

Introducción al lenguaje JavaScript

- ✓ Conocer las principales características del lenguaje JavaScript.
- ✓ Dominar la sintaxis básica del lenguaje.
- ✓ Comprender y utilizar los distintos tipos de variables y operadores presentes en el lenguaje JavaScript.
- ✓ Conocer las diferentes sentencias condicionales de JavaScript y saber realizar operaciones complejas con ellas.

En el capítulo anterior hemos visto algunos de los lenguajes y tecnologías de programación en el entorno del cliente. Algunos de estos lenguajes permiten mejorar la interactividad con el usuario y uno de los lenguajes más utilizados para este propósito es JavaScript. En este capítulo veremos las principales características de este lenguaje, comenzando con el ejemplo “Hola Mundo”, el cual se suele usar como un primer ejercicio típico que sirve como introducción al estudio de cualquier lenguaje de programación. A continuación, estudiaremos la sintaxis del lenguaje, además de los conceptos relacionados con los tipos de datos, las variables y los operadores utilizados por JavaScript. Por último, veremos las sentencias condicionales, las cuales nos permitirán aumentar la complejidad de los ejemplos y ejercicios propuestos durante todo el capítulo.

CARACTERÍSTICAS DE JAVASCRIPT

JavaScript es un lenguaje de programación interpretado que se utiliza fundamentalmente para dotar de comportamiento dinámico a las páginas web. Por ello, cualquier navegador web actual incorpora un intérprete para código JavaScript.

El primer lenguaje de *scripting* para la Web fue LiveScript, desarrollado por Netscape para proporcionar una alternativa “ligera” a Java a la hora de soportar comportamiento dinámico tanto en el lado del servidor como en el del cliente. No obstante, dado el auge y popularidad de Java, en 1995 Netscape y Sun aprovecharon el lanzamiento de la versión 2 de Navigator, el navegador web de Netscape para anunciar que el lenguaje pasaba a denominarse JavaScript. Por otro lado, el éxito de JavaScript provocó que, incluso cuando ya existía VBScript, otro lenguaje de *scripting* para la Web basado en BASIC, Microsoft lanzase su propia versión de JavaScript, denominada JScript e integrada en sus navegadores a partir de Internet Explorer 3.

A pesar de que las diferencias entre JavaScript y JScript no fueran muy grandes, el hecho de que cada navegador soportase su propio lenguaje obligaba a veces a los desarrolladores a programar dos veces la misma funcionalidad (una con cada lenguaje) e introducir en sus páginas web sentencias condicionales para distinguir en qué navegador se estaba ejecutando la página e invocar la ejecución del código escrito con el lenguaje soportado por el navegador en cuestión.

En respuesta a este tipo de problemas, la ECMA (*European Computer Manufacturers Association*) emprendió un esfuerzo de estandarización que desembocó en la publicación del estándar ECMAScript¹. Aunque a día de hoy, tanto JavaScript como JScript son conformes a dicho estándar, es el término JavaScript el que se impuso y se utiliza como denominador común para referirse al propio lenguaje y al estándar.

A continuación enumeramos algunas características de JavaScript:

La sintaxis de JavaScript se asemeja a la de C++ y la de Java. De hecho, JavaScript está basado en el concepto de objeto, pero no es un lenguaje orientado a objetos.

Cuando utilizamos JavaScript en el contexto de la programación web en el lado del cliente tendemos a confundir los objetos JavaScript con los objetos del documento HTML que manejamos con JavaScript, es decir, los objetos del *Document Object Model* (DOM). Esta distinción se hace evidente cuando utilizamos JavaScript en otros contextos, como en la programación en el lado del servidor o en las aplicaciones Flash.

Los objetos JavaScript utilizan herencia basada en prototipos, muy diferente de la herencia basada en clases propia de los lenguajes orientados a objetos, como Java. En JavaScript los objetos heredan propiedades directamente de otros objetos sin necesidad de que sean instancias de una misma clase (o de clases que participen en una misma jerarquía).

Probablemente la característica más importante de JavaScript sea que es un lenguaje débilmente tipado, lo que permite que una variable pueda contener valores de distintos tipos en diferentes momentos de la ejecución del programa. A cambio, muchos errores de programación no aparecen hasta que el programa es ejecutado, ya que el compilador es incapaz de detectarlos.

Una característica *poco deseable* de JavaScript es el hecho de que, por defecto, todas las variables son globales. Es decir, aquellas variables que no son definidas dentro de otra variable se ubican en un espacio de nombres común denominado *global object*. En general, el uso de variables globales no es una buena práctica de programación.

