**TUTORIAL JAVASCRIPT**

**Introducción**

JavaScript es un lenguaje de programación que se utiliza principalmente para crear páginas web dinámicas.

Una página web dinámica es aquella que incorpora efectos como texto que aparece y desaparece, animaciones, acciones que se activan al pulsar botones y ventanas con mensajes de aviso al usuario.

Técnicamente, JavaScript es un lenguaje de programación interpretado, por lo que no es necesario compilar los programas para ejecutarlos. En otras palabras, los programas escritos con JavaScript se pueden probar directamente en cualquier navegador sin necesidad de procesos intermedios.

El código JavaScript se encierra entre etiquetas **<script>** y se incluye en cualquier parte del documento. Se recomienda meterlo en la etiqueta **<head>**

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

*alert("Hola Mundo");*

*</script>*

Llamamos script a cada uno de los programas, aplicaciones o trozos de código creados con el lenguaje de programación JavaScript. Unas pocas líneas de código forman un script y un archivo de miles de líneas de JavaScript también se considera un script.

**Sintaxis**

La sintaxis de un lenguaje de programación se define como el conjunto de reglas que deben seguirse al escribir el código fuente de los programas para considerarse como correctos para ese lenguaje de programación.

La sintaxis de JavaScript es muy similar a la de otros lenguajes de programación como Java y C. Las normas básicas que definen la sintaxis de JavaScript son las siguientes:

* **No se tienen en cuenta los espacios en blanco y las nuevas líneas**: como sucede con XHTML, el intérprete de JavaScript ignora cualquier espacio en blanco sobrante, por lo que el código se puede ordenar de forma adecuada para entenderlo mejor (tabulando las líneas, añadiendo espacios, creando nuevas líneas, etc.)
* **Se distinguen las mayúsculas y minúsculas**: al igual que sucede con la sintaxis de las etiquetas y elementos XHTML. Sin embargo, si en una página XHTML se utilizan indistintamente mayúsculas y minúsculas, la página se visualiza correctamente, siendo el único problema la no validación de la página. En cambio, si en JavaScript se intercambian mayúsculas y minúsculas el script no funciona.
* **No definimos el tipo de las variables:** al crear una variable, no es necesario indicar el tipo de dato que almacenará. De esta forma, una misma variable puede almacenar diferentes tipos de datos durante la ejecución del script.
* **No es necesario terminar cada sentencia con el carácter de punto y coma (;):** en la mayoría de lenguajes de programación, es obligatorio terminar cada sentencia con el carácter ;. Aunque JavaScript no obliga a hacerlo, es conveniente seguir la tradición de terminar cada sentencia con el carácter del punto y coma (;).
* **Se pueden incluir comentarios:** los comentarios se utilizan para añadir información en el código fuente del programa. Aunque el contenido de los comentarios no se visualiza por pantalla, si que se envía al navegador del usuario junto con el resto del script, por lo que es necesario extremar las precauciones sobre la información incluida en los comentarios.

**Programación – Variables**

Las variables en los lenguajes de programación siguen una lógica similar a las variables utilizadas en otros ámbitos como las matemáticas. Una variable es un elemento que se emplea para almacenar y hacer referencia a otro valor. Gracias a las variables es posible crear "programas genéricos", es decir, programas que funcionan siempre igual independientemente de los valores concretos utilizados.

De la misma forma que si en Matemáticas no existieran las variables no se podrían definir las ecuaciones y fórmulas, en programación no se podrían hacer programas realmente útiles sin las variables.

