con un mensaje, un botón Aceptar y uno Cancelar: window.confirm('Desea?....') Generalmente se usa encadenada con otra función |
| window.prompt() | Muestra un cuadro de dialogo que inquiere al usuario por una respuesta, se usa: window.prompt('Mensaje','Quiere..?') Al igual que la anterior se usa con una función que se ejecuta en caso del usuario oprimir el botón Aceptar |
| window.open() | Abre una nueva ventana o pestaña según como se emplee, ninguna de las variantes funciona en el navegador Internet Explorer por cuestiones de seguridad. En este ejemplo se abre una nueva ventana llamada "new", para eso se emplea el siguiente código: nv=window.open('','new','width=344,height=444,left=50,top=50') |
| close() | Cierra una ventana, en este ejemplo cierra la ventana abierta anteriormente: nv.close() |

Window Object

| | |
|--------------------|---|
| window.open() | En este ejemplo se emplea <i>window.open()</i> para abrir una nueva pestaña |
| close() | Cierra la pestaña abierta anteriormente |
| top.close() | Cerrar pestaña |
| window.name | Muestra el nombre de una ventana, se emplea: nv=window.open('','new','width=344,height=444'); nv.document. write('Esta ventana se llama: ' + nv.name) |
| window.innerHeight | Muestra la altura en pixeles del tamaño de la ventana del navegador (No es compatible con Internet Explorer) |
| window.innerWidth | Muestra el ancho en pixeles del tamaño de la ventana del navegador (No es compatible con Internet Explorer) |
| window.print() | Imprime el contenido de la ventana. Más información sobre las opciones para utilizar window.print(), puedes leerla en la siguiente página: Cómo imprimir solo un área, parte o sección de una página web |
| window.resizeBy() | Este método cambia de tamaño la ventana del navegador, mueve la esquina derecha inferior de la ventana, la cantidad de pixeles que se determine, ya sea positiva un incremento o negativa una reducción, la esquina superior izquierda permanecerá inmóvil. En este ejemplo se emplea: window.resizeBy(100,100) lo que incrementa en 100 pixeles su tamaño en cada eje (no funciona en Opera y Chrome). |
| window.resizeTo() | Ajusta el tamaño de la ventana al número de pixeles que se determine, en este ejemplo se emplea: window.resizeTo(1000,800) |

| | |
|-------------------|--|
| window.scrollBy() | <p>Desplaza el contenido de la ventana en un determinado número de pixeles, hacia arriba o hacia abajo según se establezca de forma positiva o negativa, en este ejemplo se emplea:</p> <p>window.scrollBy(50,-50)</p> <p>Combinando la function <i>window.scrollBy</i> con <i>setTimeout</i> se puede hacer que la página vaya desplazándose continuamente, se puede iniciar la función al cargar la página mediante el evento <i>onload</i> o con un vínculo de la siguiente forma:</p> <pre><script type="text/javascript"> function pageScroll() {window.scrollBy(0,30); scrollDelay = setTimeout('pageScroll()',800);} function stopScroll() {clearTimeout(scrollDelay);} </script></pre> <p>Scroll STOP <u>Scroll STOP</u></p> |
| window.scrollTo() | <p>Desplaza el contenido de la ventana a unas coordenadas específicas, en este caso se usa:</p> <p>window.scrollTo(750,650)</p> |
| window.moveTo() | <p>Mueve la ventana del navegador a una posición específica en la pantalla definida en pixeles, en este caso:</p> <p>window.moveTo(222,222)</p> |

Window Object

| | |
|---|---|
| window.moveBy() | Mueve la ventana del navegador a una ubicación en relación con su posición actual determinada en pixeles, en este caso: window.moveBy(200,300) |
| window.screenLeft | Muestra el número de pixeles distantes a la pantalla, del borde izquierdo de la ventana al borde izquierdo de la pantalla (Solo Internet Explorer) |
| window.screenTop | Muestra el número de pixeles distantes a la pantalla, del borde superior de la ventana al borde superior de la pantalla (Solo Internet Explorer) |
| window.screenX | Muestra el número de pixeles distantes a la pantalla, del borde izquierdo de la ventana al borde izquierdo de la pantalla (Firefox y Google Chrome) |
| window.screenY | Muestra el número de pixeles distantes a la pantalla, del borde superior de la ventana al borde superior de la pantalla (Firefox y Google Chrome) |
| document. documentElement. clientHeight | Muestra la altura en pixeles del tamaño de la ventana del navegador (Todos los navegadores) |
| document. documentElement. clientWidth | Muestra el ancho en pixeles del tamaño de la ventana del navegador (Todos los navegadores) |

Veamos ahora un par de ejemplos...

