JAVASCRIPT (CONCEPTOS SENCILLOS Y BÁSICOS)

Comentarios en el código

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

Se han de respetar las mayúsculas y las minúsculas (Son diferentes). Si nos equivocamos al utilizarlas el navegador responderá con un mensaje de error, ya sea de sintaxis o de referencia indefinida.

Por regla general, los nombres de las cosas en 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.

Separación de instrucciones

Hay 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.

VARIABLE

Una variable es un espacio en memoria donde se almacena un dato, 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. El efecto sería algo parecido a esto.

```
sumando1 = 23
sumando2 = 33
suma = sumando1 + sumando2
```

Los nombres de las variables han de construirse con caracteres alfanuméricos y el carácter subrayado (_). Además, tienen que comenzar por un carácter alfabético o el subrayado. No podemos utilizar caracteres raros como el signo +, un espacio o un $. Nombres admitidos para las variables podrían ser:

Edad
paisDeNacimiento
_nombre

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 de que vas a utilizar una variable. Pero Javascript otorga un poco de libertad a la hora de declarar las variables, ya que no estamos obligados a hacerlo, al contrario de lo que pasa en la mayoría de los lenguajes de programación. Buena costumbre es no declararlas.

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
```

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

```
var operando1,operando2
```

Concepto de ámbito de variables

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

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.

Variables globales

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.

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

En el script anterior 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.

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.

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

Un consejo para los principiantes podría ser no declarar variables con los mismos nombres, para que nunca haya lugar a confusión sobre qué variable es la que tiene validez en cada momento.

Diferencias entre declarar variables con var, o no declararlas

En concreto, cuando utilizamos **var** estamos haciendo que la variable que estamos declarando sea local al ámbito donde se declara. Por otro lado, si no utilizamos la palabra var para declarar una variable, ésta será global a toda la página, sea cual sea 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 sobreescribir el valor de una variable, perdiendo el dato que pudiera contener previamente.

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

En este ejemplo, tenemos una variable global a la página llamada numero, que contiene un 2. También tenemos una función que utiliza la variable numero sin haberla declarado con var, por lo que la variable numero de la funcion será la misma variable global numero declarada fuera de la función. En una situación como esta, al ejecutar la función se sobrescribirá la variable numero y el dato que había antes de ejecutar la función se perderá.

TIPOS DE DATOS

En una variable podemos introducir varios tipos de información. Por ejemplo podríamos introducir simple texto, números enteros o reales, etc. A estas distintas clases de información se les conoce como tipos de datos. Cada uno tiene características y usos distintos.

Números

Para empezar, tenemos el tipo numérico, para guardar números como 9 o 23.6

Cadenas

El tipo cadena de carácter guarda un texto. Siempre que escribamos una cadena de caracteres debemos utilizar las comillas (").

Boleanos

También contamos con el tipo boleano, que guarda una información que puede valer si (true) o no (false).

```
var nombre_ciudad = "Valencia"
var revisado = true
```

Tipo de datos numérico

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 y octal.

Tipo de datos cadena de caracteres

El último tipo de datos es el que sirve para guardar un texto. Un texto puede estar compuesto de números, letras y cualquier otro tipo de caracteres y signos. Los textos se escriben entre comillas, dobles o simples.

```
miTexto = "Pepe se va a pescar"
miTexto = '23%%$ Letras & *--*'
```

Tipo boleano

Sirve para guardar un si o un no o dicho de otro modo, un verdadero o un falso. Se utiliza para realizar operaciones lógicas, generalmente para realizar acciones si el contenido de una variable es verdadero o falso.

Si una variable es verdadero entonces ----- Ejecuto unas instrucciones
Si no ----- Ejecuto otras
Los dos valores que pueden tener las variables booleanas son true o false.

```
miBoleana = true
<br>
miBoleana = false
```

Caracteres de escape en cadenas de texto

Hay una serie de caracteres especiales que sirven para expresar en una cadena de texto determinados controles como puede ser un salto de línea o un tabulador. Estos son los caracteres de escape y se escriben con una notación especial que comienza por una contra barra \ y luego se coloca el código del carácter a mostrar.

Tabla con todos los caracteres de escape

Salto de línea: \n
Comilla simple: \'
Comilla doble: \"
Tabulador: \t
Retorno de carro: \r
Avance de página: \f
Retroceder espacio: \b
Contrabarra: \\

Ejemplos de uso de operadores

En el primer ejemplo vamos a realizar una suma utilizando el operador suma.

3 + 5

Normalmente se combinan más de un operador para crear expresiones más útiles. La expresión siguiente es una combinación entre dos operadores, uno realiza una operación matemática y el otro sirve para guardar el resultado.

```
miVariable = 23 * 5
```

En el ejemplo anterior, el operador * se utiliza para realizar una multiplicación y el operador = se utiliza para asignar el resultado en una variable, de modo que guardemos el valor para su posterior uso.

TIPOS DE OPERADORES

Operadores aritméticos

Son los siguientes:

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

Ejemplos

```
precio = 128 //introduzco un 128 en la variable precio
unidades = 10 //otra asignación, luego veremos operadores de
asignación
factura = precio * unidades //multiplico precio por unidades, obtengo
el valor factura
resto = factura % 3 //obtengo el resto de dividir la variable factura
por 3
precio++ //incrementa en una unidad el precio (ahora vale 129)
```

Operadores de asignación

= 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 la multiplicación
/= Asignación de la división
%= Se obtiene el resto y se asigna

Ejemplos

```
ahorros = 7000 //asigna un 7000 a la variable ahorros
ahorros += 3500 //incrementa en 3500 la variable ahorros, ahora vale
10500
ahorros /= 2 //divide entre 2 mis ahorros, ahora quedan 5250
```

Operadores de cadenas

+ Concatena dos cadenas, pega la segunda cadena a continuación de la primera.

Ejemplo

```
cadena1 = "hola"
cadena2 = "mundo"
cadenaConcatenada = cadena1 + cadena2 //cadena concatenada vale
"holamundo"
```

Un caso que resultaría confuso es el uso del operador + cuando se realiza la operación con operadores texto y numéricos entremezclados. En este caso javascript asume que se desea realizar una concatenación y trata a los dos operandos como si de cadenas de caracteres se trataran, incluso si la cadena de texto que tenemos fuese un número. Esto lo veremos más fácilmente con el siguiente ejemplo.