“HOLA MUNDO” CON JAVASCRIPT

La forma más inmediata de empezar a experimentar con JavaScript es escribir secuencias de comandos simples. En este sentido, una ventaja de JavaScript es que un explorador web y un simple editor de textos constituyen un completo entorno de desarrollo; no necesitamos comprar ni descargar ningún software especial para empezar a experimentar con JavaScript.

Para ejecutar cualquier programa JavaScript tenemos que embeber el código JavaScript en una página HTML, de la misma forma que insertamos cualquier otro contenido HTML: utilizando etiquetas para marcar el principio y el fin del bloque de código JavaScript. En este caso utilizamos la etiqueta `<script></script>`. De esta forma el navegador sabe que el bloque de texto contenido entre la etiqueta de inicio y fin no se corresponde con código HTML, si no que se trata de código que debe ser procesado antes de mostrar el resultado en pantalla. Así, el navegador, que incluye un intérprete de JavaScript, se encargará automáticamente de ejecutar el código cuando cargue la página y reflejar el efecto de la ejecución sobre la página que tiene que mostrar.

Hay dos formas de embeber el código JavaScript utilizando la etiqueta <script>:

Incluirlo directamente en la página HTML mediante la etiqueta <script>. Crea un fichero HolaMundo.html y copia y pega el siguiente código en el fichero:

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="content-type" content="text/html;
      charset=utf-8">
    <title>Hola Mundo</title>
  </head>
  <body>
    <script>
      alert('Hola mundo en JavaScript')
    </script>
  </body>
</html>
```

Utilizar el atributo src de dicha etiqueta para especificar el fichero que contiene el código JavaScript.

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="content-type" content="text/html;
      charset=utf-8">
    <title>Hola Mundo</title>
    <script type="text/javascript" src="2.3HolaMundo.js">
    </script>
  </head>
  <body></body>
</html>
```

El fichero 2.3-HolaMundo.js debe contener esta línea de código:

```
alert('Hola mundo en JavaScript')
```

Si abriésemos los dos ficheros HolaMundo.html con un navegador web, suponiendo que antes hayamos creado el fichero 2.3-HolaMundo.js en el mismo directorio, el resultado sería exactamente el mismo: el navegador muestra el mensaje "Hola mundo en JavaScript".



Figura 2.1. "Hola mundo" en JavaScript

No obstante, la forma recomendada de proceder es la segunda, ya que al localizar el código JavaScript en un fichero externo estamos facilitando la reutilización y modularización de dicho código.

Dado que hay distintos lenguajes de *script*, el atributo *type* nos permite decirle al navegador cuál es el utilizado para codificar el *script* que encontrará a continuación. De esta forma, el navegador sabrá cuál es el intérprete que tiene que utilizar para ejecutar el *script*. Podemos omitirlo, ya que los navegadores más conocidos utilizan JavaScript como lenguaje de *script* por defecto, pero es una buena práctica especificarlo y, además, es obligatorio de acuerdo a la norma de la W3C, la cual es la encargada de la especificación del estándar.

ACTIVIDADES 2.1

Cree un nuevo fichero HTML vacío (puedes tomar el fichero 2.2-HolaMundo.html como ejemplo y eliminar la etiqueta <script>). Añada el siguiente guión JavaScript en el cuerpo de la página (entre las etiquetas <body> y </body>):

```
<script type="text/javascript">
    document.bgColor = "red";
    alert('Cambiaremos el color de la p\xElgina');
</script>
```

Guarde la página con otro nombre (Actividad-2.1-CambiarColor.html), ábrala con su navegador y compruebe el resultado.

Podemos observar que en la primera actividad hemos utilizado una cadena de escape para mostrar correctamente los caracteres tildados. Para evitar este problema debemos emplear la cadena \x^{dd}, donde dd corresponde al carácter especial especificado por dos dígitos hexadecimales según el estándar Unicode. Existen diferentes secuencias de escape que son útiles a la hora de introducir no solo caracteres tildados, sino también saltos de línea (\n), tabuladores (\t), etc.

EL LENGUAJE JAVASCRIPT: SINTAXIS

El lenguaje JavaScript tiene una sintaxis muy similar a la de Java o a la del lenguaje C++. La sintaxis especifica aspectos, como los caracteres que se deben utilizar para definir los comentarios, la forma de los nombres de las variables o el modo de separar las diferentes instrucciones del código.