Las variables en JavaScript se crean mediante la palabra reservada **var**. De esta forma, el ejemplo anterior se puede realizar en JavaScript de la siguiente manera:

*var valor\_1 = 5;*

*var valor \_2 = 6;*

*var resultado = valor \_1 + valor \_2;*

La palabra reservada **var** solamente se debe indicar al definir por primera vez la variable, lo que se denomina declarar una variable. Cuando se utilizan las variables en el resto de instrucciones del script, solamente es necesario indicar su nombre. En otras palabras, en el ejemplo anterior sería un error indicar lo siguiente:

*var valor \_1 = 5;*

*var valor \_2 = 6;*

*var resultado = var valor \_1 + var valor \_2;*

Si cuando se declara una variable se le asigna también un valor, se dice que la variable ha sido inicializada. En JavaScript no es obligatorio inicializar las variables, ya que se pueden declarar por una parte y asignarles un valor posteriormente. Por tanto, el ejemplo anterior se puede rehacer de la siguiente manera:

*var valor\_1;*

*var valor \_2;*

*valor \_1 = 5;*

*valor \_2 = 6;*

*var resultado = valor \_1 + valor \_2;*

Una de las características más sorprendentes de JavaScript para los programadores habituados a otros lenguajes de programación es que tampoco es necesario declarar las variables. En otras palabras, se pueden utilizar variables que no se han definido anteriormente mediante la palabra reservada var. El ejemplo anterior también es correcto en JavaScript de la siguiente forma:

var *valor* \_1 = 3;

var *valor* \_2 = 1;

resultado = *valor* \_1 + *valor* \_2;

La variable resultado no está declarada, por lo que JavaScript crea una variable global (más adelante se verán las diferencias entre variables locales y globales) y le asigna el valor correspondiente. En cualquier caso, se recomienda declarar todas las variables que se vayan a utilizar.

El nombre de una variable también se conoce como **identificador** y debe cumplir las siguientes normas:

Sólo puede estar formado por letras, números y los símbolos $ (dólar) y \_ (guión bajo).

El primer carácter no puede ser un número.

Por tanto, las siguientes variables tienen nombres correctos:

var $numero1;

var \_$letra;

var $$$otroNumero;

var $\_a\_\_$4;

**Tipos de variables**

***Numéricas***

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

*var iva = 16; // variable tipo entero*

*var total = 234.65; // variable tipo decimal*

**Cadenas de texto**

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

*var mensaje = "Bienvenido a nuestro sitio web";*

*var nombreProducto = 'Producto ABC';*

*var letraSeleccionada = 'c';*

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

*/\* El contenido de texto1 tiene comillas simples, por lo que*

*se encierra con comillas dobles \*/*

*var texto1 = "Una frase con 'comillas simples' dentro";*

*/\* El contenido de texto2 tiene comillas dobles, por lo que*

*se encierra con comillas simples \*/*

*var texto2 = 'Una frase con "comillas dobles" dentro';*

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

El mecanismo consiste en sustituir el carácter problemático por una combinación simple de caracteres. A continuación se muestra la tabla de conversión que se debe utilizar:

*Si se quiere incluir... Se debe incluir...*

*Una nueva línea \n*

*Un tabulador \t*

*Una comilla simple \'*

*Una comilla doble \"*

*Una barra inclinada \\*

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

*var texto1 = 'Una frase con \'comillas simples\' dentro';*

*var texto2 = "Una frase con \"comillas dobles\" dentro";*

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

**Arrays**

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

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

var dia1 = "Lunes";

var dia2 = "Martes";

...

var dia7 = "Domingo";

Aunque el código anterior no es incorrecto, sí que es poco eficiente y complica en exceso la programación. Si en vez de los días de la semana se tuviera que guardar el nombre de los meses del año, el nombre de todos los países del mundo o las mediciones diarias de temperatura de los últimos 100 años, se tendrían que crear decenas o cientos de variables.

En este tipo de casos, se pueden agrupar todas las variables relacionadas en una colección de variables o array. El ejemplo anterior se puede rehacer de la siguiente forma:

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

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

*var nombre\_array = [valor1, valor2, ..., valorN];*

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

*var diaSeleccionado = dias[0]; // diaSeleccionado = "Lunes"*

*var otroDia = dias[5]; // otroDia = "Sábado"*

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

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

**Booleanos**

Las variables de tipo boolean o booleano también se conocen con el nombre de variables de tipo lógico. Aunque para entender realmente su utilidad se debe estudiar la programación avanzada con JavaScript del siguiente capítulo, su funcionamiento básico es muy sencillo.

Una variable de tipo boolean almacena un tipo especial de valor que solamente puede tomar dos valores: true (verdadero) o false (falso). No se puede utilizar para almacenar números y tampoco permite guardar cadenas de texto.