```
miNumero = 23
miCadena1 = "pepe"
miCadena2 = "456"
resultado1 = miNumero + miCadena1 //resultado1 vale "23pepe"
resultado2 = miNumero + miCadena2 //resultado2 vale "23456"
miCadena2 += miNumero //miCadena2 ahora vale "45623"
```

Como hemos podido ver, también en el caso del operador +=, si estamos tratando con cadenas de texto y números entremezclados, tratará a los dos operadores como si fuesen cadenas.

Operadores lógicos

Realizan operaciones lógicas, que son aquellas que dan como resultado un verdadero o un falso.

Ejemplo: Si tengo hambre y tengo comida entonces me pongo a comer

Primero mirará si tengo hambre, si es cierto (true) mirará si dispongo de comida. Si son los dos ciertos, se puede poner a comer. En caso de que no tenga comida o que no tenga hambre no comería, al igual que si no tengo hambre ni comida. El operando en cuestión es el operando Y, que valdrá verdadero (true) en caso de que los dos operandos valgan verdadero.

Sobra decir que, para ver ejemplos de operadores condicionales, necesitamos aprender estructuras de control como if, a las que no hemos llegado todavía.

! Operador NO o negación. Si era true pasa a false y viceversa.
&& Operador Y, si son los dos verdaderos vale verdadero.
|| Operador O, vale verdadero si por lo menos uno de ellos es verdadero.

Ejemplo

```
miBoleano = true
miBoleano = !miBoleano //miBoleano ahora vale false
tengoHambre = true
tengoComida = true
comoComida = tengoHambre && tengoComida
```

Operadores condicionales

Estas expresiones se utilizan para tomar decisiones en función de la comparación de varios elementos, por ejemplo, si un numero es mayor que otro o si son iguales.

Como estas expresiones condicionales serán objeto de estudio más adelante será mejor describir los operadores condicionales más adelante. De todos modos aquí podemos ver la tabla de operadores condicionales.

== Comprueba si dos valores son iguales
!= Comprueba si dos valores son distintos
> Mayor que, devuelve true si el primer operando es mayor que el segundo
< Menor que, es true cuando el elemento de la izquierda es menor que el de la derecha
>= Mayor igual
<= Menor igual

Veremos ejemplos de operadores condicionales cuando expliquemos estructuras de control, como la condicional if.

Precedencia de los operadores

Muchas de estas reglas de precedencia están sacadas de las matemáticas, las podemos ver a continuación.

() [] . Paréntesis, corchetes y el operador punto que sirve para los objetos
! - ++ -- negación, negativo e incrementos
* / % Multiplicación división y módulo
+- Suma y resta
< <= > >= Operadores condicionales
== != Operadores condicionales de igualdad y desigualdad
& ^ | Lógicos a nivel de bit
&& || Lógicos boleanos
= += -= *= /= %= <<= >>= >>>= &= ^= != Asignación

Ejemplos:

```
12 * 3 + 4 - 8 / 2 % 3
```

En este caso primero se ejecutan los operadores * / y %, de izquierda a derecha, con lo que se realizarían estas operaciones. Primero la multiplicación y luego la división por estar más a la izquierda del módulo.

```
36 + 4 - 4 % 3
```

Ahora el módulo.

```
36 + 4 - 1
```

Por último, las sumas y las restas de izquierda a derecha.

```
40 - 1
```

Lo que nos da como resultado el valor siguiente.

```
39
```

ESTRUCTURAS DE CONTROL

Toma de decisiones

Nos sirven para realizar unas acciones u otras en función del estado de las variables. Es decir, tomar decisiones para ejecutar unas instrucciones u otras dependiendo de lo que esté ocurriendo en ese instante en nuestros programas.

En Javascript podemos tomar decisiones utilizando dos enunciados distintos.

- IF
- SWITCH

Bucles

Los bucles se utilizan para realizar ciertas acciones repetidamente.

Existen varios tipos de bucles, cada uno está indicado para un tipo de iteración distinto y son los siguientes:

- FOR
- WHILE
- DO WHILE

Estructura de control if.

IF es una estructura de control utilizada para **tomar decisiones**. Es un condicional que sirve para realizar unas u otras operaciones en función de una expresión. Funciona de la siguiente manera, primero se evalúa una expresión, si da resultado positivo se realizan las acciones relacionadas con el caso positivo.

La sintaxis de la estructura IF es la siguiente

Todas las estructuras de control se escriben en minúsculas en Javascript.

```
if (expresión) {
   //acciones a realizar en caso positivo
   //...
}
```

Opcionalmente se pueden indicar acciones a realizar en caso de que la evaluación de la sentencia devuelva resultados negativos.

```
if (expresión) {
   //acciones a realizar en caso positivo
   //...
} else {
```

```
   //acciones a realizar en caso negativo
   //...
}
```

Fijémonos en varias cosas. Para empezar, vemos como con unas llaves engloban las acciones que queremos realizar en caso de que se cumplan o no las expresiones. Estas llaves han de colocarse siempre, excepto en el caso de que sólo haya una instrucción como acciones a realizar, que son opcionales.

Veamos algún ejemplo de condicionales IF.

```
if (dia == "lunes")
   document.write ("Que tengas un feliz comienzo de semana")
```

Si es lunes nos deseará una feliz semana. No hará nada en caso contrario. Como en este ejemplo sólo indicamos una instrucción para el caso positivo, no hará falta utilizar las llaves (aunque sí sería recomendable haberlas puesto). Fíjate también en el operador condicional que consta de dos signos igual.

Vamos a ver ahora otro ejemplo, un poco más largo.