MAYÚSCULAS Y MINÚSCULAS

Una de las primeras dificultades que encuentran los programadores novatos en JavaScript es que este lenguaje distingue entre mayúsculas y minúsculas, a diferencia de HTML, que no realiza ninguna diferencia entre ellas. Esta regla cobra importancia cuando utilizamos variables, objetos, funciones o cualquier otro símbolo del lenguaje.

Por ejemplo, no es lo mismo utilizar la función `alert()` que `Alert()`. Tal y como hemos visto en el ejemplo de `HolaMundo.html`, la primera muestra un texto en una ventana emergente del navegador, mientras que la segunda no existe a menos que la defina el programador.

COMENTARIOS EN EL CÓDIGO

Al igual que la mayor parte de los lenguajes de programación, JavaScript permite insertar comentarios dentro del código. Estás líneas de código no son interpretadas por el navegador. Su función principal es la de facilitar la lectura del código al programador.

Existen dos formas de insertar comentarios. Una de ellas es a través de la doble barra (`//`), con la cual comentamos una sola línea de código. La otra forma que podemos utilizar para comentar un código es a través de los signos `/*` al inicio del comentario y los signos `*/` al final del mismo. De este modo podemos comentar varias líneas de código. En el siguiente código vemos un ejemplo de las dos formas que existen para insertar comentarios:

```
<script type="text/javascript">
  // Este modo permite comentar una sola linea
  /* Este modo permite realizar
  comentarios de
  varias lineas */
</script>
```

TABULACIÓN Y SALTOS DE LÍNEA

JavaScript ignora los espacios, las tabulaciones y los saltos de línea presentes entre los símbolos del código. La única restricción en cuanto a los saltos de línea la vemos en el siguiente apartado.

En el momento en que los programas empiezan a ser complejos, podemos apreciar la utilidad de emplear las tabulaciones y los saltos de línea adecuados para mejorar la presentación y la legibilidad del código. A continuación,

podemos observar la diferencia entre un código bien estructurado que facilita la lectura y un código que no está bien estructurado.

• **Versión 1:**

```
<script type="text/javascript">var i,j=0;
for (i=0;i<5;i++){ alert("Variable i: "+i);
for (j=0;j<5;j++){
if (i%2==0){
document.write
(i + "-" + j + "<br>"); }}}</script>
```

Versión 2:

```
<script type="text/javascript">
var i,j=0;
for (i=0;i<5;i++){
    alert("Variable i: "+i;
    for (j=0;j<5;j++){
        if (i%2==0{
            document.write(i + "-" + j + "<br>");
        }
    }
}
</script>
```

Podemos notar que en la segunda versión del ejemplo es mucho más fácil comprender a cuál sentencia corresponde cada instrucción. En el ejemplo hemos utilizado sentencias condicionales y otros conceptos que estudiaremos a lo largo del capítulo.

EL PUNTO Y COMA

Normalmente, se suele insertar un signo de punto y coma (;) al final de cada instrucción de JavaScript, al igual que se hace en lenguajes de programación como C, C++ o Java. Este signo tiene la utilidad de separar y diferenciar cada instrucción. No obstante, podemos omitir dicho signo si cada instrucción de JavaScript se encuentra en una línea independiente.

Por ejemplo, las siguientes instrucciones podrían escribirse de este modo:

```
pi_egipcio = 3.16
pi_griego = 3.14
```

Sin embargo, si queremos definir los dos valores en una misma línea, es necesario utilizar al menos el primer punto y coma:

```
pi_egipcio = 3.16; pi_griego = 3.14;
```

La omisión del punto y coma no es una buena práctica de programación y un simple olvido nos puede hacer perder un tiempo muy valioso, con lo cual es aconsejable habituarnos a su uso.

PALABRAS RESERVADAS

JavaScript contiene una serie de palabras que no podemos utilizar para definir nombres de variables, funciones o etiquetas. Según la versión 5.1 de ECMAScript, las palabras reservadas son las que vemos a continuación:

Tabla 2.1 Palabras reservadas

break	delete	if	this	while
case	do	in	throw	with
catch	else	instanceof	try	
continue	finally	new	typeof	
debugger	for	return	var	
default	function	switch	void	

Además de las anteriores palabras reservadas del lenguaje, el estándar ECMAScript contempla el uso de otras palabras reservadas en futuras versiones, como por ejemplo `class`, `const`, `enum` y `export`, entre otras. Aunque los interpretadores actuales de JavaScript permiten el uso de estas posibles futuras palabras clave, es aconsejable revisar la versión del estándar a la hora de la lectura de este libro e indagar cuáles son en el momento actual, con el fin de evitar su uso en la realización de programas.

