1. Operadores
2. TypeOf / Prompt / Alert
3. Condicionales
4. Switch
5. Bucle For
6. Bucle While
7. Break y Continue
8. Bucles Anidados

Operadores Javascript

Al desarrollar programas en cualquier lenguaje se utilizan los operadores, que sirven para hacer los cálculos y operaciones necesarios para llevar a cabo tus objetivos. Hasta el menor de los programas imaginables necesita de los operadores para realizar cosas, ya que un programa que no realizase operaciones, sólo se limitaría a hacer siempre lo mismo.

Es el resultado de las operaciones lo que hace que un programa varíe su comportamiento según los datos que tenga para trabajar y nos ofrezca resultados que sean relevantes para el usuario que lo utilice. Existen operaciones más sencillas o complejas, que se pueden realizar con operandos de distintos tipos, como números o textos.

Ejemplos de uso de operadores

Antes de enumerar los distintos tipos de operadores vamos ejemplos de éstos. En el primer ejemplo vamos a realizar una suma utilizando el operador suma.

```
3 + 5
```

Esta es una expresión muy básica que no tiene mucho sentido ella sola. Hace la suma entre los dos operandos número 3 y 5, pero no sirve de mucho porque no se hace nada con el resultado.

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.

Los operadores se pueden clasificar según el tipo de acciones que realizan: Operadores

aritméticos

Son los utilizados para la realización de operaciones matemáticas simples como la suma, resta o multiplicación. En javascript son los siguientes:

+ Suma de 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

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

Sirven para asignar valores a las variables, ya hemos utilizado en ejemplos anteriores el operador de asignación =, pero hay otros operadores de este tipo, que provienen del lenguaje C y que muchos de los lectores ya conocerán.

= 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, operadores lógicos y operadores condicionales.

Operadores de cadenas

Las cadenas de caracteres, o variables de texto, también tienen sus propios operadores para realizar acciones típicas sobre cadenas. Aunque javascript sólo tiene un operador para cadenas se pueden realizar otras acciones con una serie de funciones predefinidas en el lenguaje que veremos más adelante.

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

Ejemplo cadena1
= "hola"

cadena2 = "mundo"

cadenaConcatenada = cadena1 + cadena2 //cadena concatenada vale "holamundo"

Un detalle importante que se puede ver en este caso es que el operador + sirve para dos usos distintos, si sus operandos son números los suma, pero si se trata de cadenas las concatena. Esto pasa en general con todos los operadores que se repiten en el lenguaje, javascript es suficientemente listo para entender que tipo de operación realizar mediante una comprobación de los tipos que están implicados en élla.

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.

No se encuentran muchas operaciones típicas a realizar con cadenas, para las cuales no existen operadores. Es porque esas funcionalidades se obtienen a través de la clase String de Javascript.

Operadores condicionales

Sirven para realizar expresiones condicionales todo lo complejas que deseemos. 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.

Los operadores condicionales se utilizan en las expresiones condicionales para tomas de decisiones.

Seguidamente 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

Operadores lógicos

Estos operadores sirven para realizar operaciones lógicas, que son aquellas que dan como resultado un verdadero o un falso, y se utilizan para tomar decisiones en nuestros scripts. En vez de trabajar con números, para realizar este tipo de operaciones se utilizan operandos boleanos, que conocimos anteriormente, que son el verdadero (true) y el falso (false). Los operadores lógicos relacionan los operandos boleanos para dar como resultado otro operando boleano.

Los operadores lógicos de JavaScript son:

NOT (!), AND (&&) OR (||).

El operador **NOT** produce falso si su operando es verdadero, y viceversa.

El operador **AND** produce verdadero sólo si ambos operandos son verdaderos; si cualquiera de los operando es falso, produce falso.

El operador **OR** produce verdadero si cualquiera de los operandos es verdadero, y falso en caso de que los dos operandos sean falsos.

Operadores Lógicos Tabla de Verdad

La tabla de verdad es la que se emplea en la mayoría de los lenguajes de programación:

AND (&&)		
True	True	True
True	False	False
False	True	False
False	False	False

OR (II)		
True	True	True
True	False	True
False	True	True
False	False	False

NOT (!)	
True	False
False	True

Ahora que conocemos la tabla de verdad, es momento de emplearla en lo que es le lenguaje de programación, para ello veremos algún ejemplo practico. Retomemos a lo que eran los comparadores relacionales en el cual utilizábamos los símbolos <, >, ==, etc..