```
if (credito >= precio) {
   document.write("has comprado el artículo " + nuevoArtículo)
//enseño compra
   carrito += nuevoArticulo //introduzco el artículo en el carrito de
la compra
   credito -= precio //disminuyo el crédito según el precio del
artículo
} else {
   document.write("se te ha acabado el crédito") //informo que te
falta dinero
   window.location = "carritodelacompra.html" //voy a la página del
carrito
}
```

Expresiones condicionales

La expresión a evaluar se coloca siempre entre paréntesis y está compuesta por variables que se combinan entre si mediante operadores condicionales. Recordamos que los operadores condicionales relacionaban dos variables y devolvían siempre un resultado booleano. Por ejemplo, un operador condicional es el operador "es igual" (==), que devuelve true en caso de que los dos operandos sean iguales o false en caso de que sean distintos.

```
if (edad > 18)
   document.write("puedes ver esta página para adultos")
```

En este ejemplo utilizamos en operador condicional "es mayor" (>). En este caso, devuelve true si la variable edad es mayor que 18, con lo que se ejecutaría la línea siguiente que nos informa de que se puede ver el contenido para adultos.

Las expresiones condicionales se pueden combinar con las expresiones lógicas para crear expresiones más complejas. Recordamos que las expresiones lógicas son las que tienen

como operandos a los booleanos y que devuelvan otro valor booleano. Son los operadores negación lógica, Y lógico y O lógico.

```
if (bateria < 0.5 && redElectrica == 0)
   document.write("tu ordenador portatil se va a apagar en segundos")
```

Sentencias IF anidadas

Para hacer estructuras condicionales más complejas podemos anidar sentencias IF, es decir, colocar estructuras IF dentro de otras estructuras IF. Con un solo IF podemos evaluar y realizar una acción u otra según dos posibilidades, pero si tenemos más posibilidades que evaluar debemos anidar IFs para crear el flujo de código necesario para decidir correctamente.

Por ejemplo, si deseo comprobar si un número es mayor menor o igual que otro, tengo que evaluar tres posibilidades distintas. Primero puedo comprobar si los dos números son iguales, si lo son, ya he resuelto el problema, pero si no son iguales todavía tendré que ver cuál de los dos es mayor. Veamos este ejemplo en código Javascript.

```
var numero1=23
var numero2=63
if (numero1 == numero2){
   document.write("Los dos números son iguales")
}else{
   if (numero1 > numero2) {
      document.write("El primer número es mayor que el segundo")
   }else{
      document.write("El primer número es menor que el segundo")
   }
}
```

Estructura de control SWITCH

Nos sirve para tomar decisiones en función de distintos estados de las variables. Esta expresión se utiliza cuando tenemos múltiples posibilidades como resultado de la evaluación de una sentencia.

Su sintaxis es la siguiente.

```
switch (expresión) {
        case valor1:
        Sentencias a ejecutar si la expresión tiene como valor a valor1
        break
        case valor2:
        Sentencias a ejecutar si la expresión tiene como valor a valor2
        break
        case valor3:
        Sentencias a ejecutar si la expresión tiene como valor a valor3
        break
        default:
        Sentencias a ejecutar si el valor no es ninguno de los
anteriores
}
```

La expresión se evalúa, si vale valor1 se ejecutan las sentencias relacionadas con ese caso. Si la expresión vale valor2 se ejecutan las instrucciones relacionadas con ese valor y así sucesivamente, por tantas opciones como deseemos. Finalmente, para todos los casos no contemplados anteriormente se ejecuta el caso por defecto.

La palabra break es opcional, pero si no la ponemos a partir de que se encuentre coincidencia con un valor se ejecutarán todas las sentencias relacionadas con este y todas las siguientes. Es decir, si en nuestro esquema anterior no hubiese ningún break y la expresión valiese valor1, se ejecutarían las sentencias relacionadas con valor1 y también las relacionadas con valor2, valor3 y default.

También es opcional la opción default u opción por defecto.

Veamos un ejemplo de uso de esta estructura. Supongamos que queremos indicar que día de la semana es. Si el día es 1 (lunes) sacar un mensaje indicándolo, si el día es 2 (martes) debemos sacar un mensaje distinto y así sucesivamente para cada día de la semana, menos en el 6 (sábado) y 7 (domingo) que queremos mostrar el mensaje "es fin de semana". Para días mayores que 7 indicaremos que ese día no existe.

```
switch (dia_de_la_semana) {
        case 1:
        document.write("Es Lunes")
        break
        case 2:
        document.write("Es Martes")
        break
        case 3:
        document.write("Es Miércoles")
        break
        case 4:
        document.write("Es Jueves")
        break
        case 5:
        document.write("Es viernes")
        break
        case 6:
        case 7:
        document.write("Es fin de semana")
        break
        default:
        document.write("Ese día no existe")
}
```

El ejemplo es relativamente sencillo, solamente puede tener una pequeña dificultad, consistente en interpretar lo que pasa en el caso 6 y 7, que habíamos dicho que teníamos que mostrar el mismo mensaje. En el caso 6 en realidad no indicamos ninguna instrucción, pero como tampoco colocamos un break se ejecutará la sentencia o sentencias del caso siguiente, que corresponden con la sentencia indicada en el caso 7 que es el mensaje que informa que es fin de semana. Si el caso es 7 simplemente se indica que es fin de semana, tal como se pretendía.

El bucle FOR

Se utiliza para repetir una o más instrucciones un determinado número de veces. De entre todos los bucles, el FOR se suele utilizar cuando sabemos seguro el número de veces que queremos que se ejecute. La sintaxis del bucle for se muestra a continuación.