ACTIVIDADES 2.2

Visite la página web <http://www.ecmascript.org/> y consulte todas las palabras reservadas para las futuras versiones del estándar.

TIPOS DE DATOS

En todo programa informático manipulamos constantemente diferentes tipos de valores como por ejemplo números o textos. Los tipos de datos especifican qué tipo de valor se guardará en una determinada variable. Este concepto es importante a la hora de determinar cómo podemos combinar ciertas variables o si es posible hacerlo.

Los tres tipos de datos primitivos de JavaScript son: números, cadenas de texto y valores booleanos. Además de estos tres tipos de datos, existe el tipo de datos compuesto llamado `objeto`. Este último representa una colección de valores primitivos o de valores compuestos por otros objetos. Esta colección de valores nos permite definir tipos de datos como por ejemplo las matrices y las funciones, las cuales explicaremos en los siguientes capítulos del libro.

NÚMEROS

En JavaScript, a diferencia de la mayoría de lenguajes de programación, existe solo un tipo de datos numérico. Todos los números que utilicemos, ya sean valores enteros o valores de punto flotante, se representarán a través del formato de punto flotante de 64 bits definido por el estándar IEEE 754². Este formato es el llamado `double` en los lenguajes Java o C++.

Además de los valores enteros en base 10, en JavaScript es posible utilizar valores hexadecimales, es decir, en base 16. Para ello, es necesario iniciar el valor con `0x` seguido por una secuencia de dígitos hexadecimales.

La mayor parte de las implementaciones de JavaScript admite también la representación de valores en base 8, aunque el estándar ECMAScript no lo admite. Debido a esto es aconsejable evitar el uso de los valores en formato octal.

CADENAS DE TEXTO

El tipo de datos que utiliza JavaScript para representar cadenas de texto, es llamado `string`. Con este tipo de datos es posible representar una secuencia de letras, dígitos, signos de puntuación o cualquier otro carácter de Unicode. La cadena de caracteres la debemos definir entre comillas dobles o comillas simples (" o ').

Dado que las comillas dobles o simples las utilizamos para delimitar las cadenas de texto, es posible que surja la duda de qué hacer para representar estos signos. Para ello, necesitamos utilizar secuencias de escape haciendo uso de la barra invertida (\). La combinación de esta barra con otros caracteres permite que el navegador pueda representar un carácter, que sin la ayuda de la barra invertida no podría representarse. En la Tabla 2.2 podemos ver las principales combinaciones de secuencias de escape.

² <http://grouper.ieee.org/groups/754/>

Tabla 2.2 Secuencias de escape

Secuencia de escape	Resultado
\\"	Barra invertida
'	Comilla simple
\"	Comillas dobles
\n	Salto de línea
\t	Tabulación horizontal
\v	Tabulación vertical
\f	Salto de página
\r	Retorno de carro
\b	Retroceso

VALORES BOOLEANOS

El tipo de datos booleano, también conocido como valores lógicos, es el tipo de datos más simple debido a que admite solo dos valores: `true` o `false` (verdadero o falso). Debemos definir estos valores sin ningún tipo de comillas. Este tipo de datos es bastante útil a la hora de evaluar expresiones lógicas o verificar condiciones particulares en nuestros programas.

ACTIVIDADES 2.3

Cree un fichero HTML que permita mostrar el siguiente texto en una ventana emergente, a través de la función `alert()`:

```
I'm = I am  
I don't = I do not
```

En la Figura 2.2 puede ver el resultado esperado. Tenga en cuenta que debe usar secuencias de escape para poder representar las comillas simples y el salto de línea.

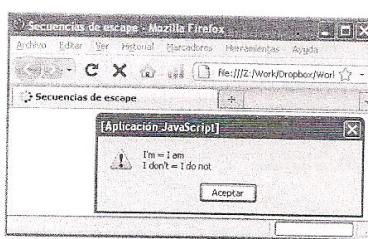


Figura 2.2. Ejemplo del uso de las secuencias de escape

VARIABLES

Las variables se pueden definir como zonas de la memoria de un ordenador que se identifican con un nombre y en las cuales se almacenan ciertos datos. Las variables nos permiten manipular y guardar datos que, en algunos casos, no sabremos su valor a priori. El desarrollo de un *script* conlleva generalmente dos aspectos relacionados con las variables:

- Declaración de variables.
- Inicialización de variables.

A continuación entraremos en los detalles de cada uno de estos dos aspectos.

