

Curso Introducción a la
programación FrontEnd
Capacitaciones ADA

JavaScript Fundamental

Primeros pasos

Temario

Introducción al lenguaje

Breve historia, comentarios, codificación.

Variables

Declaraciones, alcances, variables locales y globales, buenas prácticas.

Operadores

Operadores matemáticos, lógicos.

Estructuras de condicionales

If-else, switch

Estructuras de repetición

while, do-while, for, break

Objeto String

Atributos y métodos

Objeto Numbers

Atributos y métodos

Objeto Math

Atributos y métodos

Arrays

Concepto, utilización, atributos y métodos. Multidimensionales

Funciones creadas por el usuario

Concepto, declaración, invocación, return.

Objeto Window

Atributos, métodos. Aplicación de métodos principales

DOM

Estructura general.

Objetos button, form, body

Validación de formularios

Expresiones regulares

Manejo de Cookies

Librerías

Introducción

¿Qué es JavaScript?

JavaScript es un lenguaje de programación interpretado, dialecto del estándar ECMAScript, el cual es utilizado en su forma del lado del cliente (lado-cliente), implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas, así como también para realizar validaciones previas al envío de información al servidor de manera tal de evitar su sobrecarga. JavaScript se diseñó con una sintaxis similar al C, es importante distinguir a JavaScript de Java, porque en muchos casos suele generar confusión y su semántica y propósito es totalmente diferente.

Características principales

Orientado a objetos: permite la implementación de este paradigma de programación.

Basado en prototipos: esto significa que las "clases" no están presentes, y la re-utilización de procesos *herencia*, se obtiene a través de la clonación de objetos ya existentes, que sirven de prototipos, extendiendo sus funcionalidades. Este modelo también es conocido como *programación basada en instancias*.

Imperativo: se basa en programación imperativa, lo que significa que utiliza este paradigma, en contraposición a la programación declarativa este paradigma describe la programación en términos del estado del programa y sentencias que cambian dicho estado. Los programas imperativos son un conjunto de instrucciones que le indican al sistema cómo realizar una tarea.

Débilmente tipado: al igual que lenguajes como PHP, JavaScript no requiere la declaración del tipo de una variable, sino que simplemente el hecho de inicializarla es suficiente para que la etiqueta interprete el tipo de dato especificado.

Dinámico: este principio surge de la capacidad de los objetos que conforman el lenguaje de cambiar la definición de sus miembros en tiempo de ejecución. Lo que le permite a JavaScript modificar los métodos que componen al objeto y de esta manera el flujo de programa.

Funcionamiento básico

Tradicionalmente se venía utilizando en páginas web HTML para realizar operaciones y únicamente en el marco de la aplicación cliente, sin acceso a funciones del servidor. JavaScript se interpreta en el agente de usuario, al mismo tiempo que las sentencias van descargándose junto con el código HTML.

Todos los navegadores modernos interpretan el código JavaScript integrado en las páginas web. Para interactuar con una página web se provee al lenguaje JavaScript de una implementación del DocumentObjectModel (DOM).

Versiones de JavaScript

Javascript 1: nació con el Netscape 2.0 y soportaba casi la misma cantidad de funciones que soporta actualmente.

Javascript 1.1: esta versión se diseñó con el arribo de los navegadores 3.0, implementó el manejo de imágenes de forma dinámica y la creación de arrays.

Javascript 1.2: Versión para navegadores 4.0, su principal desventaja es que difiere en plataformas Microsoft y Netscape, ya que ambos navegadores comenzaron a diferenciarse notablemente.

Javascript 1.3: Navegadores 5.0, se ha podido acercar un poco las versiones.

Javascript 1.4: Implementa Netscape 6.

Comentarios en el código

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

Ejemplo:

```
<script>

//Este es un comentario de una línea

/*Este comentario se puede extenderpor varias líneas.

Las necesarias*/

</script>
```

Mayúsculas y minúsculas

Javascript es un lenguaje **case sensitive**, esto significa que deberemos respetar las mayúsculas y las minúsculas, dado que sino el navegador generará un mensaje de error, ya sea de sintaxis o de referencia indefinida.

Por poner un ejemplo, no es lo mismo la función `alert()` que la función `Alert()`. 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. Como se puede comprobar, para que la función la reconozca Javascript, se tiene que escribir en minúscula. Otro claro ejemplo lo veremos cuando tratemos con variables, puesto que los nombres que damos a las variables también son sensibles a las mayúsculas y minúsculas.

Por regla general, los nombres dentro de Javascript se escriben siempre en minúsculas, salvo que se utilice un nombre con más de una palabra, pues en ese caso se escribirán con mayúsculas las iniciales de las palabras siguientes a la primera. (CamelCase).

Separación de instrucciones

Las distintas instrucciones que contienen nuestros scripts se separan para indicar su finalización y que pueda ser interpretada. 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.

Por esta razón, las sentencias Javascript no necesitan acabar en punto y coma a no ser que coloquemos dos instrucciones en la misma línea. Esta suele ser otra diferencia importante, respecto a lo que estamos acostumbrados de otros lenguajes de programación.

Inserción de código JavaScript

Es común ver código javascript incluido directamente dentro de las páginas web. Dentro del código HTML, lo cual no es la mejor práctica, dado que si los scripts utilizados son extensos, esto ensucia el código y complica la lectura. Por lo tanto lo recomendable es generar archivos .js donde se definan las funciones creadas por el usuario y se incluya dentro del HTML solamente este documento y las llamadas a la hora de ser utilizadas.

Generalmente cuando el código se incluye dentro del HTML, se hace de la siguiente forma.

```
<html>

  <head>

    <script>

      // código javascript

    </script>

  </head>

  <body>

    <!-- llamadas al código

  </body>

</html>
```

Llamada a archivo .js

```
<head>

  <script src="//ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"></script>

  <script type="text/javascript" src="/archivos/js/jPlayer/jquery.jplayer.min.js"></script>

</head>
```

Variables y tipos de datos

Concepto de Variable

Una variable es una etiqueta que determina una posición de memoria donde se guardará información, cuyo contenido podrá variar durante la ejecución del programa.

Declaración de Variables

Al igual que cualquier otro lenguaje de programación, en Javascript, se puede realizar la declaración de una variable, y en otra instancia del código la inicialización. Debemos tener en cuenta que este procedimiento no es obligatorio, por lo tanto se verá en muchas oportunidades que las variables son inicializadas al momento de ser requeridas omitiendo el paso de la declaración previa.

Es importante tener en cuenta en este tipo de lenguajes no estrictos, las buenas prácticas a la hora de la programación, para crear código ordenado, es por eso que se recomienda la declaración de las variables que se utilizarán.

Ejemplo:

```
var nombre;  
var apellido;
```

Las reglas para la declaración de variables, es simple, se deben evitar los nombres reservados (return, for, if), los caracteres que formen el nombre de la variable deben ser alfanuméricos, podrán incluir el (_), pero no símbolos como +, %, \$, ni espacios en blanco en el caso que el nombre esté compuesto por varias palabras.

Ejemplos válidos:

```
edad;  
_nombre;  
nombreDePais;
```

Ejemplos inválidos

```
nombre de país;  
+edad;
```

También es posible declarar y asignar múltiples variables en la misma línea de código, separándolas con comas.

Ejemplo:

```
var lastname="Doe", age=30, job="carpenter";
```

Concepto de Ámbito de las variables

Definimos como al ámbito de las variables al lugar donde podemos disponer de ellas.

Esto se desprende de su declaración, generalmente una variable tiene alcance o scope, en el lugar donde es declarada.