```
for (inicialización; condición; actualización) {
        //sentencias a ejecutar en cada iteración
}
```

El bucle FOR tiene tres partes incluidas entre los paréntesis, que nos sirven para definir cómo deseamos que se realicen las repeticiones. La primera parte es la inicialización, que se ejecuta solamente al comenzar la primera iteración del bucle. En esta parte se suele colocar la variable que utilizaremos para llevar la cuenta de las veces que se ejecuta el bucle.

La segunda parte es la condición, que se evaluará cada vez que comience una iteración del bucle. Contiene una expresión para decidir cuándo se ha de detener el bucle, o mejor dicho, la condición que se debe cumplir para que continúe la ejecución del bucle.

Por último, tenemos la actualización, que sirve para indicar los cambios que queramos ejecutar en las variables cada vez que termina la iteración del bucle, antes de comprobar si se debe seguir ejecutando.

Después del for se colocan las sentencias que queremos que se ejecuten en cada iteración, acotadas entre llaves.

Un ejemplo donde se imprimirán los números del 0 al 10.

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

En este caso se inicializa la variable i a 0. Como condición para realizar una iteración, se tiene que cumplir que la variable i sea menor o igual que 10. Como actualización se incrementará en 1 la variable i.

Por ejemplo, escribir los número del 1 al 1.000 de dos en dos.

```
for (i=1;i<=1000;i+=2)
        document.write(i)
```

Si nos fijamos, en cada iteración actualizamos el valor de i incrementándolo en 2 unidades.

Si queremos contar descendentemente del 343 al 10 utilizaríamos este bucle.

```
for (i=343;i>=10;i--)
        document.write(i)
```

En este caso decrementamos en una unidad la variable i en cada iteración, comenzando en el valor 343 y siempre que la variable tenga un valor mayor o igual que 10.

Ejercicio de ejemplo del bucle for

Se trata de hacer un bucle que escriba en una página web los encabezamientos desde `<H1>` hasta `<H6>` con un texto que ponga "Encabezado de nivel x".

Lo que deseamos escribir en una página web mediante Javascript es lo siguiente:

```
<H1>Encabezado de nivel 1</H1>
<H2>Encabezado de nivel 2</H2>
<H3>Encabezado de nivel 3</H3>
<H4>Encabezado de nivel 4</H4>
<H5>Encabezado de nivel 5</H5>
<H6>Encabezado de nivel 6</H6>
```

Para ello tenemos que hacer un bucle que empiece en 1 y termine en 6 y en cada iteración escribiremos el encabezado que toca.

```
for (i=1;i<=6;i++) {
        document.write("<H" + i + ">Encabezado de nivel " + i + "</H" +
i + ">")
}
```

Bucle WHILE

Estos bucles se utilizan cuando queremos repetir la ejecución de unas sentencias un número indefinido de veces, siempre que se cumpla una condición.

```
while (condición){
        //sentencias a ejecutar
}
```

Un ejemplo de código donde se utiliza este bucle se puede ver a continuación.

```
var color = ""
while (color != "rojo"){
        color = prompt("dame un color (escribe rojo para salir)","")
}
```

Este es un ejemplo de lo más sencillo que se puede hacer con un bucle while. Lo que hace es pedir que el usuario introduzca un color y lo hace repetidas veces, mientras que el color introducido no sea rojo. Para ejecutar un bucle como este primero tenemos que inicializar la variable que vamos utilizar en la condición de iteración del bucle. Con la variable inicializada podemos escribir el bucle, que comprobará para ejecutarse que la variable color sea distinto de "rojo".

Nota: Hemos utilizado en este ejemplo la función prompt de Javascript. Esta función sirve para que mostrar una caja de diálogo donde el usuario debe escribir un texto.

Bucle DO...WHILE

Se utiliza generalmente cuando no sabemos cuantas 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)
```

El bucle se ejecuta siempre una vez y al final se evalúa la condición para decir si se ejecuta otra vez el bucle o se termina su ejecución.

Veamos el ejemplo que escribimos para un bucle WHILE en este otro tipo de bucle.

```
var color
do {
        color = prompt("dame un color (escribe rojo para salir)","")
} while (color != "rojo")
```

Ejemplo de uso de los bucles while

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. Será necesario utilizar el bucle WHILE porque no sabemos exactamente el número de iteraciones que tendremos que realizar (dependerá de los valores aleatorios que se vayan obteniendo).

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

Así pues, existen dos instrucciones que se pueden usar en de las distintas estructuras de control y principalmente en los bucles, que te servirán para controlar dos tipos de situaciones. **Son las instrucciones break y continue**.:

- **break:** Significa detener la ejecución de un bucle y salirse de él.
- **continue:** Sirve para detener la iteración actual y volver al principio del bucle para realizar otra iteración, si corresponde.

Break

Se detiene un bucle saliéndose de él y dejarlo todo como está para continuar con el flujo del programa inmediatamente después del bucle.

```
for (i=0;i<10;i++){
        document.write (i)
        escribe = prompt("dime si continuo preguntando...", "si")
        if (escribe == "no")
```

```
        break
}
```

Este ejemplo escribe los números del 0 al 9 y en cada iteración del bucle pregunta al usuario si desea continuar. Si el usuario dice cualquier cosa continua, excepto cuando dice "no", situación en la cual se sale del bucle y deja la cuenta por donde se había quedado.

Continue

Sirve para volver al principio del bucle en cualquier momento, sin ejecutar las líneas que haya por debajo de la palabra continue.

```
var i=0
while (i<7){
        incrementar = prompt("La cuenta está en " + i + ", dime si
incremento", "si")
        if (incrementar == "no")
        continue
        i++
}
```

Este ejemplo, en condiciones normales contaría hasta desde i=0 hasta i=7, pero cada vez que se ejecuta el bucle pregunta al usuario si desea incrementar la variable o no. Si introduce "no" se ejecuta la sentencia continue, con lo que se vuelve al principio del bucle sin llegar a incrementar en 1 la variable i, ya que se ignorarían las sentencias que hayan por debajo del continue.