DECLARACIÓN DE VARIABLES

El primer paso para utilizar una variable es definirla. En JavaScript las variables se definen a través de la palabra clave `var` seguida por el nombre que queramos darle a la variable, tal y como vemos en el siguiente ejemplo:

```
var mi_variable_1;  
var mi_variable_2;
```

Es posible declarar más de una variable en una sola línea de código a través de la separación por comas:

```
var mi_variable_1, mi_variable_2;
```

En los anteriores ejemplos hemos declarado las variables, pero no les hemos asignado ningún valor. El contenido de las variables estará indefinido hasta que una parte del código almacene un valor en ellas. Este proceso lo describimos en el siguiente apartado.

INICIALIZACIÓN DE VARIABLES

Podemos realizar la asignación de un valor a una variable de tres formas:

- Asignación directa de un valor concreto.
- Asignación indirecta a través de un cálculo en el que se implican otras variables o constantes.
- Asignación a través de la solicitud del valor al usuario del programa.

A continuación, podemos ver los ejemplos respectivos de cada caso:

```
var mi_variable_1 = 30;  
var mi_variable_2 = mi_variable_1 + 10;  
var mi_variable_3 = prompt('Introduce un valor:');
```

En todas ellas debemos utilizar el operador de asignación (el signo igual, “=”). Si en algún fragmento del código intentamos utilizar una variable que no haya sido inicializada, JavaScript generará un error.



Podemos inicializar una variable sin utilizar la palabra clave `var`. De este modo JavaScript declara implícitamente dicha variable, pero tenemos que tener en cuenta que esta variable será una variable global. Esto quiere decir que su valor tendrá un ámbito global, incluso dentro de funciones locales. De este modo corremos el riesgo de modificar un valor en el cual se basa algún otro fragmento del programa. Afortunadamente, este problema lo podemos evitar simplemente si nos acordamos siempre de utilizar la palabra clave `var`.

Una vez declarada una variable y haberle asignado un valor, podemos utilizarla en otras instrucciones del programa. En el momento en que usamos una variable, el navegador accede a la memoria del ordenador, recupera su valor y lo sustituye en la instrucción.

ACTIVIDADES 2.4

Cree un nuevo fichero HTML y, al igual que en las anteriores actividades, copie el siguiente código JavaScript dentro del cuerpo de la página:

```
<script type="text/javascript">
    var primer_saludo = "hola";
    var segundo_saludo = primer_saludo;
    primer_saludo = "hello";
    alert(segundo_saludo);
</script>
```

Antes de ver el resultado, ¿cuál cree que será el valor en la ventana emergente?

OPERADORES

Hasta el momento hemos utilizado ejemplos con sentencias bastante sencillas pero, a partir de ahora, podremos construir diferentes expresiones en las que tendremos una colección de símbolos, palabras o números con los que podremos realizar cálculos más complejos. Es posible construir este tipo de expresiones gracias al uso de los operadores de JavaScript.

JavaScript utiliza principalmente cinco tipos de operadores:

- Operadores aritméticos.
- Operadores lógicos.
- Operadores de asignación.
- Operadores de comparación.
- Operadores condicionales.

Los elementos utilizados en una expresión y unidos a través de un operador se definen como operandos. A continuación presentamos los diferentes tipos de operadores.

OPERADORES ARITMÉTICOS

Los operadores aritméticos permiten realizar cálculos elementales entre variables numéricas. En la Tabla 2.3 podemos ver las cuatro operaciones aritméticas elementales: suma, resta, multiplicación y división, además de otros tres operadores menos comunes: módulo, incremento y decremento.

Tabla 2.3 Operadores aritméticos

Operador	Nombre	Descripción
+	Suma	Efectúa la suma entre los operandos.
-	Resta	Efectúa la resta entre los operandos.
*	Multiplicación	Efectúa la multiplicación entre los operandos.
/	División	Efectúa la división entre los operandos.
%	Módulo	Extrae la parte entera del resultado de la división entre los operandos.
++	Incremento	Permite incrementar un valor.
--	Decremento	Permite decrementar un valor.

El operador módulo divide un número entre otro y lo que devuelve es el resto de dicha división. Por ejemplo, si quisieramos comprobar que el año 2012 es un año bisiesto (teniendo en cuenta algunas excepciones, los años bisiestos son divisibles por 4) debemos realizar la siguiente operación:

```
var anyo = 2012;
var bisiesto = anyo % 4;
```

Si la variable `bisiesto` almacena el valor 0, podemos afirmar que 2012 es un año bisiesto.