Miremos el siguiente código:

```
console.log(4>2);
```

Esto daba como resultado el Booleano True

Si teníamos el siguiente código:

```
console.log(2>4);
```

En este caso daba como resultado False.

En ambos ejemplos solo utilizamos una relación pero si utilizamos los operadores lógicos podremos comparar dos valores o mas, con el cual nos dará un resultado booleano.

Ejemplo Practico Operador AND (&&)

En principio utilizaremos el operador lógico AND (&&), recordemos que este solo da valor verdadero (True) si al comparar dos valores, los mismos dan como resultado true

Ejemplo:

AND (&&)		
True	True	True
True	False	False
False	True	False
False	False	False

```
console.log(4>2 && 5>1)
```

En este caso por un lado compara el resultado del valor 4>2 = True y 5>1 =True, y siguiendo la regla de la tabla lógica, cuando dos valores son del tipo True el resultado final es True, caso contrario seria false, veamos un ejemplo practico en el cual nos de false.

```
console.log(4>2 && 1>3)
```

En ese ejemplo el valor de la primera relación 4>2 es True pero la segunda relación 1>3 es False, el resultado general es False ya que mirando la tabla AND, en el caso que alguno de los valores sea False el resultado general es False.

Ejemplo Practico Operador OR (||)

Ahora haremos un ejemplo practico para ver como trabaja el operador lógico OR, para ellos recordemos la tabla

OR (II)		
True	True	True
True	False	True
False	True	True
False	False	False

En este caso solo da valor verdadero si alguno de los resultados da True, para ello podemos comprobarlo con el siguiente ejemplo practico.

Ejemplo:

```
console.log(4>2 || 3>1)
```

En el ejemplo anterior las dos operaciones de relación dan valor True, 4>2 = True y 3>1 = True, probemos con otro ejemplo distinto.

Ejemplo:

```
console.log(4>2 || 3<1)
```

En el ejemplo anterior da como resultado general True ya que cumple con que alguno de los valores es True, 4>2 = true y 3<1 = false

Ejemplo Practico Operador NOT (!)

El operador NOT lo que hace es negar la expresión como se ve en su tabla:

En este caso veremos cómo trabaja

NOT (!)	
True	False
False	True

el operador:

Ejemplo:

```
console.log(!4>2)
```

Como se ve en el ejemplo anterior, la expresión en si 4>2 da como resultado True pero al tener el signo de negación NOT (!), lo que hace es invertir el resultado, en este caso da False.

Ejemplo:

```
console.log(!1>3)
```

En este ultimo ejemplo el resultado general es false ya que 1 no es mayor a 3 pero al tener el operador NOT, da como resultado True

En el siguiente grupo de operadores veremos sobre los operadores condicionales, que se pueden utilizar junto con los operadores lógicos para realizar sentencias todo lo complejas que necesitemos.

Por ejemplo:

```
if (x==2 && y!=3){
  //SI la variable x vale 2 y la variable y es distinta de tres
}
```

En la expresión condicional anterior estamos evaluando dos comprobaciones que se relacionan con un operador lógico. Por una parte x==2 devolverá un true en caso que la variable x valga 2 y por otra, y!=3 devolverá un true cuando la variable y tenga un valor distinto de 3. Ambas comprobaciones devuelven un boleano cada una, que luego se le aplica el operador lógico && para comprobar si ambas comprobaciones se cumplieron al mismo tiempo.

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

Precedencia de los operadores

La evaluación de una sentencia de las que hemos visto en los ejemplos anteriores es bastante sencilla y fácil de interpretar, pero cuando en una sentencia entran en juego multitud de operadores distintos puede puede ser dificil interpretar qué operadores son los que se ejecutan antes que otros.