Ejemplo adicional de la sentencia break

Un ejemplo más práctico sobre estas instrucciones se puede ver a continuación. Se trata de un bucle FOR planeado para llegar hasta 1.000 pero que lo vamos a parar con break cuando lleguemos a 333.

```
for (i=0;i<=1000;i++){
        document.write(i + "<br>")
        if (i==333)
        break;
}
```

Anidar un bucle

Consiste en meter ese bucle dentro de otro.

Un bucle anidado tiene una estructura como la que sigue. Vamos a tratar de explicarlo a la vista de estas líneas:

```
for (i=0;i<10;i++){
        for (j=0;j<10;j++) {
        document.write(i + "-" + j)
        }
}
```

La ejecución funcionará de la siguiente manera. Para empezar, se inicializa el primer bucle, con lo que la variable i valdrá 0 y a continuación se inicializa el segundo bucle, con lo que la variable j valdrá también 0. En cada iteración se imprime el valor de la variable i, un guión ("-") y el valor de la variable j, como las dos variables valen 0, se imprimirá el texto "0-0" en la página web.

Debido al flujo del programa en esquemas de anidación como el que hemos visto, el bucle que está anidado (más hacia dentro) es el que más veces se ejecuta.

FUNCIONES EN JAVASCRIPT

Qué es una función

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.

Ya hemos visto alguna función en nuestros sencillos ejemplos anteriores. Por ejemplo, cuando hacíamos un document.write() en realidad estábamos llamando a la función write() asociada al documento de la página, que escribe un texto en la página.

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 en 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>")
}
```

Simplemente escribe en la página un texto.

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()
```

En principio, podemos colocar las funciones en cualquier parte de la página, siempre entre etiquetas <SCRIPT>, claro está. No obstante, existe una limitación a la hora de colocarla con relación a los lugares desde donde se la llame. Te adelantamos que lo más fácil es colocar la función antes de cualquier llamada a la misma y así seguro que nunca nos equivocaremos.

Existen dos opciones posibles para colocar el código de una función:

a) Colocar la función en el mismo bloque de script: En concreto, la función se puede definir en el bloque <SCRIPT> donde esté la llamada a la función, aunque es indiferente si la llamada se encuentra antes o después del código de la función, dentro del mismo bloque <SCRIPT>.

```
<SCRIPT>
miFuncion()
function miFuncion(){
        //hago algo...
        document.write("Esto va bien")
}
</SCRIPT>
```

Funciona bien porque la función está declarada en el mismo bloque que su llamada.

b) Colocar la función en otro bloque de script: También es válido que la función se encuentre en un bloque <SCRIPT> anterior al bloque donde está la llamada.

```
<HTML>
<HEAD>
   <TITLE>MI PÁGINA</TITLE>
<SCRIPT>
function miFuncion(){
        //hago algo...
        document.write("Esto va bien")
}
</SCRIPT>
</HEAD>
<BODY>

<SCRIPT>
miFuncion()
</SCRIPT>

</BODY>
</HTML>
```

Como se puede comprobar, las funciones están en la cabecera de la página (dentro del HEAD). Éste es un lugar excelente donde colocarlas, porque se supone que en la cabecera no se van a utilizar todavía y siempre podremos disfrutar de ellas en el cuerpo porque sabemos seguro que ya han sido declaradas.

Para que quede claro este asunto de la colocación de funciones veamos el siguiente ejemplo, **que daría un error**. Examina atentamente el código siguiente, que lanzará un error, debido a que hacemos una llamada a una función que se encuentra declarada en un bloque `<SCRIPT>` posterior.

```
<SCRIPT>
miFuncion()
</SCRIPT>

<SCRIPT>
function miFuncion(){
        //hago algo...
        document.write("Esto va bien")
}
</SCRIPT>
```

Parámetros

Los parámetros son los valores de entrada que recibe una función.

Por poner un ejemplo sencillo de entender, una función que realizase una suma de dos números tendría como parámetros a esos dos números. Los dos números son la entrada, así como la salida sería el resultado de la suma.

Veamos un ejemplo anterior en el que creábamos una función para mostrar un mensaje de bienvenida en la página web, pero al que ahora le vamos a pasar un parámetro que contendrá el nombre de la persona a la que hay que saludar.

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

La variable donde recibimos el parámetro tendrá vida durante la ejecución de la función y dejará de existir cuando la función termine su ejecución.

Para llamar a una función que tiene parámetros se coloca entre paréntesis el valor del parámetro. Para llamar a la función del ejemplo habría que escribir:

```
escribirBienvenida("Alberto García")
```

Al llamar a la función así, el parámetro nombre toma como valor "Alberto García" y al escribir el saludo por pantalla escribirá "Hola Alberto García" entre etiquetas `<H1>`.

Los parámetros pueden recibir cualquier tipo de datos, numérico, textual, booleano.

Múltiples parámetros

Una función puede recibir tantos parámetros como queramos y para expresarlo se colocan los nombres de los parámetros separados por comas, dentro de los paréntesis.

```
function escribirBienvenida(nombre,colorTexto){
    document.write("<FONT color='" + colorTexto + "'>")
    document.write("<H1>Hola " + nombre + "</H1>")
    document.write("</FONT>")
}
```

Llamaríamos a la función con esta sintaxis. Entre los paréntesis colocaremos los valores de los parámetros.

```
var miNombre = "Pepe"
var miColor = "red"
escribirBienvenida(miNombre,miColor)
```

He colocado entre los paréntesis dos variables en lugar de dos textos entrecomillados.