Los operadores incremento y decremento permiten realizar una de las operaciones más comunes en los lenguajes de programación, es decir, incrementar o decrementar un valor en una unidad. Es posible utilizar estos operadores antes o después de la variable, aunque su efecto es diferente en ambos casos. Por ejemplo, el siguiente código almacena el valor 2 en ambas variables

```
variable_1 = 1;
variable_2 = ++variable_1;
```

Mientras que en este otro ejemplo la primera variable contendrá al valor 2 y la segunda 1:

```
variable_1 = 1;
variable_2 = variable_1++;
```

OPERADORES LÓGICOS

Los operadores lógicos tienen la utilidad de combinar o manipular diferentes expresiones lógicas con el fin de evaluar si el resultado de esta combinación es verdadero o falso. Generalmente, se utiliza este tipo de operadores en el caso en que debamos tomar decisiones dentro de un programa. Podemos ver los tres operadores lógicos que existen en la Tabla 2.4.

Tabla 2.4 Operadores lógicos

Operador	Nombre	Descripción
&&	Y	Ejecuta la operación booleana AND sobre los valores. Devuelve true solo si todos los valores son true. Devuelve false en caso contrario.
	O	Ejecuta la operación booleana OR sobre los valores. Devuelve true en el caso en que al menos uno de los valores sea true. Devuelve false en caso contrario.
!	No	Invierte el valor booleano de su operando.

OPERADORES DE ASIGNACIÓN

Además del operador de asignación igual (=), JavaScript cuenta con otros operadores de asignación que tienen un carácter en común con los operadores aritméticos de incremento y decremento. Esta característica es la de ahorrarnos tiempo y disminuir la cantidad de código escrito en los programas. Gracias al uso de estos operadores, podemos obtener métodos abreviados que permiten evitar el tener que escribir dos veces la variable que se encuentra a la izquierda del operador. La Tabla 2.5 muestra todos los operadores de asignación.

Tabla 2.5 Operadores de asignación

Operador	Nombre	Descripción
<code>+=</code>	Suma y asigna	Ejecuta una suma y asigna el valor al operando de la izquierda.
<code>-=</code>	Resta y asigna	Ejecuta una resta y asigna el valor al operando de la izquierda.
<code>*=</code>	Multiplica y asigna	Ejecuta una multiplicación y asigna el valor al operando de la izquierda.
<code>/=</code>	Divide y asigna	Ejecuta una división y asigna el valor al operando de la izquierda.
<code>%=</code>	Módulo y asigna	Ejecuta el módulo y asigna el valor al operando de la izquierda.

Por ejemplo, podríamos usar el operador de restar y asignar de este modo:

```
var deudas = 1500; // tenemos unas deudas de 1500 euros  
  
deudas -= 300; // nuestras deudas disminuyen de 300 euros  
// esto es lo mismo que escribir deudas = deudas - 300
```

OPERADORES DE COMPARACIÓN

Los operadores de comparación, tal y como indica su nombre, nos permiten comparar todo tipo de variables con el fin de verificar sus valores. El resultado de dicha comparación nos devolverá un valor booleano y determina el orden relativo de dos valores. La comparación ejecutada por estos operadores se puede realizar solo sobre números y cadenas. Si utilizamos otro tipo de datos, estos se convertirán automáticamente para intentar que la comparación se pueda llevar a cabo.

Tabla 2.6 Operadores de comparación

Operador	Nombre	Descripción
<	Menor que	Verifica si el operando a la izquierda del operador es menor que el operando de la derecha. Devuelve <code>true</code> en ese caso o <code>false</code> en caso contrario.
<=	Menor o igual que	Verifica si el operando a la izquierda del operador es menor o igual que el operando de la derecha. Devuelve <code>true</code> en ese caso o <code>false</code> en caso contrario.
==	Igual	Verifica si los dos operandos son iguales. Devuelve <code>true</code> en ese caso o <code>false</code> en caso contrario.
>	Mayor que	Verifica si el operando a la izquierda del operador es mayor que el operando de la derecha. Devuelve <code>true</code> en ese caso o <code>false</code> en caso contrario.
>=	Mayor o igual que	Verifica si el operando a la izquierda del operador es mayor o igual que el operando de la derecha. Devuelve <code>true</code> en ese caso o <code>false</code> en caso contrario.
!=	Diferente	Verifica si los dos operandos son diferentes. Devuelve <code>true</code> en ese caso o <code>false</code> en caso contrario.
====	Estrictamente igual	Verifica si el operando a la izquierda del operador es igual y del mismo tipo de datos que el operando de la derecha. Devuelve <code>true</code> en ese caso o <code>false</code> en caso contrario.
!==	Estrictamente diferente	Verifica si el operando a la izquierda del operador es diferente y/o de tipo diferente que el operando de la derecha. Devuelve <code>true</code> en ese caso o <code>false</code> en caso contrario.