En Javascript como en otros lenguajes web, el máximo alcance una variable, es la página en la cual ha sido declarada, estas serán **variables globales** de la página en cuestión.

Variables Globales

Son aquellas cuyo alcance es el más amplio posible, podrán ser accedidas y modificadas desde cualquier parte del código de la página en la cual ha sido declarada.

Variables Locales

A diferencia de las variables locales, este tipo de variables poseen un alcance acotado, generalmente las variables locales son declaradas y utilizadas dentro de funciones.

Ejemplo:

```
<script>
    functionmiFuncion(){
        varvariableLocal;
    }
</script>
```

¿Por qué declarar las variables?

La respuesta a esta pregunta tiene una respuesta clara, para evitar la modificación indeseada de las variables. Ahora analicemos que es lo que sucede.

Como hemos dicho, en Javascript tenemos libertad para declarar o no las variables con la palabra **var**, pero los efectos obtenidos en cada caso serán distintos.

Cuando utilizamos **var** estamos haciendo que la variable declarada sea **local al ámbito donde se declara**. Por otro lado, si omitimos la palabra **var** para declarar una variable, ésta será global a toda la página, sin importar el ámbito en el que haya sido declarada.

En el caso de una variable declarada en la página web, fuera de una función o cualquier otro ámbito más reducido, nos es indiferente si se declara o no con **var**, desde un punto de vista funcional. Esto es debido a que cualquier variable declarada fuera de un ámbito es global a toda la página.

La diferencia se puede apreciar en una función por ejemplo, ya que si utilizamos **var** la variable será local a la función y si no lo utilizamos, la variable será global a la página. Esta diferencia es fundamental a la hora de controlar correctamente el uso de las variables en la página, ya que si no lo hacemos en una función podríamos sobre escribir el valor de una variable, perdiendo el dato que pudiera contener previamente.

Tipos de datos - Numérico

En este lenguaje sólo existe un tipo de datos numérico, al contrario que ocurre en la mayoría de los lenguajes de programación. *Todos los números son del tipo numérico, independientemente de la precisión que tengan o si son números reales o enteros.*

Los números enteros son números que no tienen coma, como 3 o 339. Los números reales son números fraccionarios, como 2.69 o 0.25, que también se pueden escribir en notación científica, por ejemplo 2.482e12.

Con Javascript también podemos escribir números en otras bases, como la hexadecimal. Existen tres bases que podemos utilizar:

- Base 10, es el sistema que utilizamos habitualmente, el sistema decimal. Cualquier número, por defecto, se entiende que está escrito en base 10.
- Base 8, también llamado sistema octal, que utiliza dígitos del 0 al 7. Para escribir un número en octal basta con escribir ese número precedido de un 0, por ejemplo 045.
- Base 16 o sistema hexadecimal, es el sistema de numeración que utiliza 16 dígitos, los comprendidos entre el 0 y el 9 y las letras de la A a la F, para los dígitos que faltan. Para escribir un número en hexadecimal debemos escribirlo precedido de un cero y una equis, por ejemplo 0x3EF.

Tipos de datos - boolean

El tipo boolean, sirve para guardar un valor verdadero o falso (true-false). Se utiliza para realizar operaciones lógicas, generalmente para realizar acciones si el contenido de una variable es verdadero o falso.

Tipos de datos - cadena de caracteres

Todo lo que se coloca entre comillas, es tratado como una cadena de caracteres independientemente de lo que coloquemos en el interior de las comillas. Entonces no es lo mismo 4 que "4", principalmente a la hora de aplicar funciones sobre la variable, no podríamos sumar "4"+"2", sino que estaríamos concatenando dos variables cuyo contenido son cadenas de caracteres.

Operadores

Los operadores se pueden clasificar según el tipo de acciones que realizan. A continuación vamos a ver cada uno de estos grupos de operadores y describiremos la función de cada uno.

Operadores aritméticos

Son utilizados para la realización de operaciones matemáticas simples como la suma, resta o multiplicación.

Símbolo	Función
+	Suma dos valores
-	Resta de dos valores, también puede utilizarse para cambiar el signo de un número si lo utilizamos con un solo operando -23
*	Multiplicación de dos valores
/	División de dos valores
%	El resto de la división de dos números (3%2 devolvería 1, el resto de dividir 3 entre 2)
++	Incremento en una unidad, se utiliza con un solo operando
--	Decremento en una unidad, utilizado con un solo operando

Ejemplo:

```
precio = 128;
```

```
unidades = 10;
```

```
factura = precio * unidades; //multiplico precio por unidades, obtengo el valor factura
```

Operadores de asignación

Se utilizan para asignar valores a las variables, este operador (=) resulta muy familiar, dado que la mayoría de los lenguajes de programación lo utilizan para sus asignaciones.

Símbolo	Función
=	Asignación. Asigna la parte de la derecha del igual a la parte de la izquierda. A la derecha se colocan los valores finales y a la izquierda generalmente se coloca una variable donde queremos guardar el dato.
+=	Asignación con suma. Realiza la suma de la parte de la derecha con la de la izquierda y guarda el resultado en la parte de la izquierda.
-=	Asignación con resta.
*=	Asignación de multiplicación
/=	Asignación de división
%=	Se obtiene el resto y se asigna

Ejemplo:

```
total = 7000 //asigna un 7000 a la variable ahorros
```

```
total += 3500 //incrementa en 3500 la variable ahorros, ahora vale 10500
```

Operadores de comparación

Hemos revisado dos tipos importantes de operadores, cuyo uso es básico, en este punto, presentamos los operadores de comparación, los cuales generalmente serán utilizados en bloques condicionales para poder determinar acciones dentro de nuestro programa.

En la siguiente tabla vemos todos los operadores de comparación disponibles en JavaScript, es importante tener en cuenta que para todos ellos el valor de retorno, será un valor boolean (true-false).

Símbolo	Función	Expresión	Valor de Retorno
==	Es igual a	<code>x==8</code>	<i>false</i>
		<code>x==5</code>	<i>true</i>
===	es exactamente igual (valor and tipo)	<code>x==="5"</code>	<i>false</i>
		<code>x===5</code>	<i>true</i>
!=	No igual (distinto)	<code>x!=8</code>	<i>true</i>
!==	No igual (ni valor ni tipo)	<code>x!="5"</code>	<i>true</i>
		<code>x!==5</code>	<i>false</i>
>	Mayor que	<code>x>8</code>	<i>false</i>
<	Menor que	<code>x<8</code>	<i>true</i>
>=	Mayor e igual	<code>x>=8</code>	<i>false</i>
<=	Menor e igual	<code>x<=8</code>	<i>true</i>

Operadores lógicos

Este grupo de operadores son simplemente la representación del lenguaje de las funciones de lógica básica. Al igual que para los operadores de comparación, se utilizan ampliamente en los bloques condicionales.

Símbolo	Función	Expresión
&&	and	<code>(x < 10 && y > 1)</code> is true
 	or	<code>(x==5 y==5)</code> is false
!	not	<code>!(x==y)</code> is true

A continuación, revisaremos el comportamiento de los operadores lógicos teniendo en consideración sus tablas de verdad.

Operador AND

El operador lógico AND (&&) obliga a que ambas condiciones sean verdaderas para obtener un resultado verdadero.

A continuación, se ve la tabla de verdad de la función, generalmente se realiza la analogía de esta función con la multiplicación, dado que siempre que hay un valor FALSE (0), el resultado es también FALSE (0).

X	Y	Salida
FALSE	FALSE	FALSE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
TRUE	TRUE	TRUE

Operador OR

De forma complementaria se puede utilizar el operador lógico OR (`||`), este dará como resultado verdadero cuando al menos una de las condiciones sea verdadera.