Para marcar unas pautas en la evaluación de las sentencias y que estas se ejecuten siempre igual y con sentido común existe la precedencia de operadores, que no es más que el orden por el que se irán ejecutando las operaciones que ellos representan. En un principio todos los operadores se evalúan de izquierda a derecha, pero existen unas normas adicionales, por las que determinados operadores se evalúan antes que otros. Muchas de estas reglas de precedencia se basan de normas matemáticas y son comunes a otros lenguajes, 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

<< >> >>> Cambios a nivel de bit

< <= > >= Operadores condicionales

== != Operadores condicionales de igualdad y desigualdad &
^ | Lógicos a nivel de bit

&& || Lógicos booleanos

En los siguientes ejemplos podemos ver cómo las expresiones podrían llegar a ser confusas, pero con la tabla de precedencia de operadores podremos entender sin errores cuál es el orden por el que se ejecutan.

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

De todos modos, el uso de los paréntesis puede ahorrarnos mucho tiempo de desarrollo cuando veamos poco claro el orden con el que se ejecutarán las sentencias, ya que podemos utilizarlos y así forzar que se evalúe antes el segmento de expresión que se encuentra dentro de los paréntesis.

Operador typeof - control de tipos

Hemos podido comprobar que, para determinados operadores, es importante el tipo de datos que están manejando, puesto que si los datos son de un tipo se realizarán operaciones distintas que si son de otro.

Por ejemplo, cuando se utiliza el operador +, si se trataba de números los sumaba, pero si se trataba de cadenas de caracteres las concatenaban, entonces, el tipo de los datos determina que nuestras operaciones se realicen tal como esperábamos.

Para comprobar el tipo de un dato se puede utilizar el operador typeof, que devuelve una cadena de texto que describe el tipo del operador que estamos comprobando.

```
var boleano = true var
numerico = 22

var numerico_flotante = 13.56
var texto = "mi texto"

var fecha = new Date()

document.write("<br>El tipo de boleano es: " + typeof boleano)
document.write("<br>El tipo de numerico es: " + typeof numerico)
document.write("<br>El tipo de numerico_flotante es: " + typeof numerico_flotante)
document.write("<br>El tipo de texto es: " + typeof texto)

document.write("<br>El tipo de fecha es: " + typeof fecha)
```

Si ejecutamos este script obtendremos que en la página se escribirá el siguiente texto: El

tipo de boleano es: boolean

El tipo de numerico es: number

El tipo de numerico_flotante es: number
El tipo de texto es: string

El tipo de fecha es: object

En este ejemplo se imprimen los distintos tipos de las variables. Estos pueden ser:

boolean, para los datos boleanos. (True o false)

number, para los numéricos.

string, para las cadenas de caracteres.
object, para los objetos.

function, para las funciones.

undefined, para variables declaradas a las que no se les ha asignado valores.

Es de destacar que los números, ya tengan o no parte decimal, son siempre del tipo de datos numérico.

La variable fecha es un objeto de la clase Date(), que se utiliza para el manejo de fechas en los scripts y la veremos más adelante, así como los objetos.

Si el tipo de una variable es undefined, puede utilizarse para obtener la respuesta sobre si la variable ha sido inicializada o no.

```
let variable;
if(typeof variable == 'undefined') {
        console.log('La variable vale undefined, no tiene valor definido);
}
```

Función Alert

En esta unidad veras algunas funciones comunes con las cuales hacemos una breve introducción, solamente aprenderemos a invocarlas, en capítulos próximos aprenderemos a crearlas, veremos cómo llamarlas y cuál es su función principal. También debes saber que una función es un conjunto de sentencias que realizan una tarea en particular.

Esta función es un método del objeto Window, es la encargada de mostrar una pequeña ventana de aviso en la pantalla, así, si se requiere que aparezca un mensaje cuando ocurra determinada acción en el programa, podemos hacer uso de esta función. La función **alert** recibe como parámetro el mensaje que se debe mostrar en la ventana.

Ejemplo:

```
alert ( "Hola Mundo JavaScript" )
```

También podremos pasar una variable a la función alert para que la muestre por pantalla:

```
var texto = 'Hola mundo JavaScript';
alert (texto);
var numero = 10;
alert (numero);
var concateno = 50;
alert('Se esta concatenando una variable ' + concateno);
```

Prompt

Al igual que la función **alert**, la función **prompt** es también un método del objeto Window. Esta función se utiliza cuando el usuario ingresa datos por medio del teclado. Con esta función

aparece una ventana en la pantalla, con un espacio para el valor que se debe ingresar y un botón **aceptar** para que la información sea guardada.