**Operadores**

Las variables por sí solas son de poca utilidad. Hasta ahora, sólo se ha visto cómo crear variables de diferentes tipos y cómo mostrar su valor mediante la función alert(). Para hacer programas realmente útiles, son necesarias otro tipo de herramientas.

Los operadores permiten manipular el valor de las variables, realizar operaciones matemáticas con sus valores y comparar diferentes variables. De esta forma, los operadores permiten a los programas realizar cálculos complejos y tomar decisiones lógicas en función de comparaciones y otros tipos de condiciones.

**Asignación**

El operador de asignación es el más utilizado y el más sencillo. Este operador se utiliza para guardar un valor específico en una variable. El símbolo utilizado es = (no confundir con el operador == que se verá más adelante):

*var valor1 = 3;*

A la izquierda del operador, siempre debe indicarse el nombre de una variable. A la derecha del operador, se pueden indicar variables, valores, condiciones lógicas, etc:

*var valor1 = 3;*

*var valor2 = 4;*

/\* Error, la asignación siempre se realiza a una variable,

por lo que en la izquierda no se puede indicar un número \*/

5 = numero1;

// Ahora, la variable numero1 vale 5

numero1 = 5;

// Ahora, la variable numero1 vale 4

numero1 = numero2;

**Incremento y decremento**

Estos dos operadores solamente son válidos para las variables numéricas y se utilizan para incrementar o decrementar en una unidad el valor de una variable.

*Ejemplo:*

*var numero = 5;*

*++numero;*

*alert(numero); // numero = 6*

El operador de incremento se indica mediante el prefijo ++ en el nombre de la variable. El resultado es que el valor de esa variable se incrementa en una unidad. Por tanto, el anterior ejemplo es equivalente a:

*var numero = 5;*

*numero = numero + 1;*

*alert(numero); // numero = 6*

De forma equivalente, el operador decremento (indicado como un prefijo -- en el nombre de la variable) se utiliza para decrementar el valor de la variable:

*var numero = 5;*

*--numero;*

*alert(numero); // numero = 4*

El anterior ejemplo es equivalente a:

*var numero = 5;*

*numero = numero - 1;*

*alert(numero); // numero = 4*

Los operadores de incremento y decremento no solamente se pueden indicar como prefijo del nombre de la variable, sino que también es posible utilizarlos como sufijo. En este caso, su comportamiento es similar pero muy diferente. En el siguiente ejemplo:

*var numero = 5;*

*numero++;*

*alert(numero); // numero = 6*

El resultado de ejecutar el script anterior es el mismo que cuando se utiliza el operador ++numero, por lo que puede parecer que es equivalente indicar el operador ++ delante o detrás del identificador de la variable. Sin embargo, el siguiente ejemplo muestra sus diferencias:

*var numero1 = 5;*

*var numero2 = 2;*

*numero3 = numero1++ + numero2;*

*// numero3 = 7, numero1 = 6*

*var numero1 = 5;*

*var numero2 = 2;*

*numero3 = ++numero1 + numero2;*

*// numero3 = 8, numero1 = 6*

Si el operador ++ se indica como prefijo del identificador de la variable, su valor se incrementa antes de realizar cualquier otra operación. Si el operador ++ se indica como sufijo del identificador de la variable, su valor se incrementa después de ejecutar la sentencia en la que aparece.

Por tanto, en la instrucción numero3 = numero1++ + numero2;, el valor de numero1 se incrementa después de realizar la operación (primero se suma y numero3 vale 7, después se incrementa el valor de numero1 y vale 6). Sin embargo, en la instrucción numero3 = ++numero1 + numero2;, en primer lugar se incrementa el valor de numero1 y después se realiza la suma (primero se incrementa numero1 y vale 6, después se realiza la suma y numero3 vale 8).

**Lógicos**

Los operadores lógicos son imprescindibles para realizar aplicaciones complejas, ya que se utilizan para tomar decisiones sobre las instrucciones que debería ejecutar el programa en función de ciertas condiciones.