A continuación, se muestra la tabla de verdad de la función.

X	Y	Salida
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	TRUE

Operador condicional

Como en algunos otros lenguajes, JavaScript cuenta con el operador condicional, este operador asigna un valor a una variable a partir de una determinada condición.

Sintaxis:

variable1=(condicion) ? valor :valor2

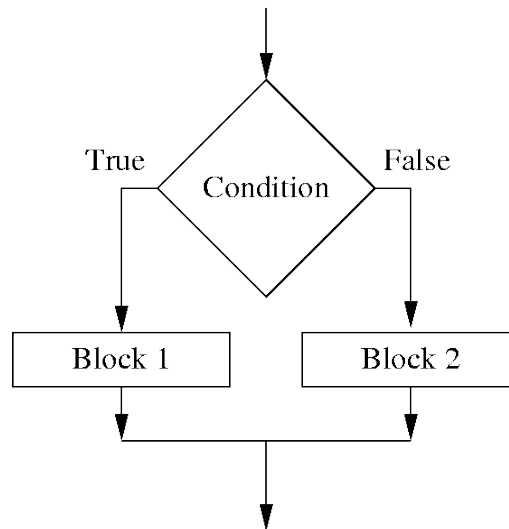
(x%2 ==0) 2 3

La variable1 tomará el valor 2 o 3.

Estructuras de control condicionales

Hasta este punto, hemos visto todos los elementos necesarios para escribir bloques de código, los cuales se ejecutan de forma secuencial. Es claro también, que cualquier sistema necesita poder manejar bloques o estructuras de decisión, esto significa que a partir del valor de una variable o varias se determine la secuencia de código a ejecutar de forma tal de poder brindarle al sistema cierta inteligencia y capacidad de decisión.

Para esto se utilizan los bloques (**IF-ELSE**), en la siguiente imagen se muestra un diagrama de flujo que permite visualizar la idea de la bifurcación a partir de una decisión.



Dependiendo del cumplimiento o no de la condición se ejecutará el bloque 1 o el bloque 2 de código.

Ejemplo:

```
if(time<20)
{
  x="Good day";
}
```

Como vemos en el ejemplo, solo nos interesa la opción verdadera, no tomamos ninguna acción para el resultado falso.

Ejemplo:

```
if(time<20)
{
  x="Good day";
}else{
  x="Cold day";
}
```

En el Segundo ejemplo, agregamos el bloque **else**, para determinar la acción que se llevará a cabo en el caso que la condición resulte falsa.

Ya hemos visto los operadores lógicos, los cuales nos permitirán elaborar condiciones compuestas para los bloques **if-else** tan complejas como requiramos.

Bloque de múltiple condiciones

Otra opción del bloque if-else, cuando no es útil anidarlos, es usar

```
if (condition1) { code to be executed if condition1 is true }
```

```
else if (condition2) { code to be executed if condition2 is true }
```

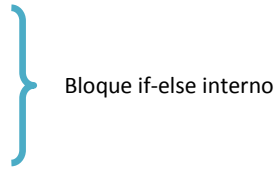
```
else { code to be executed if neither condition1 nor condition2 is true }
```

Bloques anidados

Al igual que en otros lenguajes de programación JavaScript permite el uso anidado de bloques condicionales.

Ejemplo:

```
if(time<20)
{
    if(hour < 30)
    {
        wait+=5;
    }else{
        wait =0;
    }
}else
{
    x = "Finish";
}
```



Switch

Este bloque de programación resulta una alternativa al uso de **if-else**, en el caso que se requiera manejar distintas opciones mutuamente excluyentes. Qué significa esto, en el caso por ejemplo que se quiera manejar un menú de opciones podría resolverse tanto con un bloque **if-else**, o con un bloque **switch**.

```
if(opcion == 1){ x += 3; }
```

```
if(opcion == 2) { x*=2;}
```

```
if(opcion==3){ x-=5;}
```

Y así podrían agregarse más opciones. Este mismo ejemplo puede resolverse con el bloque **switch**.

```
switch(opcion){  
  
    case 1:x += 3; break;  
  
    case 2:x*=2; break;  
  
    default:x;  
  
}
```

La opción se evalúa para cada caso y si coincide el valor, se ejecutarán las instrucciones correspondientes. La sentencia **break**, es muy importante dado que evita que se continúe con la ejecución de las sentencias que corresponden a las otras opciones. La ausencia de **break**, no provocará un error sintáctico que pueda ser detectado por el navegador al momento de interpretar el código, sino que el error será lógico y el resultado obtenido será erróneo.

La sentencia **default** se utiliza para determinar una acción en el caso que el valor no coincida con ninguna opción válida.

Ejemplo:

```
var day=new Date().getDay();  
  
switch (day){  
  
    case 0: x="Domingo"; break;  
  
    case 1: x="Lunes";break;  
  
    case 2: x="Martes"; break;  
  
    case 3: x="Miercoles"; break;  
  
    case 4:x="Jueves"; break;  
  
    case 5: x="Viernes"; break;  
  
    case 6: x="Sabado";break;  
  
}
```

Estructuras de repetición

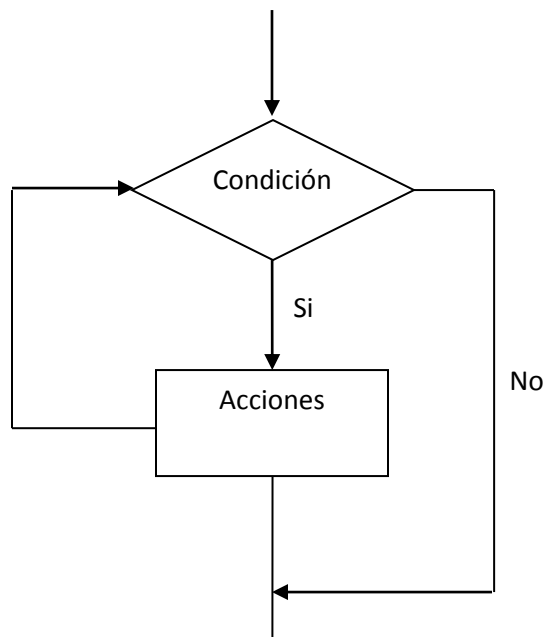
En muchas instancias de un programa es necesario repetir ciertos bloques de código una determinada cantidad de veces, para esto como otros lenguajes de programación PHP soporta tres tipos de bucles distintos.

While

En primera instancia se analizará la estructura **while**, esta estructura repite las sentencias que componen su cuerpo mientras la condición resulte verdadera.

El **while** evalúa la veracidad de la condición antes de ingresar al ciclo por lo cual, las sentencias pueden ejecutarse ninguna o muchas veces, dependiendo del cumplimiento de la condición.

El siguiente gráfico detalla el funcionamiento de este tipo de loop. Se puede ver que mientras se cumpla la condición se repetirán las acciones y una vez que el resultado de la evaluación da falso, se ejecutará la siguiente línea de código fuera del ciclo de repetición. Se evidencia además que en caso que inicialmente no se cumpla la condición se seguirá la línea alternativa y el flujo de programa omitiendo la entrada en el bloque de repetición.