Los parámetros se pasan por valor

Esto quiere decir que estamos pasando valores y no variables.

```
function pasoPorValor(miParametro){
        miParametro = 32
        document.write("he cambiado el valor a 32")
}
var miVariable = 5
pasoPorValor(miVariable)
document.write ("el valor de la variable es: " + miVariable)
```

En el ejemplo tenemos una función que recibe un parámetro y que modifica el valor del parámetro asignándole el valor 32. También tenemos una variable, que inicializamos a 5 y posteriormente llamamos a la función pasándole esta variable como parámetro. Como dentro de la función modificamos el valor del parámetro podría pasar que la variable original cambiase de valor, pero como los parámetros no modifican el valor original de las variables, ésta no cambia de valor.

De este modo, una vez ejecutada la función, al imprimir en pantalla el valor de miVariable se imprimirá el número 5, que es el valor original de la variable, en lugar de 32 que era el valor con el que habíamos actualizado el parámetro.

En Javascript sólo se pueden pasar las variables por valor.

Devolución de valores en las funciones

Las funciones en Javascript también pueden retornar valores. De modo que, al invocar una función, se podrá realizar acciones y ofrecer un valor como salida.

Por ejemplo, una función que calcula el cuadrado de un número tendrá como entrada a ese número y como salida tendrá el valor resultante de hallar el cuadrado de ese número.

Veamos un ejemplo de función que calcula la media de dos números. La función recibirá los dos números y retornará el valor de la media.

```
function media(valor1,valor2){
        var resultado
        resultado = (valor1 + valor2) / 2
        return resultado
}
```

Para especificar el valor que retornará la función se utiliza la palabra **return** seguida de el valor que se desea devolver. En este caso se devuelve el contenido de la variable resultado, que contiene la media calculada de los dos números.

Quizás nos preguntemos ahora cómo recibir un dato que devuelve una función. Realmente en el código fuente de nuestros programas podemos invocar a las funciones en el lugar que deseemos. Cuando una función devuelve un valor simplemente se sustituye la llamada a la función por ese valor que devuelve. Así pues, para almacenar un valor de devolución de una función, tenemos que asignar la llamada a esa función como contenido en una variable, y eso lo haríamos con el operador de asignación =.

Para ilustrar esto se puede ver este ejemplo, que llamará a la función media() y guardará el resultado de la media en una variable para luego imprimirla en la página.

```
var miMedia
miMedia = media(12,8)
document.write (miMedia)
```

Múltiples return

En realidad, en Javascript las funciones sólo pueden devolver un valor, por lo que en principio no podemos hacer funciones que devuelvan dos datos distintos.

Ahora bien, aunque sólo podamos devolver un dato, en una misma función podemos colocar más de un return. Como decimos, sólo vamos a poder retornar una cosa, pero dependiendo de lo que haya sucedido en la función podrá ser de un tipo u otro, con unos datos u otros.

En esta función podemos ver un ejemplo de utilización de múltiples return. Se trata de una función que devuelve un 0 si el parámetro recibido era par y el valor del parámetro si este era impar.

```
function multipleReturn(numero){
        var resto = numero % 2
        if (resto == 0)
        return 0
        else
        return numero
}
```

Para averiguar si un número es par hallamos el resto de la división al dividirlo entre 2. Si el resto es cero es que era par y devolvemos un 0, en caso contrario -el número es impar- devolvemos el parámetro recibido.

Ámbito de las variables en funciones

Dentro de las funciones podemos declarar variables. Sobre este asunto debemos de saber que todas las variables declaradas en una función son locales a esa función, es decir, sólo tendrán validez durante la ejecución de la función.

Podría darse el caso de que podemos declarar variables en funciones que tengan el mismo nombre que una variable global a la página. Entonces, dentro de la función, la variable que tendrá validez es la variable local y fuera de la función tendrá validez la variable global a la página.

En cambio, si no declaramos las variables en las funciones se entenderá por javascript que estamos haciendo referencia a una variable global a la página, de modo que si no está creada la variable la crea, pero siempre global a la página en lugar de local a la función.

Veamos el siguiente código.

```
function variables_glogales_y_locales(){
   var variableLocal = 23
   variableGlobal = "qwerty"
}
```

En este caso variableLocal es una variable que se ha declarado en la función, por lo que será local a la función y sólo tendrá validez durante su ejecución. Por otra parte, variableGlobal no se ha llegado a declarar (porque antes de usarla no se ha utilizado la palabra var para declararla). En este caso la variable variableGlobal es global a toda la página y seguirá existiendo, aunque la función finalice su ejecución.

Funciones incorporadas en Javascript

eval(string) Recibe una cadena de caracteres y la ejecuta como si fuera una sentencia de Javascript.

parseInt(cadena,base) Recibe una cadena y una base. Devuelve un valor numérico resultante de convertir la cadena en un número en la base indicada.

parseFloat(cadena) Convierte la cadena en un número y lo devuelve.

escape(carácter) Devuelve un el carácter que recibe por parámetro en una codificación ISO Latin 1.

unescape(carácter) Hace exactamente lo opuesto a la función escape.

isNaN(número) Devuelve un booleano dependiendo de lo que recibe por parámetro. Si no es un número devuelve un true, si es un numero devuelve false.

Existen muchas otras funciones lo que ocurre es que están asociadas a objetos. Existen funciones de cadenas de caracteres, que están asociadas a objetos string, funciones para

trabajo con cálculos matemáticos avanzados, que están asociadas a la clase Math, funciones para trabajo con el objeto de la ventana del navegador, con el documento, etc.

Ejemplos de uso de las funciones incorporadas en Javascript

Función eval

```
var miTexto = "3 + 5"
eval("document.write(" + miTexto +")")
```

Primero creamos una variable con un texto, en la siguiente línea utilizamos la función eval y como parámetro le pasamos una instrucción javascript para escribir en pantalla. Si concatenamos los strings que hay dentro de los paréntesis de la función eval nos queda esto.

```
document.write(3 + 5)
```

La función eval ejecuta la instrucción que se le pasa por parámetro, así que ejecutará esta sentencia, lo que dará como resultado que se escriba un 8 en la página web. Primero se resuelve la suma que hay entre paréntesis, con lo que obtenemos el 8 y luego se ejecuta la instrucción de escribir en pantalla.