El resultado de cualquier operación que utilice operadores lógicos siempre es un valor lógico o booleano.

***Negación***

Uno de los operadores lógicos más utilizados es el de la negación. Se utiliza para obtener el valor contrario al valor de la variable:

*var visible = true;*

*alert(!visible); // Muestra "false" y no "true"*

La negación lógica se obtiene prefijando el símbolo ! al identificador de la variable. El funcionamiento de este operador se resume en la siguiente tabla:

*variable !variable*

*true false*

*false true*

Si la variable original es de tipo booleano, es muy sencillo obtener su negación. Sin embargo, ¿qué sucede cuando la variable es un número o una cadena de texto? Para obtener la negación en este tipo de variables, se realiza en primer lugar su conversión a un valor booleano:

Si la variable contiene un número, se transforma en false si vale 0 y en true para cualquier otro número (positivo o negativo, decimal o entero).

Si la variable contiene una cadena de texto, se transforma en false si la cadena es vacía ("") y en true en cualquier otro caso.

*var cantidad = 0;*

*vacio = !cantidad; // vacio = true*

*cantidad = 2;*

*vacio = !cantidad; // vacio = false*

*var mensaje = "";*

*mensajeVacio = !mensaje; // mensajeVacio = true*

*mensaje = "Bienvenido";*

*mensajeVacio = !mensaje; // mensajeVacio = false*

***AND***

La operación lógica AND obtiene su resultado combinando dos valores booleanos. El operador se indica mediante el símbolo && y su resultado solamente es true si los dos operandos son true:

*variable1 variable2 variable1 && variable2*

*true true true*

*true false false*

*false true false*

*false false false*

*var valor1 = true;*

*var valor2 = false;*

*resultado = valor1 && valor2; // resultado = false*

*valor1 = true;*

*valor2 = true;*

*resultado = valor1 && valor2; // resultado = true*

***OR***

La operación lógica OR también combina dos valores booleanos. El operador se indica mediante el símbolo || y su resultado es true si alguno de los dos operandos es true:

variable1 variable2 variable1 || variable2

true true true

true false true

false true true

false false false

var valor1 = true;

var valor2 = false;

resultado = valor1 || valor2; // resultado = true

valor1 = false;

valor2 = false;

resultado = valor1 || valor2; // resultado = false

**Matemáticos**

JavaScript permite realizar manipulaciones matemáticas sobre el valor de las variables numéricas. Los operadores definidos son: suma (+), resta (-), multiplicación (\*) y división (/). Ejemplo:

var numero1 = 10;

var numero2 = 5;

resultado = numero1 / numero2; // resultado = 2

resultado = 3 + numero1; // resultado = 13

resultado = numero2 – 4; // resultado = 1

resultado = numero1 \* numero 2; // resultado = 50

Además de los cuatro operadores básicos, JavaScript define otro operador matemático que no es sencillo de entender cuando se estudia por primera vez, pero que es muy útil en algunas ocasiones.

Se trata del operador "módulo", que calcula el resto de la división entera de dos números. Si se divide por ejemplo 10 y 5, la división es exacta y da un resultado de 2. El resto de esa división es 0, por lo que módulo de 10 y 5 es igual a 0.

Sin embargo, si se divide 9 y 5, la división no es exacta, el resultado es 1 y el resto 4, por lo que módulo de 9 y 5 es igual a 4.

El operador módulo en JavaScript se indica mediante el símbolo %, que no debe confundirse con el cálculo del porcentaje:

*var numero1 = 10;*

*var numero2 = 5;*

*resultado = numero1 % numero2; // resultado = 0*

*numero1 = 9;*

*numero2 = 5;*

*resultado = numero1 % numero2; // resultado = 4*

Los operadores matemáticos también se pueden combinar con el operador de asignación para abreviar su notación:

*var numero1 = 5;*

*numero1 += 3; // numero1 = numero1 + 3 = 8*

*numero1 -= 1; // numero1 = numero1 - 1 = 4*

*numero1 \*= 2; // numero1 = numero1 \* 2 = 10*

*numero1 /= 5; // numero1 = numero1 / 5 = 1*

*numero1 %= 4; // numero1 = numero1 % 4 = 1*

**Relacionales**

Los operadores relacionales definidos por JavaScript son idénticos a los que definen las matemáticas: mayor que (>), menor que (<), mayor o igual (>=), menor o igual (<=), igual que (==) y distinto de (!=).