La sintaxis del loop es:

```
while (condicion) {  
    codigo a ejecutar;  
    cambio en la condicion;  
}
```

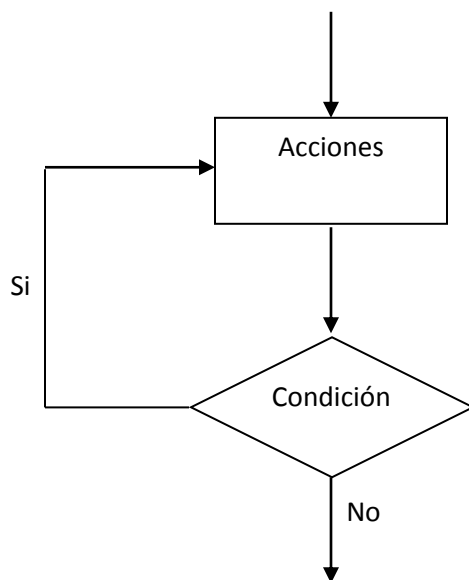

Ejemplo

```
while (i<5) {  
  
    x= "El numero es " + i + "<br>";  
  
    i++;  
  
}
```

Do-while

Como opción al **while**, el bloque **do-while**, evalúa la condición al final, esto significa que luego de ejecutado el ciclo se evalúa la condición por lo cual, en este caso al menos una vez se ejecutan las sentencias.

La imagen a continuación muestra el diagrama de flujo que representa al **do-while**.



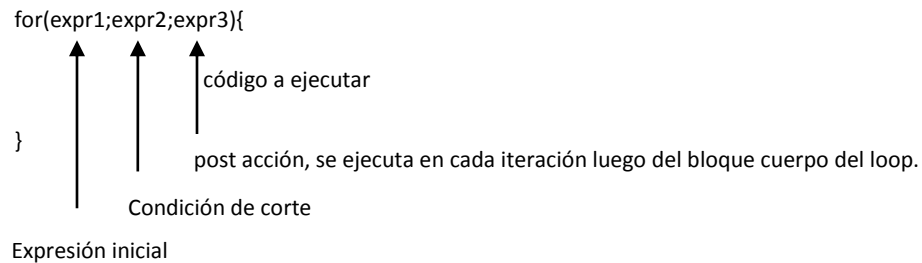
Realizando un análisis similar al del caso del **while**, se ve claramente que las acciones se ejecutarán al menos una vez, dado que la validez de la condición, se verifica recién al final.

Ejemplo

```
do {  
  
    x= "El numero es " + i + "<br>";  
  
    i++;  
  
}while (i<5);
```

For

Como último bloque de iteraciones analizaremos el **for**, este se compone por tres expresiones, como se muestra en el siguiente esquema. La primera expresión (*expr1*) es evaluada (ejecutada) una vez incondicionalmente al comienzo del bucle, la segunda (*expr2*) es la condición que determina cuando finaliza la serie de iteraciones y la última (*expr3*) es la denominada post-acción, la cual se ejecuta al finalizar cada iteración y donde se realizan las modificaciones de la variable de control.



Ejemplo:

Si deseamos imprimir los valores corridos del 1 al 10, el código resultante se muestra a continuación:

```
for (vari=0;i<10;i++){  
    document.write(i + "<br>");  
}
```

En el comienzo de cada iteración, la *expr2* (*i<10*) es evaluada, Si es **TRUE**, el bucle continúa y las sentencias dentro del cuerpo de la estructura son ejecutadas. Si se evalúa como **FALSE**, termina la ejecución del bucle.

Al final de cada iteración, la *expr3* (*i++*) es evaluada (ejecutada).

Cada una de las expresiones puede estar vacía o contener múltiples expresiones separadas por comas. En la *expr2*, todas las expresiones separadas por una coma son evaluadas pero el resultado se toma de la última parte.

La *expr3*, puede ser omitida y resuelta la modificación dentro del cuerpo del loop.

Ejemplo:

```
vari=0;  
for (; i<len; ){  
    document.write(i + "<br>");  
    i++; // post acción  
}
```

Break

Cuando analizamos el bloque switch, realizamos la presentación de la sentencia break, la cual se utilizaba para determinar el final de la ejecución de un bloque de código. En este caso dentro de los bloques de iteración la sentencia break permite salir de un bucle, esto significa que break, permite la interrupción de un ciclo de iteración y continuar con la siguiente instrucción fuera del mismo.

Ejemplo:

```
for (i=0;i<10;i++) {  
    if (i==3) {  
        break;  
    }  
    document.write(i + "<br>");  
}
```

Objeto String

JavaScript como muchos otros lenguajes de programación posee funciones predefinidas (métodos) para el manejo de los objetos que lo componen, las cuales realizan tareas específicas y permiten simplificar la programación.

Es importante reconocerlos como objetos métodos de un objeto específico que forma parte del lenguaje y no como simples funciones aisladas, dado que estos métodos en algunos casos modifican los atributos.

En la siguiente tabla enumeramos algunos de los métodos más utilizados para el manejo de cadenas de caracteres.

Función	Descripción	Ejemplo
length	Determina la cantidad de caracteres de una cadena.	<code>texto.length</code>
indexOf()	Devuelve la posición de la primera ocurrencia del texto pasado por parámetro. Retorna -1 si no se encuentra.	<code>var str="Hola mundo."; var n=str.indexOf("hola");</code>
lastIndexOf()	Comienza a buscar desde el final de la cadena.	Idem anterior
replace()	Reemplaza una parte específica de una cadena por otra.	<code>str="Visitenuestra web" var n=str.replace("web","generandoIT");</code>
toUpperCase() / toLowerCase()	Convierte a mayúsculas o minúsculas una cadena.	<code>var txt="Hola Mundo"; var txt1=txt.toUpperCase(); var txt2=txt.toLowerCase();</code>
split()	Convierte una cadena en un array, como parámetro se toma el elemento separador.	<code>txt="a,b,c,d,e"; txt.split(",");</code>
slice()	Extrae partes de una cadena y los retorna en una nueva.	<code>var str="Hello world!"; var n=str.slice(1,5);</code>
substr()	Extrae una parte de una cadena comenzando por el carácter especificado y la cantidad de caracteres también pasadas por parámetro.	<code>var str="Hello world!"; var n=str.substr(2,3)</code>
search()	Busca una cadena específica dentro de otra a partir de una expresión regular y devuelve la posición. Retorna -1 si no la encuentra.	<code>var str="Me encanta javascript"; var n=str.search("Javascript");</code>

Objeto Numbers

Funciones numéricas

Como ya hemos mencionado antes al analizar los tipos de datos soportados por javascript, el tipo numérico soportado es único por lo tanto, los métodos asociados nos permiten realizar ciertas conversiones sobre los elementos numéricos, los mostraremos en la siguiente tabla con más detalle.

Función	Descripción	Ejemplo
toExponential()	Convierte un número a su notación exponencial.	<code>varnum = 5.56789; var n=num.toExponential()</code>
toFixed()	Convierte un número a cadena de caracteres, manteniendo la cantidad de decimales definida como parámetro.	<code>varnum = 5.56789; var n=num.toFixed(2);</code>
toString()	Convierte un número a cadena de caracteres	<code>varnum = 5.56789; var n=num.toString();</code>
toPrecision()	Formatea el número a la precisión definida por parámetro.	<code>varnum = new Number(13.3714); var n=num.toPrecision(2);</code>

Objeto Math

Este objeto permite la realización de ciertas operaciones matemáticas, como atributos definimos a continuación una lista de las constantes que lo componen.