Ejemplo 01

```
<HTML>
<HEAD>
<TITLE>Abriendo y cerrando ventanas con
Javascript</TITLE>
<SCRIPT>
<!--
var nuevaVentana;

function doble( x ) {
    return 2*x;
}

function nueva() {
    nuevaVentana=nuevaVentana=window.
open("", "segundaPag",
    "toolbar=yes,location=no,menubar=yes,resi
zable=yes,"+
    "width=30,height=30" );
    nuevaVentana.document.
write("<HTML><HEAD><TITLE>" +
    "Sin Titulo</TITLE></HEAD>\n");
    nuevaVentana.document.
write("<BODY><form>\n");
    nuevaVentana.document.write("<input
type='button' "+
    "value='Cerrar' onClick='window.
close();>\n");

    nuevaVentana.document.write("</
form>\n");
    nuevaVentana.document.write("</BODY></
HTML>\n");
    nuevaVentana.document.close();
}
// -->
</SCRIPT>
<BODY>
<FORM>
    <INPUT TYPE="button" VALUE="Abrir"
onClick="nueva();"><br>
</FORM>
</BODY>
</HTML>
```


Ejemplo 02 ▶

Mensajes de error

En ocasiones al solicitar el resultado de una variable, Javascript devuelve los siguientes mensajes predeterminados:

| **NaN**: "Not-a-Number", significa que el resultado no es un número.

| **Undefined**: Indica que a la variable no se le ha asignado ningún valor.

| **Infinity**: Significa un valor imposible de representar.

Referencia a objetos con This

JavaScript posee una palabra reservada especial: **this**. Esta se puede utilizar dentro de un método para referirse al objeto actual. Por ejemplo, supongamos que tenemos una función llamada **validar**, que valida un valor de una propiedad del objeto. Dado el objeto y sus valores máximo y mínimo:

```
<HTML>
<HEAD>
<TITLE>Ejemplo de uso de prompt</TITLE>
<SCRIPT>
var cadena=prompt( "Escribe tu nombre");
document.write ("Esta pagina esta optimizada para "+cadena );
</SCRIPT>
</HEAD>
<BODY>
Hola,
<SCRIPT>
document.write(cadena+"<p>");
</SCRIPT>
</BODY>
</HTML>
```



A veces, al solicitar el resultado de una variable Javascript te puede salir un MENSAJE DE ERROR



NaN o **Not-a-Number**, significa que el resultado no es un número.



Undefined te indica que a la variable no se le ha asignado ningún valor.



Infinity significa un valor imposible de representar.

```
function validar(objeto, val_min, val_max) {
  if ((objeto.value < val_min) || (objeto.value >
val_max))
    alert("Valor inválido!");
}
```

Entonces, puede invocar a validar en cada manejador del evento de cambio **[onchange]** de los elementos del formulario, utilizando **this** para pasar a este el elemento **form**, tal como el siguiente ejemplo:

```
<input type="text" name="edad" size="3"
onChange="validar(this, 18, 99)">
```

Cuando se combina con la propiedad **form**, **this** puede referirse al objeto actual del formulario padre. En el siguiente ejemplo, el formulario **miFormulario** contiene el objeto **Text** y un **botón**. Cuando el usuario hace clic en el botón, el valor del objeto **Text** es configurado con el nombre del formulario. El manejador del evento **onclick** del botón utiliza **this.form** para referirse al formulario padre, **miFormulario**.

```
<form name="miFormulario">
<p><label>Nombre del formulario:<input
type="text" name="texto1" value="Nombre"></
label>
<p><input name="botón1" type="button"
value="Muéstrame el nombre del
formulario"onclick="this.form.text1.value=this.
form.name">
</p>
</form>
```

¿Qué pasa cuando this se combina con la propiedad form?



Cuando se combina con la propiedad form, this puede referirse al objeto actual del formulario padre.



→ Desempeños

→ Desempeño 1

→ Desempeño 2

Realizá una página para calcular los números primos entre 1 y 100.

Realizá una página que muestre un formulario para la conversión de Euros a Pesos o viceversa como podés ver en la imagen. Los campos del formulario han de poder ser limpiados.