Los operadores que relacionan variables son imprescindibles para realizar cualquier aplicación compleja, como se verá en el siguiente capítulo de programación avanzada. El resultado de todos estos operadores siempre es un valor booleano:

*var numero1 = 3;*

*var numero2 = 5;*

*resultado = numero1 > numero2; // resultado = false*

*resultado = numero1 < numero2; // resultado = true*

*numero1 = 5;*

*numero2 = 5;*

*resultado = numero1 >= numero2; // resultado = true*

*resultado = numero1 <= numero2; // resultado = true*

*resultado = numero1 == numero2; // resultado = true*

*resultado = numero1 != numero2; // resultado = false*

Se debe tener especial cuidado con el operador de igualdad (==), ya que es el origen de la mayoría de errores de programación, incluso para los usuarios que ya tienen cierta experiencia desarrollando scripts. El operador == se utiliza para comparar el valor de dos variables, por lo que es muy diferente del operador =, que se utiliza para asignar un valor a una variable:

// El operador "=" asigna valores

*var numero1 = 5;*

*resultado = numero1 = 3; // numero1 = 3 y resultado = 3*

// El operador "==" compara variables

*var numero1 = 5;*

*resultado = numero1 == 3; // numero1 = 5 y resultado = false*

Los operadores relacionales también se pueden utilizar con variables de tipo cadena de texto:

*var texto1 = "hola";*

*var texto2 = "hola";*

*var texto3 = "adios";*

*resultado = texto1 == texto3; // resultado = false*

*resultado = texto1 != texto2; // resultado = false*

*resultado = texto3 >= texto2; // resultado = false*

Cuando se utilizan cadenas de texto, los operadores "mayor que" (>) y "menor que" (<) siguen un razonamiento no intuitivo: se compara letra a letra comenzando desde la izquierda hasta que se encuentre una diferencia entre las dos cadenas de texto. Para determinar si una letra es mayor o menor que otra, las mayúsculas se consideran menores que las minúsculas y las primeras letras del alfabeto son menores que las últimas (a es menor que b, b es menor que c, A es menor que a, etc.)

**Estructuras**

**Estructura if**

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

*if(condicion) {*

*...*

*}*

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

Ejemplo:

*var mostrarMensaje = true;*

*if(mostrarMensaje) {*

*alert("Hola Mundo");*

*}*

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

El ejemplo se podría reescribir también como:

*var mostrarMensaje = true;*

*if(mostrarMensaje == true) {*

*alert("Hola Mundo");*

*}*

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

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

*var mostrarMensaje = true;*

// Se comparan los dos valores

*if(mostrarMensaje == false) {*

*...*

*}*

// Error - Se asigna el valor "false" a la variable

*if(mostrarMensaje = false) {*

*...*

*}*

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

*var mostrado = false;*

*if(!mostrado) {*

*alert("Es la primera vez que se muestra el mensaje");*

*}*

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

*var mostrado = false;*

*var usuarioPermiteMensajes = true;*

*if(!mostrado && usuarioPermiteMensajes) {*

*alert("Es la primera vez que se muestra el mensaje");*

*}*

La condición anterior está formada por una operación AND sobre dos variables. A su vez, a la primera variable se le aplica el operador de negación antes de realizar la operación AND. De esta forma, como el valor de mostrado es false, el valor !mostrado sería true. Como la variable usuarioPermiteMensajes vale true, el resultado de !mostrado && usuarioPermiteMensajes sería igual a true && true, por lo que el resultado final de la condición del if() sería true y por tanto, se ejecutan las instrucciones que se encuentran dentro del bloque del if().