Constantes

Math.E : valor de e

Math.PI : constante pi

Math.SQRT2

Math.SQRT1_2

Math.LN2 : logaritmo en base 2

Math.LN10 : logaritmo en base 10

Math.LOG2E : logaritmo natural

Math.LOG10E

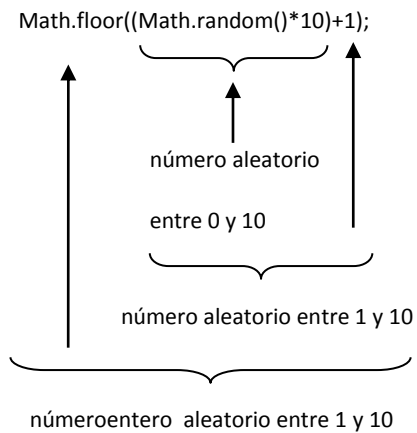
Ejemplo

```
var x=2 *Math.PI *radio;
```

Función	Descripción	Ejemplo
round()	Redondea al entero más cercano	Math.round(2.5);
random()	Retorna un número aleatorio entre 0 y 1	Math.random();
floor()	Redondea al entero más cercano para abajo.	Math.floor(1.6);

El método `random()`, merece que se le brinde principal atención, dado que si bien en su versión básica nos devolverá un valor aleatorio entre 0 y 1, nos permite combinarlo de forma tal de flexibilizar el resultado obtenido.

Analicemos el siguiente ejemplo.



Objeto Array

Antes de introducirnos en el manejo de la estructura por parte de javascript, debemos analizar conceptualmente que es un Array.

¿ Qué es un array ?

Durante los primeros pasos en la programación, se utilizan variables simples, en las cuales se puede guardar un valor determinado por cada variable declarada.

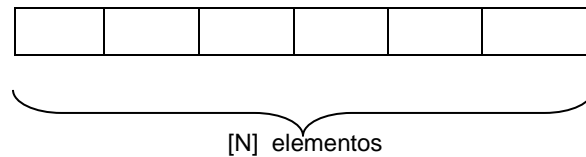
En el caso que debamos manejar un conjunto de datos asociados a la misma variable, debemos recurrir a los arrays.

Definición

Un array en PHP es realmente un mapa ordenado. Un mapa es un tipo de datos que asocia valores con claves. Este tipo es optimizado para varios usos diferentes, puede ser usado como una matriz real, una lista (vector), una tabla asociativa (una implementación de un mapa), diccionario, colección, pila, cola, etc. Los valores de un array pueden ser otros arrays, árboles y hasta arrays multidimensionales.

Los arrays se distinguen por su dimensión o tamaño, en la siguiente imagen se muestra un array unidimensional de N elementos, por lo cual se dice que su tamaño es N (1xN).

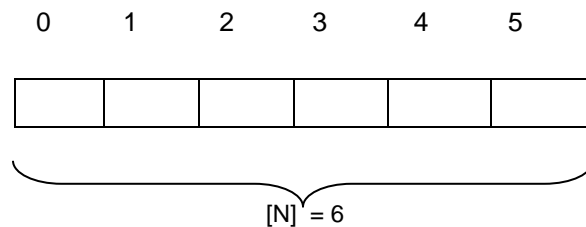
Cada una de las casillas modeladas en la imagen contendrá un elemento del array que podrá ser accedido a través de una clave única.



Arrays secuenciales

En este tipo de arrays, las claves utilizadas son numéricas y asignadas automáticamente cada vez que se agrega un nuevo elemento al array, de forma incremental.

Hemos ya definido uno de los componentes que definen a un array, la longitud, por otro lado, tenemos la clave o índice que en el caso de los arrays secuenciales se definen de 0 a N-1, como se muestra en la figura, un array secuencial de longitud 6, tendrá sus claves definidas entre 0 y 5.



Arrays en JavaScript

En este lenguaje existen tres formas de crear un Array.

1-Regular

```
varmyCars=new Array();

myCars[0]="Saab";

myCars[1]="Volvo";

myCars[2]="BMW";
```

2- Compacta

```
varmyCars=new Array("Saab","Volvo","BMW");
```

3- Literal

```
varmyCars=["Saab","Volvo","BMW"];
```

Claramente las dos últimas opciones resultan útiles cuando conocemos al momento de la creación el valor de los elementos que deseamos almacenar. En el caso que decidamos completar un **array** utilizando un bucle for, la opción regular podría ser reescrita.

En la siguiente tabla se enumeran los métodos propios del objeto array de javascript.

Método	Descripción
<u>concat()</u>	Une dos o más arrays, y retorna una copia de los arrays unidos
<u>indexOf()</u>	Busca dentro de un array un elemento y devuelve su índice
<u>join()</u>	Une todos los elementos de un array en una cadena de caracteres.
<u>lastIndexOf()</u>	Search the array for an element, starting at the end, and returns its position
<u>pop()</u>	Remueve el último elemento de un array, y lo retorna.
<u>push()</u>	Agrega un nuevo elemento al final del array, y retorna la nueva longitud.
<u>reverse()</u>	Invierte el orden de los elementos de un array.
<u>shift()</u>	Quita el primer elemento de unarray, y lo retorna
<u>slice()</u>	Selecciona una parte de un array, y lo retorna
<u>sort()</u>	Ordena los elementos de un array.
<u>splice()</u>	Agrega/quita elementos de un array
<u>toString()</u>	Convierte un array a string
<u>unshift()</u>	Agrega nuevos elementos al principio del array, y retorna la nueva longitud
<u>valueOf()</u>	Retorna el valor primitive de un array

Arrays multidimensionales

Javascript al igual que otros lenguajes soporta arrays multidimensionales (matrices), las cuales en muchos casos son utilizadas para guardar información y operar con ella.

Ya hemos visto que los arrays, son definidos por dos componentes, su dimensión y su índice, el cual nos permite a cada elemento de forma particular. En los arrays multidimensionales simplemente se trabajará con múltiples índices para tener acceso a los elementos.

Ya vimos que dentro de un array puede guardarse cualquier tipo de información, veamos la siguiente creación literal.

```
var array = ["My", "Small array", true, {name:'John'}, 345]
```

Diagram illustrating the types of elements in the array:

- String elements: "My", "Small array"
- Boolean element: true
- JSON element: {name:'John'}
- Number element: 345

De forma similar definimos matrices (arrays multidimensionales), como arrays de arrays.

Ejemplo

```
var matrix = [ [1, 2, 3], [4, 5, 6], [7, 8, 9] ]
```

Diagram illustrating the structure of the matrix:

- Row 1 elements: [1, 2, 3]
- Row 3 elements: [7, 8, 9]

Funciones definidas por el usuario

En todos los lenguajes de programación, una de las formas mas prácticas de mantener el código ordenado y poder reutilizar bloques, es la utilización de funciones definidas por nosotros mismos.

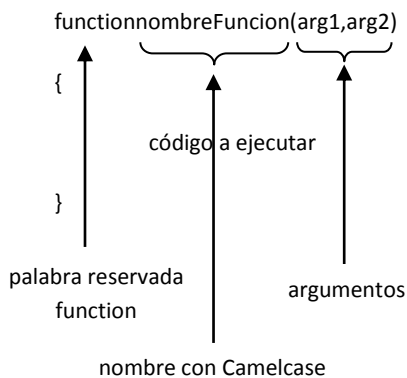
Definición de función

Una función es un bloque de código que realiza una acción determinada, al cual se le define un alcance y puede ser invocado cada vez que sea necesario dentro de un programa.

Una vez revisada la definición veamos la definición y utilización de funciones en Javascript.

Declaración de una función

Vemos la estructura de una función simple, la cual recibe dos argumentos y no retorna ningún valor, a diferencia de otros lenguajes de programación en el caso de que no se retornen valores, puede omitirse la palabra reservada `return` dentro del cuerpo de la función.



Declaración de una función con valor de retorno

En lo que respecta a la declaración de la función, simplemente se agregará la palabra reservada **return** y la variable a retornar.