Función parseInt

Esta función recibe un número, escrito como una cadena de caracteres, y un número que indica una base.

Las distintas bases que puede recibir la función son 2, 8, 10 y 16. Si no le pasamos ningún valor como base la función interpreta que la base es decimal. El valor que devuelve la función siempre tiene base 10, de modo que si la base no es 10 convierte el número a esa base antes de devolverlo.

Veamos una serie de llamadas a la función parseInt para ver lo que devuelve y entender un poco más la función.

```
document.write (parseInt("34"))
```

Devuelve el numero 34

```
document.write (parseInt("101011",2))
```

Devuelve el numero 43

```
document.write (parseInt("34",8))
```

Devuelve el numero 28

```
document.write (parseInt("3F",16))
```

Devuelve el numero 63

Esta función se utiliza en la práctica para un montón de cosas distintas en el manejo con números, por ejemplo, obtener la parte entera de un decimal.

```
document.write (parseInt("3.38"))
```

Devuelve el numero 3

También es muy habitual su uso para saber si una variable es numérica, pues si le pasamos un texto a la función que no sea numérico nos devolverá NaN (Not a Number) lo que quiere decir que No es un Número.

```
document.write (parseInt("desarrolloweb.com"))
```

Devuelve el numero NaN

Este mismo ejemplo es interesante con una modificación, pues si le pasamos una combinación de letras y números nos dará lo siguiente.

```
document.write (parseInt("16XX3U"))
```

Devuelve el numero 16

```
document.write (parseInt("TG45"))
```

Devuelve el numero NaN

Como se puede ver, la función intenta convertir el string en número y si no puede devuelve NaN.

Función isNaN

Esta función devuelve un booleano dependiendo de si lo que recibe es un número o no. Lo único que puede recibir es un número o la expresión NaN. Si recibe un NaN devuelve true y si recibe un número devuelve false.

La función suele trabajar en combinación con la función parseInt o parseFloat, para saber si lo que devuelven estas dos funciones es un número o no.

```
miInteger = parseInt("A3.6")
isNaN(miInteger)
```

En la primera línea asignamos a la variable miInteger el resultado de intentar convertir a entero el texto A3.6. Como este texto no se puede convertir a número la función parseInt devuelve NaN. La segunda línea comprueba si la variable anterior es NaN y como si que lo es devuelve un true.

```
miFloat = parseFloat("4.7")
isNaN(miFloat)
```

En este ejemplo convertimos un texto a número con decimales. El texto se convierte perfectamente porque corresponde con un número. Al recibir un número la función isNaN devuelve un false.

ARRAYS

El primer paso para utilizar un array es crearlo. Esta es la sentencia para crear un objeto array:

```
var miArray = new Array()
```

Esto crea un array en la página que esta ejecutándose. El array se crea sin ningún contenido, es decir, no tendrá ninguna casilla o compartimiento creado. También podemos crear el array Javascript especificando el número de compartimentos que va a tener.

```
var miArray = new Array(10)
```

En este caso indicamos que el array va a tener 10 posiciones, es decir, 10 casillas donde guardar datos.

Es importante que nos fijemos que la palabra Array en código Javascript se escribe con la primera letra en mayúscula.

Tanto se indique o no el número de casillas del **array javascript**, podemos introducir en el array cualquier dato. Si la casilla está creada se introduce simplemente y si la casilla no estaba creada se crea y luego se introduce el dato, con lo que el resultado final es el mismo. Esta creación de casillas es dinámica y se produce al mismo tiempo que los scripts se ejecutan. Veamos a continuación cómo introducir valores en nuestros arrays.

```
miArray[0] = 290
miArray[1] = 97
miArray[2] = 127
```

Se introducen indicando entre corchetes el índice de la posición donde queríamos guardar el dato. En este caso introducimos 290 en la posición 0, 97 en la posición 1 y 127 en la 2.

Los arrays en Javascript empiezan siempre en la posición 0, así que un array que tenga por ejemplo 10 posiciones, tendrá casillas de la 0 a la 9. Para recoger datos de un array lo hacemos igual: poniendo entre corchetes el índice de la posición a la que queremos acceder. Veamos cómo se imprimiría en la pantalla el contenido de un array.

```
var miArray = new Array(3)

miArray[0] = 155
miArray[1] = 4
miArray[2] = 499

for (i=0;i<3;i++){
        document.write("Posición " + i + " del array: " + miArray[i])
        document.write("<br>")
}
```

Hemos creado un array con tres posiciones, luego hemos introducido un valor en cada una de las posiciones del array y finalmente las hemos impreso.

Tipos de datos en los arrays

En las casillas de los arrays podemos guardar datos de cualquier tipo. Podemos ver un array donde introducimos datos de tipo carácter.

```
miArray[0] = "Hola"
miArray[1] = "a"
miArray[2] = "todos"
```

Incluso, en Javascript podemos guardar distintos tipos de datos en las casillas de un mismo array. Es decir, podemos introducir números en unas casillas, textos en otras, boleanos o cualquier otra cosa que deseemos.

```
miArray[0] = "desarrolloweb.com"
miArray[1] = 1275
miArray[1] = 0.78
miArray[2] = true
```

Declaración e inicialización resumida de Arrays

En Javascript tenemos a nuestra disposición una manera resumida de declarar un array y cargar valores en un mismo paso. Fijémonos en el código siguiente:

```
var arrayRapido = [12,45,"array inicializado en su declaración"]
```

Como se puede ver, se está definiendo una variable llamada arrayRapido y estamos indicando en los corchetes varios valores separados por comas. Esto es lo mismo que haber declarado el array con la función Array() y luego haberle cargado los valores uno a uno.