Podés valerte de esta ayuda para que te sea más fácil el primer desempeño

```
primo=0;
for(j=2;j<i;j++) { if(i%j==0)
primo=1; }
if (primo==0) {document.
write(i); document.
write('<br>');}
```

Conversión de Pesos y Euros

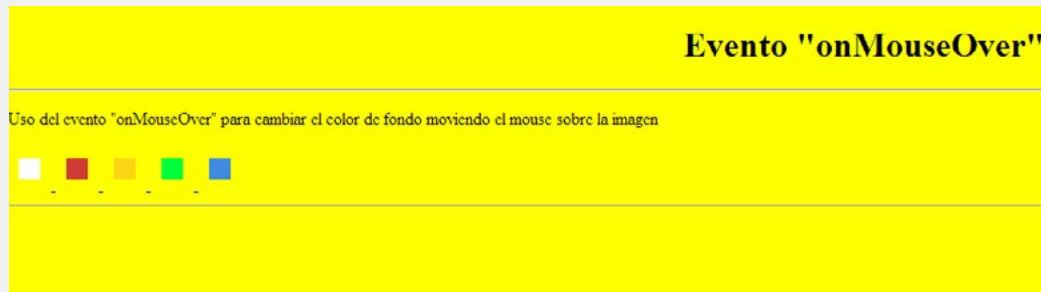
Pesos: Euros:

Euros: Pesos:

Utilizá para este ejercicio el siguiente evento asociado a un botón:
onclick="if(!isNaN(this.form.pesetas.value)) this.form.c_euros.value=this.form.pesetas.value/10.85; else alert('No es un numero');">

→ Desempeño 3

Elaborá una página que muestre cinco cuadrados de diferentes colores. Cuando el ratón pase por encima de alguno de ellos el color de fondo será del color de relleno del cuadrado. A esto podés verlo en la siguiente imagen.



Luego en el HTML

`<P>Uso del evento "onMouseOver" para cambiar el color de fondo
moviendo el mouse sobre la imagen</P>`

```
<A onmouseover="changecolor('ffffff')" href="#"
<IMG height=20 hspace=10 src="blanco.jpg" width=20 vspace=10>
</A>
```

Podés utilizar este código como ayuda para el desempeño 3.

```
function changecolor(code) { document.  
bgColor=code }
```



→ Desempeño 4

Programá las acciones **adelante**, **atrás** y **actualizar** tal como podés ver en la imagen . Los enlaces se pueden presentar en tres versiones:

- 1 | Enlace de texto;
- 2 | Enlace con imagen; y
- 3 | Enlace con botón de formulario



Utilizá para ello:
javascript:history.go(-1), **javascript:history.go(1)**,
javascript:history.go(0)

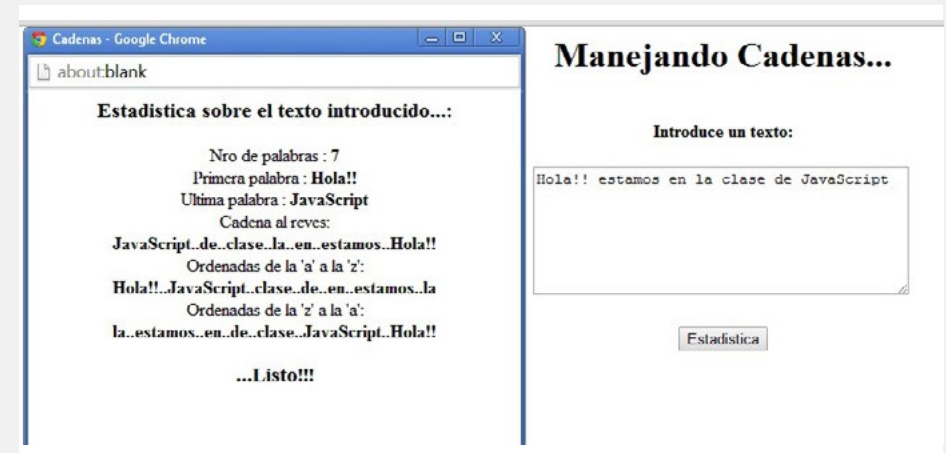
→ Desempeño 5

Ingresa una **cadena de texto** en un formulario y generará un **array** con la función **split()**. Posteriormente, mostrará la siguiente información: número de palabras, primera palabra, última palabra, las palabras colocadas en orden inverso, las palabras ordenadas de la **a** la **z** y las palabras ordenadas de la **z** a la **a**. Sacar toda esta información en una ventana nueva.

Para poder hacer el desempeño 5 utilizará esta ayuda.

```
function tratarTexto() {  
    var array_palabras = new Array();  
    var nueva_ventana;  
    var num_palabras;  
  
    array_palabras=miFormulario.  
miArea.value.split(" ");  
    num_palabras=array_palabras.  
length;  
    .....  
}
```

El resultado debe quedar algo semejante a la imagen...