Ejemplo:

```
prompt ('Ingrese su nombre');
```

También podremos guardar el valor que la función prompt pide por la ventana en una variable, como mostramos a continuación:

```
var nombre = prompt ('Ingrese su nombre');
```

Condicionales

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 y 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. 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
  //...
```

}

Las 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.

Por ejemplo:

```
if (llueve) alert("Cae
  agua");
```

Sería exactamente igual que este código: if

```
(llueve){
  alert("Cae agua");
}
```

O incluso, igual a este otro:

```
if (llueve) alert("Cae agua");
```

Generalmente, cuando utilizamos las llaves, el código queda bastante más claro, porque se puede apreciar en un rápido vistazo qué instrucciones están dependiendo del caso positivo del if.

Los saltos de línea tampoco son necesarios y se han colocado también para que se vea mejor la estructura. Perfectamente podríamos colocar toda la instrucción IF en la misma línea de código, pero eso no ayudará a que las cosas estén claras.

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).

Ejemplo :

```
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
}
```

Este ejemplo comprueba si tiene crédito para realizar una supuesta compra. Para ello analizamos si el crédito es mayor o igual que el precio del artículo, si es así informo de la compra, introduzco el artículo en el carrito y resto el precio al crédito acumulado. Si el precio del artículo es superior al dinero disponible informo de la situación y mando al navegador a la página donde se muestra su carrito de la compra.

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 boleano. 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")
```

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 boleanos y que devuelvan otro valor boleano.

Son los operadores negación lógica, Y lógico y O lógico. if

```
(bateria < 0.5 && redElectrica == 0)
  document.write("la notebook se va a apagar en segundos")
```

Lo que hacemos es comprobar si la batería de nuestra supuesta notebook es menor que 0.5 (está casi vacía) y también comprobamos si no tiene red eléctrica (desenchufado). Luego, el operador lógico los relaciona con un Y, de modo que si está casi sin batería Y sin red eléctrica, muestro el mensaje que el equipo se va a apagar.

La estructura if es de las más utilizadas en lenguajes de programación, para tomar decisiones en función de la evaluación de una sentencia.

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 podemos anidar IF's y crear el flujo de código necesario para decidir correctamente.

Por ejemplo, para 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")
  }
}
```

El flujo del programa primero evalúa si los dos números son iguales, en caso positivo se muestra un mensaje informando de ello, en caso contrario ya sabemos que son distintos, pero aun debemos averiguar cuál de los dos es mayor. Para eso se hace otra comparación para saber si el primero es mayor

que el segundo. Si esta comparación da resultado positivo mostramos un mensaje que el primero es mayor que el segundo, en caso contrario indico que el primero es menor que el segundo.

Las llaves son en este caso opcionales, y sólo ejecutan una sentencia para cada caso. Además, los saltos de línea y tabuladores son también opcionales en todo caso y nos sirven sólo para ver el código de una manera más ordenada.

Mantener el código bien estructurado y escrito de una manera comprensible es muy importante.

Operador IF

Este operador es un claro ejemplo de ahorro de líneas y caracteres al escribir los scripts. .

Un ejemplo de uso del operador IF se puede ver a continuación.

Variable = (condición) ? valor1 : valor2

Este ejemplo no sólo realiza una comparación de valores, además asigna un valor a una variable. Lo que hace es evaluar la condición (colocada entre paréntesis) y si es positiva asigna el valor1 a la variable y en caso contrario le asigna el valor2. Veamos un ejemplo:

momento = (hora_actual < 12) ? "Antes del mediodía" : "Después del mediodía"

Este ejemplo analiza si la hora actual es menor que 12. Si es así, es antes del mediodía, y asigna "Antes del mediodía" a la variable momento. Si la hora es mayor o igual a 12 es después del mediodía, con lo que se asigna el texto "Después del mediodía" a la variable momento.

Estructura SWITCH de Javascript

Las estructuras de control son la manera con la que se puede dominar el flujo de los programas, para hacer cosas distintas en función de los estados de las variables.

Switch se utiliza 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.

Ejemplo: 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.

Bucle FOR

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 de utilización de este bucle lo podemos ver a continuación, 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.

Como se puede comprobar, en una sola línea podemos indicar muchas cosas distintas y muy variadas, lo que permite una rápida configuración del bucle y una versatilidad enorme.

Por ejemplo si queremos escribir los número del 1 al 1.000 de dos en dos se escribirá el siguiente bucle.