Todos los arrays, aparte de almacenar el valor de cada una de sus casillas, también almacenan el número de posiciones que tienen. Para ello utilizan una propiedad del objeto array, la propiedad **length.** que almacena un número igual al número de casillas que tiene el array.

Para acceder a una propiedad de un objeto se ha de utilizar el operador punto. Se escribe el nombre del array que queremos acceder al número de posiciones que tiene, sin corchetes ni paréntesis, seguido de un punto y la palabra length.

```
var miArray = new Array()

miArray[0] = 155
miArray[1] = 499
miArray[2] = 65

document.write("Longitud del array: " + miArray.length)
```

Este código imprimiría en pantalla el número de posiciones del array, que en este caso es 3. Recordamos que un array con 3 posiciones abarca desde la posición 0 a la 2.

Es muy habitual que se utilice la propiedad length para poder recorrer un array por todas sus posiciones. Para ilustrarlo vamos a ver un ejemplo de recorrido por este array para mostrar sus valores.

```
for (i=0;i<miArray.length;i++){
        document.write(miArray[i])
}
```

Hay que fijarse que el bucle for se ejecuta siempre que i valga menos que la longitud del array, extraída de su propiedad length.

Los arrays multidimensionales son unas estructuras de datos que almacenan los valores en más de una dimensión. Los arrays de 2 dimensiones guardan sus valores, por decirlo de alguna manera, en filas y columnas y por ello necesitaremos dos índices para acceder a cada una de sus posiciones.

En Javascript no existe un auténtico objeto array-multidimensinal. Para utilizar estas estructuras podremos definir arrays que donde en cada una de sus posiciones habrá otro array.

En este ejemplo vamos a crear un array de dos dimensiones donde tendremos por un lado ciudades y por el otro la temperatura media que hace en cada una durante de los meses de invierno.

```
var temperaturas_medias_ciudad0 = new Array(3)
temperaturas_medias_ciudad0[0] = 12
temperaturas_medias_ciudad0[1] = 10
temperaturas_medias_ciudad0[2] = 11

var temperaturas_medias_ciudad1 = new Array (3)
temperaturas_medias_ciudad1[0] = 5
temperaturas_medias_ciudad1[1] = 0
temperaturas_medias_ciudad1[2] = 2

var temperaturas_medias_ciudad2 = new Array (3)
temperaturas_medias_ciudad2[0] = 10
temperaturas_medias_ciudad2[1] = 8
temperaturas_medias_ciudad2[2] = 10
```

Con las anteriores líneas hemos creado tres arrays de 1 dimensión y tres elementos, como los que ya conocíamos. Ahora crearemos un nuevo array de tres elementos e introduciremos dentro de cada una de sus casillas los arrays creados anteriormente, con lo que tendremos un array de arrays, es decir, un array de 2 dimensiones.

```
var temperaturas_cuidades = new Array (3)
temperaturas_cuidades[0] = temperaturas_medias_ciudad0
temperaturas_cuidades[1] = temperaturas_medias_ciudad1
temperaturas_cuidades[2] = temperaturas_medias_ciudad2
```

Vemos que para introducir el array entero hacemos referencia al mismo sin paréntesis ni corchetes, sino sólo con su nombre. El array temperaturas_cuidades es nuestro array bidimensinal.

También es interesante ver cómo se realiza un recorrido por un array de dos dimensiones.

El método para hacer un recorrido dentro de otro es colocar un bucle dentro de otro, lo que se llama un bucle anidado. En este ejemplo vamos a meter un bucle FOR dentro de otro. Además, vamos a escribir los resultados en una tabla, lo que complicará un poco el script, pero así podremos ver cómo construir una tabla desde Javascript a medida que realizamos el recorrido anidado al bucle.

```
document.write("<table width=200 border=1 cellpadding=1
cellspacing=1>");
for (i=0;i<temperaturas_cuidades.length;i++){
        document.write("<tr>")
        document.write("<td><b>Ciudad " + i + "</b></td>")
        for (j=0;j<temperaturas_cuidades[i].length;j++){
        document.write("<td>" + temperaturas_cuidades[i][j] + "</td>")
        }
        document.write("</tr>")
}
document.write("</table>")
```

La primera acción consiste en escribir la cabecera de la tabla, es decir, la etiqueta <TABLE> junto con sus atributos. Con el primer bucle realizamos un recorrido a la primera dimensión del array y utilizamos la variable i para llevar la cuenta de la posición actual. Por cada iteración de este bucle escribimos una fila y para empezar la fila abrimos la etiqueta <TR>. Además, escribimos en una casilla el numero de la ciudad que estamos recorriendo en ese momento. Posteriormente ponemos otro bucle que va recorriendo cada una de las casillas del array en su segunda dimensión y escribimos la temperatura de la ciudad actual en cada uno de los meses, dentro de su etiqueta <TD>. Una vez que acaba el segundo bucle se han impreso las tres temperaturas y por lo tanto la fila está terminada. El primer bucle continúa repitiéndose hasta que todas las ciudades están impresas y una vez terminado cerramos la tabla.

Inicialización de arrays

Una manera de inicializar sus valores a la vez que lo declaramos, así podemos realizar de una manera más rápida el proceso de introducir valores en cada una de las posiciones del array.

Veámoslo con un ejemplo que crea un array e inicializarlo al mismo tiempo con los nombres de los días de la semana.

```
var diasSemana = new
Array("Lunes","Martes","Miércoles,","Jueves","Viernes","Sábado","Domin
go")
```

El array se crea con 7 casillas, de la 0 a la 6 y en cada casilla se escribe el día de la semana correspondiente (Entre comillas porque es un texto).

Ahora vamos a declarar el array bidimensional del ejemplo anterior de las temperaturas de las ciudades en los meses en una sola línea, introduciendo los valores a la vez.

```
var temperaturas_cuidades = new Array(new Array (12,10,11), new
Array(5,0,2),new Array(10,8,10))
```