Desempeño 6

Realizá una página que efectúe **test de evaluación** a través de **radio button** de tres opciones. La página prporcionará la evaluación y las respuesta correctas a petición del usuario. Las imágenes 1 y 2 te servirán para ver cómo debés maquetar los ejercicios. En el caso de la imagen 1 verás una pantalla de bienvenida al test con un mensaje **Alert**, a su vez este mensaje aparecerá cuando

se pulse el botón **Menu principal**. Por su parte, en la imagen 2, podés observar que si se pulsa el **Boton resultado** se muestran, mediante un Alert, mensajes que indican porcentajes a preguntas correctas, incorrectas etc. Además puede contener botones adicionales tales como Reiniciar test, ver código fuente etc.

1

1. Cuantas rectas

☐ A. Infinitas

☐ B. Dos.

☐ C. Dependiendo de la figura

Bienvenido al test de conocimientos matematicos

Seleccione las respuestas que crea correctas teniendo en cuenta las siguientes consideraciones:

1.- Los aciertos tienen puntuaciones variables en funcion de la dificultad de la pregunta.

2.- Las preguntas no contestadas ni suman ni restan puntos.

3.- Las respuestas equivocadas restan mas o menos puntos en funcion del tipo de fallo.

NOTA:
El test que va a realizar a continuacion carece de rigor 'cientifico'. Unicamente se utiliza para mostrar una posibilidad mas de la programacion en JavaScript.

GRACIAS

Aceptar

2. Por un punto de

☐ A. Ninguno

☐ B. Uno.

☐ C. Infinitos

3. Determine el r

☐ A. $x = 29$

☐ B. $x = 27$

☐ C. $x = 16$

Resultado del test

Reiniciar test

Respuestas correctas

Menu Principal

Salir

Codigo Fuente

2

1. Cuantas rectas

☒ A. Infinitas

☐ B. Dos.

☐ C. Dependiendo de la figura

Alerta de JavaScript

Su puntuacion es 5.

El numero maximo de puntos que podia conseguir era de 5.

Ha dejado sin contestar 0.

Su porcentaje de aciertos es de 100%.

SOBRESALIENTE. Sus conocimientos de matematicas son excelentes.

Aceptar

2. Por un punto de

☐ A. Ninguno

☒ B. Uno.

☐ C. Infinitos

3. Determine el resultado de la siguiente ecuacion: $7x - 28 = 6x - 1$

☐ A. $x = 29$.

☒ B. $x = 27$.

☐ C. $x = 16$.

Resultado del test

Reiniciar test

Respuestas correctas

Autoevaluación

Ahora intentá responder a estas preguntas. Recordá que una buena autoevaluación es clave para darte cuenta cuáles son los puntos que todavía debés repasar.

1 | ¿Cómo se acceden a las Cookies desde Java?

2 | ¿Qué tipo de lenguaje es JavaScript?

Las siguientes son respuestas de múltiple opción.

3 | El DOM esta estandarizado ¿por cuál de las siguientes entidades? Detallá y describí.

W3C
Apple
ECMA
ISO
Microsoft
Mozilla

4 | Según el DOM toda pagina es un...

Atributo

Nodo

Tabla

Las tres son correctas

Luego de responder, inentá detallar el por qué.

5 | Investigá qué hacen las siguientes funciones : pow(), abs(), acos(), asin(), atan(), ceil(), floor(), cos(), exp(), log(), max(), Min(), random(), round(), sin(), sqrt(), tan().

6 | JavaScript ¿es un lenguaje compilado o interpretado? Justificá.

Autoevaluación

7 | Explicá con tus palabras y probá lo que realizan los siguientes script:

```
<script type="text/javascript">
window.onload=function(){alert('Bienvenido a esta pagina');}
window.onunload=function(){alert('Vuelva en otro momento');}
</script>
```

Descripcion:

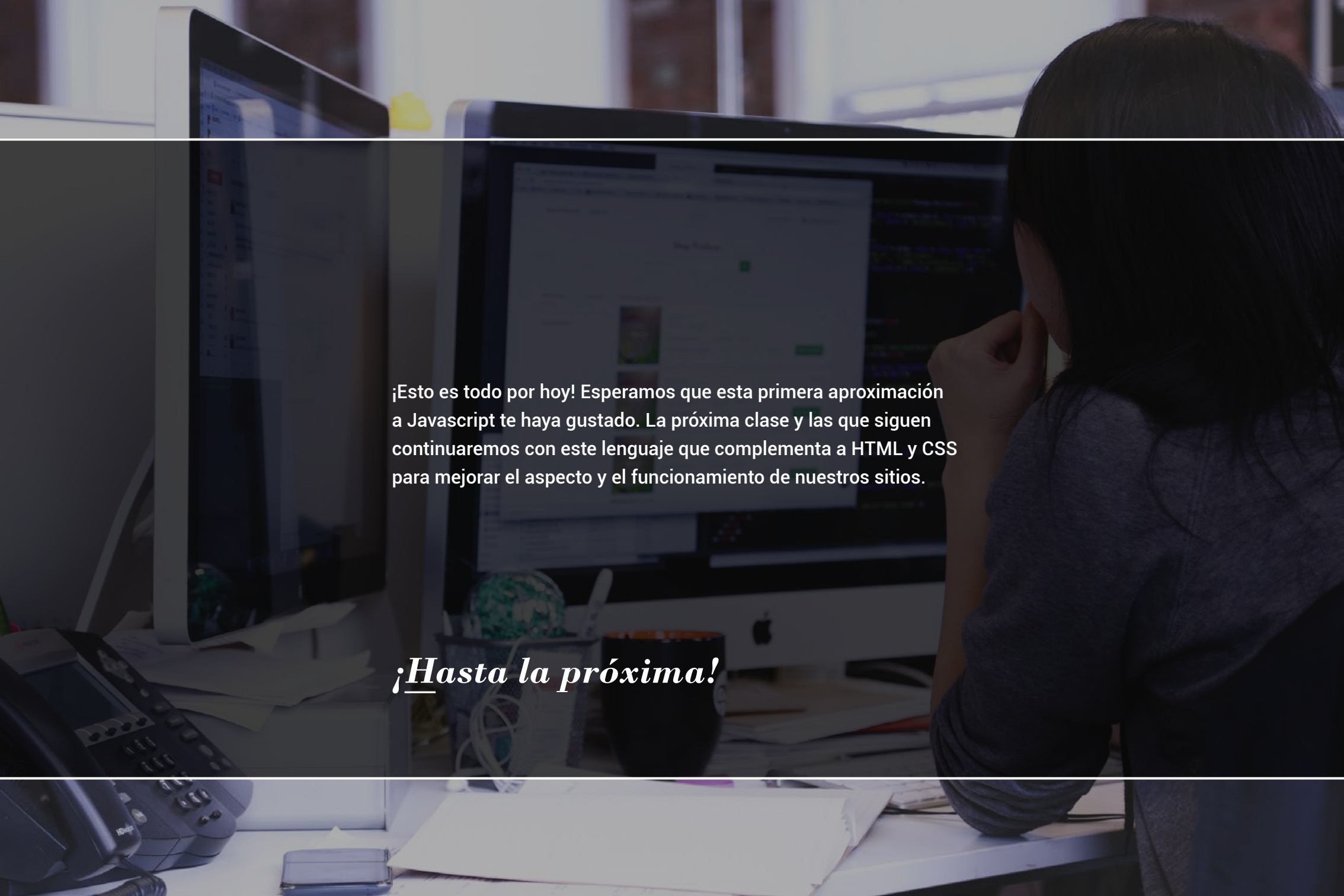
```
<script type="text/javascript">
window.onload = function(){
    // Para internet Explorer
    document.onselectstart = function(){return false;}
    // Para Firefox
    document.onmousedown = function(){return false;}}
</script>
```

Descripcion:

8 | El siguiente código tiene errores, indicá cuales son, corregilos y probalos.

```
<SCRIPT>
<HTML LANGUAGE="JavaScript"
// EVAL2A.HTM
var fahrenheit,celsius,
var s="";
for(i=-2;i<=12:i++)
{
    celsius=10*i;
    fahrenheit=32+(celsius*9)/5;
    s=s+"C= "+celsius+" F= "+fahrenheit+"\n";
    if (celsius==0) s=s+"Punto congelación del Agua\n";
    if (celsius==100) s=s+"Punto de ebullición del Agua\n";
}
alert(s;
<\\SCRIPT>
<HTML>
```

9 | Desarrollá ejemplos de código para las siguientes sentencias : return y case.

A person with long dark hair, wearing a grey sweater, is seen from the side, sitting at a desk in a dimly lit office. They are looking at two computer monitors. The monitor on the left shows a web browser with a list of items, and the monitor on the right shows a code editor with JavaScript code. The person's hand is resting on their chin, suggesting they are thinking or focused on their work. The desk is cluttered with papers, a telephone, and a small cup. The overall atmosphere is quiet and professional.

¡Esto es todo por hoy! Esperamos que esta primera aproximación a Javascript te haya gustado. La próxima clase y las que siguen continuaremos con este lenguaje que complementa a HTML y CSS para mejorar el aspecto y el funcionamiento de nuestros sitios.

¡Hasta la próxima!