Ejemplo

```
functionesPar(numero){  
  
  var x;  
  if(numero%2 == 0){  
    x = "El numero"+numero+" es par";  
  
  }else{  
  
    x = "El numero"+numero+" es impar"  }  
  
  return x;  
  
}
```

En el ejemplo anterior se muestra una función la cual devuelve una cadena de caracteres luego de verificar si un número pasado por argumento es o no par.

Ahora veamos cómo se invoca esta función desde cualquier parte de nuestro programa, teniendo en cuenta el ejemplo anterior.

Invocación de una función

```
varesPar = esPar(5);
```

El valor retornado en la variable esPar será: “El número 5 es impar” y permanecerá en la variable para futuros usos, hasta que decidamos modificar su contenido.

Existe una alternativa a la utilización de una variable de retorno, esta opción se utiliza mayormente cuando deseamos que el resultado sea mostrado directamente sin ser manipulado.

Ejemplo

```
function myFuncion(a,b)
{
    return a*b;
}

document.getElementById("demo").innerHTML=myFuncion(4,3);
```

Accedo al elemento Setea con el valor de
retorno de la función

Uno de los primeros conceptos que hemos revisado en este manual fue el alcance de las variables (local, global), en este punto es importante mencionar que una variable declarada utilizando **var dentro de una función**, tendrá alcance local dentro de la misma y no podrá ser utilizada fuera. Esto también deja ver que si tenemos variables dentro de distintas funciones se puede utilizar el mismo nombre, ya que son locales a las mismas y no habrá problemas de solapamiento.

Tiempo de vida de las variables

Una variable local declarada dentro de una función es eliminada al terminar la ejecución de la función.

Una variable global es eliminada cuando se cierra la página web.

Objeto Window

Definición

El objeto **window** representa una ventana abierta en un navegador.

Características

Si un documento contiene marcos (<frame> o <iframe>tags), el navegador crea un objeto de la ventana para el documento HTML, y un objeto adicional de la ventana para cada fotograma. No existe una norma pública que se aplica al objeto **window**, pero todos los navegadores soportan.

A continuación mostraremos en detalle los métodos que conforman este objeto, si bien muchos de ellos como alert() ya comentado o utilizados previamente en otros ejemplos.

Método	Descripción	Nota
<u>alert()</u>	Muestra un alert box con un mensaje y un botón OK	alert("Gracias por su visita").
<u>blur()</u>	Quita el foco de la ventana actual.	
<u>clearInterval()</u>	Borra el seteo de setInterval()	
<u>clearTimeout()</u>	Clears a timer set with setTimeout()	
<u>close()</u>	Cierra la ventana actual	
<u>confirm()</u>	Muestra un dialog box con un mensaje y botones OK y Cancel.	confirm("Esta seguro de abandonar esta página")
<u>createPopup()</u>	Crea un pop-up	Este método solo es soportado por IE.
<u>focus()</u>	Pone foco en la ventana actual	
<u>moveBy()</u>	Mueve una ventana en forma relativa a la posición actual.	
<u>moveTo()</u>	Mueve una ventana a una posición específica.	
<u>open()</u>	Abre una nueva ventana de navegador	window.open(URL,name,specs,replace)
<u>print()</u>	Prints the content of the current window	
<u>prompt()</u>	Muestra un dialog box para que el usuario pueda ingresar contenido.	
<u>resizeBy()</u>	Resiza la ventana a los pixels especificados	
<u>resizeTo()</u>	Resiza la ventana a un ancho y alto específico	
<u>scrollBy()</u>	Scrollea el contenido un específico número de pixels	
<u>scrollTo()</u>	Scrollea el contenido a coordenadas específicas	window.scrollTo(100,200)
<u>setInterval()</u>	Invoca una función o evalúa una expresión cada una determinada cantidad de tiempo (milisegundos).	
<u>setTimeout()</u>	Invoca una función o evalúa una expresión luego de una determinada cantidad de milisegundos	

Miremos algunos ejemplos de los métodos más usados, como el método confirm() combinado con alert().

Ejemplo:

```
<html><head>

<script>

    function disp_confirm(){

        var r=confirm("Por favor, presione un boton")

        if (r==true) { alert("OK") }

        else { alert("Cancel") }

    }

</script></head>

<body>

<input type="button" onclick="disp_confirm()" value="Confirmar">

</body></html>
```

window.open(URL,name,specs,replace), resulta un método muy útil es por eso que analizaremos sus parámetros en forma detallada.

Parámetros

- URL: Opcional. Especifica la dirección URL de la página que desea abrir. Si no se especifica ninguna dirección URL, una nueva ventana se abre en blanco.
- name: Opcional. Especifica el atributo de destino o el nombre de la ventana. Los valores siguientes son compatibles:
 - `_blank` - URL se carga en una ventana nueva. Esta es la opción predeterminada
 - `_parent` - URL se carga en el marco padre
 - `_self` - URL reemplaza la página actual
 - `_top` - URL reemplaza cualquier conjuntos de marcos que se pueden cargar
 - `name` - El nombre de la ventana
- specs: Opcional. Una lista separada por comas de materiales. Los valores siguientes son compatibles:
 - `channelmode` = yes | no | 1 | 0 si desea que se muestre la ventana en modo teatro. El valor predeterminado es no. es decir, sólo
 - `directorios` = yes | no | 1 | 0 si desea que se agregue botones de directorio. Por defecto es sí. es decir, sólo
 - `fullscreen` = yes | no | 1 | 0 si desea que se muestre el navegador en modo de pantalla completa. El valor predeterminado es no. Una ventana en modo de pantalla completa también debe estar en modo teatro. es decir, sólo
 - `altura` = píxeles la altura de la ventana. Min. valor es 100
 - `izquierda` = píxeles La posición izquierda de la ventana
 - `location` = yes | no | 1 | 0 si desea que se muestre la barra de direcciones. Por defecto es sí

- menubar = yes | no | 1 | 0 si desea que se muestre la barra de menús. Por defecto es sí
 - tamaño variable = yes | no | 1 | 0 Sea o no la ventana es de tamaño variable. Por defecto es sí
 - barras de desplazamiento = yes | no | 1 | 0 Sea o no mostrar las barras de desplazamiento. Por defecto es sí
 - status = yes | no | 1 | 0 Sea o no agregar una barra de estado. Por defecto es sí
 - titlebar = yes | no | 1 | 0 si desea que se muestre la barra de título. Omite a menos que la aplicación de llamada es una aplicación HTML o un cuadro de diálogo de confianza. Por defecto es sí
 - toolbar = yes | no | 1 | 0 si desea que se muestre la barra de herramientas del navegador. Por defecto es sí.
 - top = píxeles. La posición superior de la ventana. Sólo IE.
 - width = píxeles El ancho de la ventana. Min. valor es 100
- replace: Opcional. Determina si la URL crea una entrada nueva o sustituye la entrada actual en la lista de historial. Los valores siguientes son compatibles:
 - true - URL reemplaza el documento actual en la lista del historial.
 - false - URL crea una nueva entrada en la lista del historial.

window.focus(), este método no tiene parámetros adicionales por lo tanto sólo mostraremos un ejemplo de cómo crear una nueva ventana sobre la actual.