**Estructura if...else**

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

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

*if(condicion) {*

*...*

*}*

*else {*

*...*

*}*

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

*var edad = 18;*

*if(edad >= 18) {*

*alert("Eres mayor de edad");*

*}*

*else {*

*alert("Todavía eres menor de edad");*

*}*

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

El siguiente ejemplo compara variables de tipo cadena de texto:

*var nombre = "";*

*if(nombre == "") {*

*alert("Aún no nos has dicho tu nombre");*

*}*

*else {*

*alert("Hemos guardado tu nombre");*

*}*

La condición del if() anterior se construye mediante el operador ==, que es el que se emplea para comparar dos valores (no confundir con el operador = que se utiliza para asignar valores). En el ejemplo anterior, si la cadena de texto almacenada en la variable nombre es vacía (es decir, es igual a "") se muestra el mensaje definido en el if(). En otro caso, se muestra el mensaje definido en el bloque else { }.

La estructura if...else se puede encadenar para realizar varias comprobaciones seguidas:

*if(edad < 12) {*

*alert("Todavía eres muy pequeño");*

*}*

*else if(edad < 19) {*

*alert("Eres un adolescente");*

*}*

*else if(edad < 35) {*

*alert("Aun sigues siendo joven");*

*}*

*else {*

*alert("Piensa en cuidarte un poco más");*

*}*

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

**Estructura for**

Las estructuras if y if...else no son muy eficientes cuando se desea ejecutar de forma repetitiva una instrucción. Por ejemplo, si se quiere mostrar un mensaje cinco veces, se podría pensar en utilizar el siguiente if:

var veces = 0;

if(veces < 4) {

alert("Mensaje");

veces++;

}

Se comprueba si la variable veces es menor que 4. Si se cumple, se entra dentro del if(), se muestra el mensaje y se incrementa el valor de la variable veces. Así se debería seguir ejecutando hasta mostrar el mensaje las cinco veces deseadas.

Sin embargo, el funcionamiento real del script anterior es muy diferente al deseado, ya que solamente se muestra una vez el mensaje por pantalla. La razón es que la ejecución de la estructura if() no se repite y la comprobación de la condición sólo se realiza una vez, independientemente de que dentro del if() se modifique el valor de la variable utilizada en la condición.

La estructura for permite realizar este tipo de repeticiones (también llamadas bucles) de una forma muy sencilla. No obstante, su definición formal no es tan sencilla como la de if():

*for(inicializacion; condicion; actualizacion) {*

*...*

*}*

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

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

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

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

*var mensaje = "Hola, estoy dentro de un bucle";*

*for(var i = 0; i < 5; i++) {*

*alert(mensaje);*

*}*

La parte de la inicialización del bucle consiste en:

*var i = 0;*

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

La zona de condición del bucle es:

*i < 5*

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

Como la variable i se ha inicializado a un valor de 0 y la condición para salir del bucle es que i sea menor que 5, si no se modifica el valor de i de alguna forma, el bucle se repetiría indefinidamente.

Por ese motivo, es imprescindible indicar la zona de actualización, en la que se modifica el valor de las variables que controlan el bucle:

*i++*

En este caso, el valor de la variable i se incrementa en una unidad después de cada repetición. La zona de actualización se ejecuta después de la ejecución de las instrucciones que incluye el for.

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

Normalmente, la variable que controla los bucles for se llama i, ya que recuerda a la palabra índice y su nombre tan corto ahorra mucho tiempo y espacio.

El ejemplo anterior que mostraba los días de la semana contenidos en un array se puede rehacer de forma más sencilla utilizando la estructura for:

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

*for(var i=0; i<7; i++) {*

*alert(dias[i]);*

*}*

3.4.4. Estructura for...in

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

for(indice in array) {

...

}

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

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

*for(i in dias) {*

*alert(dias[i]);*

*}*

La variable que se indica como indice es la que se puede utilizar dentro del bucle for...in para acceder a los elementos del array. De esta forma, en la primera repetición del bucle la variable i vale 0 y en la última vale 6.

Esta estructura de control es la más adecuada para recorrer arrays (y objetos), ya que evita tener que indicar la inicialización y las condiciones del bucle for simple y funciona correctamente cualquiera que sea la longitud del array. De hecho, sigue funcionando igual aunque varíe el número de elementos del array.