```
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.

No utilizamos las llaves englobando las instrucciones del bucle FOR porque sólo tiene una sentencia y en este caso no es obligado, tal como pasaba con las instrucciones del IF.

Si queremos contar descendentemente del 200 al 10 utilizaríamos este bucle.

```
for (i=200;i>=10;i--)
        document.write(i)
```

En este caso decrementamos en una unidad la variable i en cada iteración, comenzando en el valor 200 y siempre que la variable tenga un valor mayor o igual que 10.

Bucles WHILE y DO WHILE

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.

Sólo se indica, como veremos a continuación, la condición que se tiene que cumplir para que se realice una iteración.

```
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". En cada iteración del bucle se pide un nuevo color al usuario para actualizar la variable color y se termina la iteración, con lo que retornamos al principio del bucle, donde tenemos que volver a evaluar si lo que hay en la variable color es "rojo" y así sucesivamente mientras que no se haya introducido como color el texto "rojo".

Bucle DO...WHILE

El bucle do...while es la última de las estructuras para implementar repeticiones de las que dispone en Javascript y es una variación del bucle while visto anteriormente. 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.

Ejemplo: para un bucle WHILE :

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

Este ejemplo funciona exactamente igual que el anterior, excepto que no tuvimos que inicializar la variable color antes de introducirnos en el bucle. Pide un color mientras que el color introducido es distinto que "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>")
```

}

Break y continue

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 utilizando la palabra break. Detener un bucle significa salirse 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 sentencia que hayan por debajo del continue.

Ejemplo de la sentencia break

Un bucle FOR planeado para llegar hasta 1.000 pero que lo vamos a detener con break cuando lleguemos a 333.

```
for (i=0;i<=1000;i++){

        document.write(i + "<br>")
        if (i==333)

        break;
```

Bucles anidados en Javascript

Anidar un bucle consiste en ingresar ese bucle dentro de otro. La anidación de bucles es necesaria para hacer determinados procesamientos un poco más complejos que los que hemos visto en los ejemplos anteriores.

Un bucle anidado tiene una estructura como la que sigue.

```
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. En este ejemplo, para cada iteración del bucle más externo el bucle anidado se ejecutará por completo una vez, es decir, hará sus 10 iteraciones. En la página web se escribirían estos valores, en la primera iteración del bucle externo y desde el principio:

0-0

0-1

0-2

0-3

0-4

0-5

0-6

0-7

0-8

0-9

Para cada iteración del bucle externo se ejecutarán las 10 iteraciones del bucle interno o anidado. Hemos visto la primera iteración, ahora vamos a ver las siguientes iteraciones del bucle externo. En cada una acumula una unidad en la variable i, con lo que saldrían estos valores.

1-0
1-1
1-2
1-3
1-4
1-5
1-6
1-7
1-8
1-9

Y luego estos:

2-0

2-1

2-2

2-3

2-4

2-5

2-6

2-7

Así hasta que se terminen los dos bucles, que sería cuando se alcanzase el valor 9-9.

Veamos otro ejemplo. Se trata de imprimir en la página las todas las tablas de multiplicar. Del 1 al 9, es decir, la tabla del 1, la del 2, del 3...

```
for (i=1;i<10;i++){
        document.write("<br><b>La tabla del " + i + ":</b><br>")
        for (j=1;j<10;j++) {
        document.write(i + " x " + j + ": ")
        document.write(i*j)
        document.write("<br>")
        }
}
```

Con el primer bucle controlamos la tabla actual y con el segundo bucle la desarrollamos. En el primer bucle escribimos una cabecera, en negrita, indicando la tabla que estamos escribiendo, primero la del 1 y luego las demás en orden ascendente hasta el 9. Con el segundo bucle escribo cada uno de los valores de cada tabla.