Ejemplo:

```
<html><head>

<script>

    function openWin(){

        myWindow=window.open("",'',width=200,height=100');

        myWindow.document.write("<p>This is 'myWindow'</p>");

        myWindow.focus();

    }

</script></head>

<body>

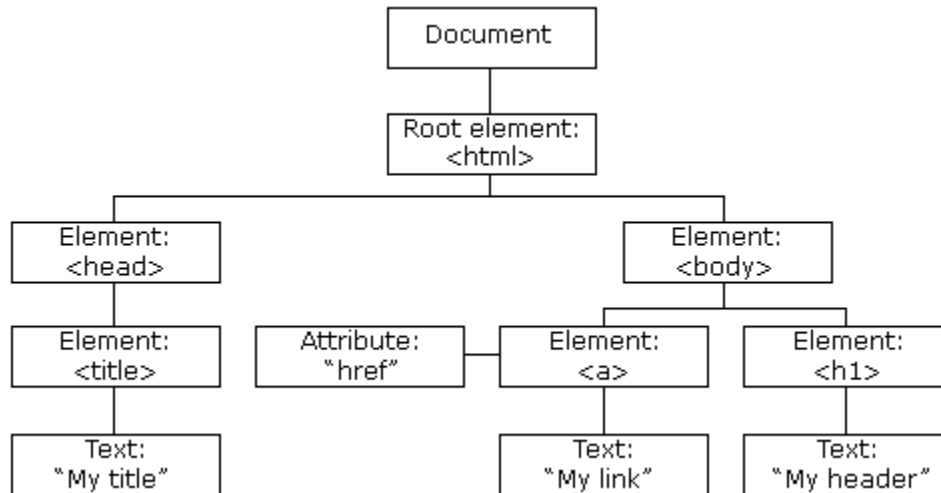
    <input type="button" value="Open window" onclick="openWin()">

</body></html>
```

DOM (Document Object Model)

Una vez conocidos los conceptos básicos de programación en JavaScript, es muy importante analizar el DOM, dado que nos permite acceder a todos los elementos del HTML y manipular de forma dinámica la página.

Cada vez que se carga una página web, se crea un DOM de la misma, el cual posee la siguiente estructura de árbol.



En el gráfico vemos los elementos conocidos de HTML, en este punto, podemos utilizar JavaScript para realizar modificaciones sobre distintos aspectos del DOM.

- Modificaciones sobre los elementos.
- Modificaciones sobre los atributos de los elementos.
- Modificaciones sobre los estilos CSS.
- Agregar comportamientos y respuesta a eventos del usuario.

Claramente para realizar cualquiera de las acciones que comentamos deberemos distinguir unívocamente cada uno de los elementos y localizarlos. Esto puede hacerse de tres formas distintas:

1. Determinando al elemento por el **Id**.
2. Determinando al elemento por el **tagname**.
3. Determinando al elemento por **classname**.

En la siguiente tabla se enumeran los métodos propios del DOM, los que nos dan la posibilidad de manipular los elementos HTML de una página.

Método	Descripción
getElementById()	Retorna el elemento perteneciente al ID con el valor pasado

getElementsByTagName()	Retorna una lista (collection/array of nodes) conteniendo todos los elementos con el Tagname especificado.
getElementsByClassName()	Retorna una lista de los nodos que contengan todos los elementos de la clase específica.
appendChild()	Agrega un nuevo nodo hijo a un nodo específico
removeChild()	Elimina un nodo hijo.
replaceChild()	Reemplaza un nodo hijo.
insertBefore()	Inserta un nuevo nodo hijo antes de un nodo específico
createAttribute()	Crea un atributo para el nodo
createElement()	Crea un elemento para el nodo
createTextNode()	Crea texto
getAttribute()	Devuelve el valor del atributo especificado
setAttribute()	Setea o cambia el atributo especificado al valor especificado

Revisemos en detalle alguno de los métodos, en pequeños ejemplos.

Ejemplo:

```
<html><head>

<script>

function getValue() {

var x=document.getElementById("myHeader");

alert(x.innerHTML);

}

</script></head>

<body>

<h1 id="myHeader" onclick="getValue()">Click aqui</h1>

</body></html>
```

En el ejemplo anterior hemos combinado distintos conceptos, en primera instancia creamos una función denominada `getValue()`, dentro de la cual se toma el valor del elemento a través del método `document.getElementById()` y se muestra utilizando un `alert()`. Luego utilizamos el evento `onclick` para invocar a nuestra función a partir del link.

Objeto button

Dentro del modelo DOM, definimos los elementos básicos utilizados comúnmente por javascript. Este objeto en particular lo hemos utilizado anteriormente para invocar funciones a través de sus eventos. En esta sección lo analizaremos en profundidad.

Definición

El objeto **button** representa un botón que se pulse en un formulario HTML.

Características

- Un botón se utiliza a menudo para activar el JavaScript cuando un usuario hace clic en él.
- `<input type="button">` se crea un objeto Button.
- Se puede acceder a un objeto Button mediante la búsqueda a través de los elementos de matriz [] de un formulario, o utilizando `document.getElementById ()`.

Propiedad	Descripción	W3C
<u>disabled</u>	Establece o devuelve si un botón está desactivado, o no	Si
<u>form</u>	Devuelve una referencia al formulario que contiene el botón	Si
<u>name</u>	Establece o devuelve el valor del atributo de nombre de un botón	Si
<u>type</u>	Devuelve el tipo de botón	Si
<u>value</u>	Establece o devuelve el valor del atributo value de un botón	Si

Ejemplo:

El atributo **disabled** es muy utilizado en formularios cuando se desea bloquear una acción, hasta que se complete un campo.

```
<button type="button" disabled>Ok</button>
```

Objeto Form

Otro objeto muy utilizado en la programación web, es el objeto formulario, el cual permite el intercambio de información entre cliente y servidor.

Definición

El objeto **Form** representa un formulario HTML, el cual contendrá todos los elementos necesarios como input, botones. Para crear uno de estos objetos se utiliza el tag <form>.

Características

Este objeto al igual que otros que ya hemos analizado, posee tanto atributos como métodos asociados, los cuales mostramos en las siguientes tablas.

Propiedades

Propiedad	Descripción	W3C
<u>acceptCharset</u>	Setea o retorna el valor de accept-charset en el form	Si
<u>action</u>	Setea o retorna la URL a la cual se envía la información	Si
<u>enctype</u>	Sets or returns the value of the enctype attribute in a form	Si
<u>length</u>	Retorna el número de elementos del form	Si
<u>method</u>	Setea o retorna el método (POST, GET)	Si
<u>name</u>	Setea o retorna el nombre del formulario	Si
<u>target</u>	Setea o retorna el target	Si

Métodos

A continuación vemos los dos métodos de este objeto los cuales nos permiten enviarlo o limpiarlo, dependiendo de que se desee.

Method	Description	W3C
<u>reset()</u>	Resetea un form	Yes
<u>submit()</u>	Submite un form	Yes

Eventos

En los objetos anteriores no hemos mencionado eventos, sin embargo en este caso resulta importante presentarlos dado que más adelante cuando trabajamos sobre la validación de formularios resultarán muy útiles y haremos distintas referencias a ellos.

Evento	Ocurrencia	W3C
<u>onreset</u>	El botón reset está clickeado	Yes
<u>onsubmit</u>	El botón submit está clickeado	Yes

Ejemplo

```
<html>

<body>

    <form id="frm1" action="form_action.php">

        First name: <input type="text" name="fname" value="Donald"><br>

        Last name: <input type="text" name="lname" value="Duck"><br>

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

    </form>

<script>

document.write(document.getElementById("frm1").action);

</script>

</body>

</html>
```

Salida: form_action.php

En el ejemplo anterior, verificamos el valor retornado mostrado por pantalla haciendo uso del atributo **action**, de la misma manera podemos verificar otro atributo como **method**.

Objeto Body

Definición

El elemento **body** contiene todo el contenido de un documento HTML, como texto, hipervínculos, imágenes, tablas, listas, etc.

El objeto **body** representa el elemento del cuerpo HTML y define el cuerpo de un documento.

En la siguiente tabla enumeramos los atributos de este objeto.

Propiedad	Descripción	W3C
<u>aLink</u>	Setea o retorna el valor del atributo alink	Si
<u>background</u>	Setea o retorna el valor del atributo background del Body	Si
<u>bgColor</u>	Setea o retorna el valor de bgcolor del Body	Si
<u>link</u>	Setea o retorna el valor de link	Si
<u>text</u>	Setea o retorna el valor del atributo text	Si
<u>vLink</u>	Setea o retorna el valor vLink	Si

Ejemplo

```
<html>

<body id="bg" background="fondo.png">

<script>

    document.write("The background image is: ")

    document.write(document.getElementById("bg").background);

</script>

</body>

</html>
```

Salida: fondo.png

Expresiones regulares

Como muchos otros lenguajes de programación Javascript soporta trabajar con expresiones regulares, haciendo uso de funciones predefinidas del lenguaje.

¿Qué es una expresión regular?

Las expresiones regulares son modelos que describen las combinaciones de caracteres en el texto. Se podrían definir como una serie de caracteres que forman un patrón, que representan a otro grupo de caracteres mayor, de tal forma que podemos comparar el patrón con otros conjuntos de caracteres para ver las coincidencias.

Todos los patrones se escriben en barras, de esta forma podemos definir un patrón como:

```
varpatron = /prueba/
```

A continuación se muestran ejemplos de dos métodos javascript para evaluar expresiones regulares.

Método test()

El método test () busca en una cadena un valor especificado por parámetro y devuelve true o false, dependiendo de si lo encuentra o no.

Ejemplo

```
var patt1=new RegExp("e");  
  
document.write(patt1.test("GenerandoIT"));
```

Salida: true

Método exec()

El método exec () busca dentro de una cadena un valor especificado por parámetro y devuelve el texto del valor encontrado. Si no se encuentra una coincidencia, se devuelve un valor nulo.

Reescribiendo el objeto anterior vemos que valor se retorna

```
var patt1=new RegExp("e");  
  
document.write(patt1.exec("GenerandoIT"));
```

Salida: e

A continuación revisaremos un ejemplo completo utilizado en la validación de formularios. En este caso quiere validarse que la clave ingresada resulte segura. Condición: entre 8 y 10 caracteres, por lo menos un dígito y un alfanumérico, y no puede contener caracteres espaciales.

```
<html ><head>
```

```
<title></title>
```

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

```
function validatePass(campo) {
```

```
varRegExPattern = /(?!^[0-9]*$)(?!^[a-zA-Z]*$)^[a-zA-Z0-9]{8,10}$/;
```

← Patrón para verificación

```
varerrorMessage = 'Password Incorrecta.';
```

```
if ((campo.value.match(RegExPattern)) && (campo.value!="")) {
```

```
    alert('Password Correcta');
```

```
} else {
```

```
    alert(errorMessage);
```

```
    campo.focus();
```

```
}
```

```
}
```

```
</script></head>
```

```
<body>
```

```
<form action="#" method="post">
```

```
    <p><input type="text" name="date" onblur="validatePass(this);">
```

```
    <input name="button" type="button" value="Probar">
```

```
    <br>
```

```
</form>
```

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

El método `match()` busca una cadena contra una expresión regular y devuelve las coincidencias en un objeto Array.

Muestra un mensaje de error y pone el foco en el campo que se volver a completar.

Se invoca a la función cuando el campo pierde el focus. (onblur)

Cookies

¿Qué es una cookie?

Una cookie es una variable que se almacena en la computadora del visitante. Cada vez que se realicen solicitudes del equipo a una página con un navegador, junto con la información solicitada se enviará la cookie también. Con JavaScript, se pueden crear y recuperar los valores de las cookies.

En un primer momento las cookies fueron utilizadas para recordar información y preferencias del usuario, por lo tanto cuando este retornaba a una página, la cual había guardado una cookie, esta la re-direcciona automáticamente al último lugar visitado.

Analicemos a continuación funciones de Javascript para el manejo de Cookies.

Setear una cookie (set)

El objeto es guardar una cookie que conserve el nombre de un visitante de un sitio web, durante un determinado tiempo también determinado por parámetro. Luego cuando el usuario retorne se mostrará un mensaje de bienvenida a partir de la cookie.

```
function setCookie(c_name,value,exdays){
```

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

← Se crea un objeto date, con fecha y hora actual.

```
    exdate.setDate(exdate.getDate() + exdays);
```

← Sete la fecha a partir de los parámetros pasados. Se genera la fecha de expiración de la cookie, sumando la actual mas una cantidad de días.

```
    var c_value=escape(value) + ((exdays==null) ? "" : "; expires=" + exdate.toUTCString());
```

```
    document.cookie=c_name + "=" + c_value;
```

```
}
```

Recuperar una cookie (get)

Esta función permite retornar el valor de una cookie específica, tengamos en cuenta que como en otros lenguajes de programación cookies es un array que contendrá todas las cookies que creemos.

```
function getCookie(c_name){
```

```
    varc_value = document.cookie;
```

Permite obtener y setear las cookies asociadas con el documento actual.

```
    varc_start = c_value.indexOf(" " + c_name + "=");
```

```
    if (c_start == -1) {
```

```
        c_start = c_value.indexOf(c_name + "=");
```

```
    }
```

```
    if (c_start == -1) {
```

```
        c_value = null;
```

```
    }else {
```

```
        c_start = c_value.indexOf("=", c_start) + 1;
```

```
        varc_end = c_value.indexOf(";", c_start);
```

```
        if (c_end == -1) {
```

```
            c_end = c_value.length;
```

```
        }
```

```
        c_value = unescape(c_value.substring(c_start,c_end));
```

```
    }
```

```
    return c_value;
```

```
}
```

Por último analizaremos un ejemplo en el cual chequeamos el valor de la cookie creada con los métodos anteriores.

check Cookie

```
function checkCookie(){
```

```
var username=getCookie("username");
```

Utilizamos el método anterior para recuperar el valor de la cookie pasada por parámetro.

```
    if (username!=null && username!="") {
```

```
        alert("Welcome again " + username);
```

```
    }else {
```

```
        username=prompt("Please enter your name:", "");
```

```
        if (username!=null && username!="") {
```

```
            setCookie("username",username,365);
```

Si la cookie solicitada no está seteada, se solicita al usuario que cargue la información.

```
        }
```

```
    }
```

```
}
```

Librerías JavaScript

La programación Avanzada sobre JavaScript (especialmente el manejo de las diferencias entre los navegadores Web), puede ser a menudo muy difícil y requiere mucho tiempo para trabajar con ellos. Por este motivo generalmente se utilizan librerías (frameworks) que resuelven ciertas necesidades.

Los frameworks javascript más populares son:

- jQuery
- Prototype
- MooTools
- Dojo

Todos estos framework tienen funciones para tareas comunes de JavaScript como animaciones, manipulación DOM y manipulación Ajax.

jQuery

jQuery es uno de los frameworks de JavaScript más populares hoy. Utiliza selectores CSS para acceder y manipular los elementos HTML (DOM Objects) en una página web.

jQuery también proporciona una interfaz de usuario amigable (IU) y un conjunto de plug-ins, que permiten optimizar las aplicaciones Javascript.

Prototype

Prototype es una librería JavaScript que proporciona una API simple para realizar tareas comunes web. Prototype mejora la programación en JavaScript al proporcionar clases y herencia.

¿Qué es una API?

API es la abreviatura de Interfaz de programación de aplicaciones. Es una biblioteca de propiedades y métodos para manipular el DOM HTML.