OPERADORES CONDICIONALES

Existe otro tipo de operador un poco más complejo, pero bastante útil a la hora de tomar decisiones en los programas de JavaScript. Se trata del operador condicional. Con este operador podemos indicarle al navegador que ejecute una acción en concreto después de evaluar una expresión. El operador condicional consta de tres partes: la primera es la expresión a evaluar, la segunda es la acción a realizar si la expresión es verdadera, y la tercera parte es la acción a realizar si la expresión es falsa.

Tabla 2.7 Operadores condicionales

Operador	Nombre	Descripción
: ?	Condicional	Si la expresión antes del operador es verdadera, se utiliza el primer valor a la derecha. En caso contrario se utiliza el segundo valor a la derecha.

Por ejemplo, si queremos avisar al usuario que no se puede efectuar una división por cero, es posible hacerlo mediante un operador condicional:

```
<script type="text/javascript">
    var dividendo = prompt("Introduce el dividendo: ");
    var divisor = prompt("Introduce el divisor: ");
    var resultado;
    divisor != 0 ? resultado = dividendo/divisor :
        alert("No es posible la división por cero");
    alert("El resultado es: " + resultado);
</script>
```

ACTIVIDADES 2.5

La precedencia de los operadores determina el orden en que se ejecuta una expresión. Realice una búsqueda con el fin de encontrar alguna de las tablas que indiquen el orden de precedencia de los operadores de JavaScript.

SENTENCIAS CONDICIONALES

Los ejemplos que hemos utilizado hasta el momento han sido ejemplos lineales en los que las sentencias las ejecutábamos unas detrás de otras, desde la primera hasta la última. Podemos controlar la toma de decisiones y el posterior resultado por parte del navegador a través del uso de sentencias condicionales. Dichas sentencias permiten evaluar condiciones y ejecutar ciertas instrucciones si la condición es verdadera y ejecutar otras instrucciones diferentes si la condición es falsa.

Existen tres tipos de sentencias condicionales: la sentencia **if**, la sentencia **switch** y las sentencias en bucle **while** y **for**. La sentencia **if** indica al navegador si debe ejecutar una parte de código en base al valor lógico de una expresión condicional. La sentencia **switch** compara el valor de una variable con una serie de valores conocidos. Si uno de los valores conocidos coincide con el valor de la variable, se ejecuta el código asociado a dicho valor conocido. Las sentencias en bucle permiten al navegador ejecutar un fragmento de código de forma repetida mientras la condición sea verdadera.

SENTENCIA IF

La sentencia if es una sentencia fundamental a la hora de realizar controles en la ejecución de los programas. De este modo, el navegador tomará una decisión y ejecutará ciertas instrucciones de forma condicional. Su sintaxis es la siguiente:

```
if(expresión){  
    instrucciones  
}
```

La expresión puede ser una comparación o una expresión lógica que devuelve los valores true o false. Las instrucciones son el código que ejecutaremos si la expresión devuelve el valor true. JavaScript ignora las instrucciones en el caso en que la expresión devuelva el valor false.

El uso de las llaves {} no es obligatorio en el caso en que debamos ejecutar una sola instrucción. De lo contrario, sí que es obligatorio su uso.

La segunda forma de utilizar la sentencia if es agregando la palabra clave else. De este modo podemos ejecutar instrucciones en el caso en que la expresión devuelva el valor false. Así es como definimos una sentencia if-else:

```
if(expresión){  
    instrucciones_si_true  
}else {  
    instrucciones_si_false  
}
```

El diagrama de flujo de control de la sentencia if-else lo vemos en la Figura 2.3:

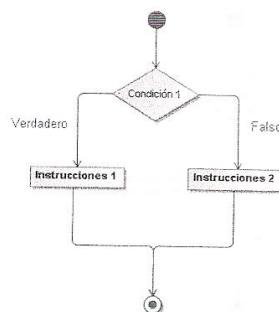


Figura 2.3. Diagrama de flujo de control de la sentencia if-else

La tercera forma de utilizar la sentencia `if` es mediante la sentencia `if-else-if`. La Figura 2.4 muestra su diagrama de flujo de control. De este modo, le pedimos al navegador que evalúe una expresión, y si ésta devuelve el valor `false`, el navegador evaluará una nueva expresión. Si ninguna de las dos o más expresiones utilizadas devuelve el valor `true`, es posible utilizar un último `else`, donde es posible escribir el código que se ejecutará en este caso. La estructura de la sentencia `if-else-if` es la siguiente:

```
if(expresión_1){  
    instrucciones_1  
}else if (expresión_2){  
    instrucciones_2  
}else{  
    instrucciones_3  
}
```



En el caso en que debamos utilizar demasiadas sentencias del tipo `else-if`, es preferible utilizar la sentencia `switch`, que estudiaremos en el siguiente apartado. Este es un error muy común en los programadores principiantes. El navegador no tendrá ningún problema en ejecutar el código, pero la lectura y mantenimiento de una sentencia compleja de varios `if-else` se puede convertir en una tarea difícil.

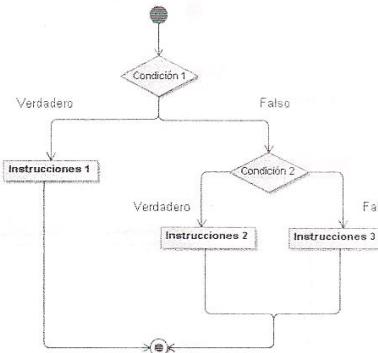


Figura 2.4. Diagrama de flujo de control de la sentencia `if-else-if`

Una vez que conoczamos y dominemos el uso de la sentencia `if`, será posible escribir tipos de código con tomas de decisiones más complejas a través de `if` anidados. Los `if` anidados nos permitirán solucionar problemas que se nos presentan más a menudo, como por ejemplo la toma de decisiones basadas en decisiones precedentes.

SENTENCIA SWITCH

En algunos casos es necesario comparar el valor de una variable con algunos valores conocidos. Para estas situaciones, JavaScript proporciona la sentencia `switch`. En esta sentencia se tiene una expresión a evaluar y una serie de posibles valores de dicha expresión, llamados casos, en los que se encuentran las instrucciones a ejecutar cuando coincidan el valor de la expresión y el valor de cada caso. La sintaxis de esta sentencia es la siguiente:

```
switch (expresión){  
    case valor1:  
        instrucciones a ejecutar si expresión = valor1  
        break;  
    case valor2:  
        instrucciones a ejecutar si expresión = valor2  
        break;  
    case valor3:  
        instrucciones a ejecutar si expresión = valor3  
        break  
    default:  
        instrucciones a ejecutar si expresión es diferente a  
        los valores anteriores  
}
```

Cada caso termina con la palabra clave `break`. La utilidad de esta palabra clave es la de detener la ejecución del `switch` en el momento en que uno de los casos resulte verdadero. De este modo no perdemos tiempo en evaluar los demás casos. El diagrama de flujo de la sentencia `switch` podemos verlo en la Figura 2.5:

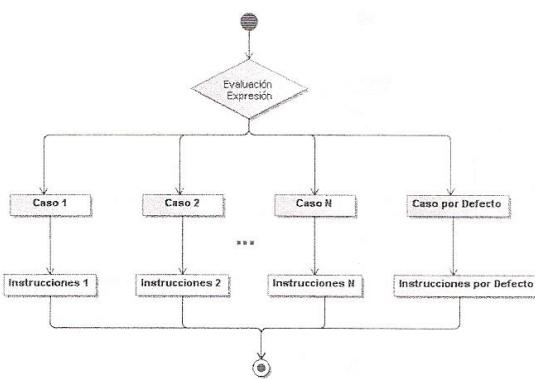


Figura 2.5. Diagrama de flujo de control de la sentencia `switch`

Las instrucciones que se encuentren dentro de la sentencia opcional llamada `default`, se ejecutarán solo cuando ninguno de los valores de cada caso coincida con el valor de la expresión.

BUCLE WHILE

El bucle más sencillo en JavaScript es el bucle `while`. A través de bucles de este tipo, el navegador ejecutará una o más instrucciones continuamente hasta que una cierta condición deje de ser verdadera. La Figura 2.6 muestra su diagrama de flujo. Este bucle es el más utilizado cuando no disponemos de la información que nos indique el número de veces que debemos repetir la iteración del bucle. El bucle `while` consta de tres partes fundamentales: la palabra clave `while`, la expresión condicional y las instrucciones que se ejecutan en el caso en que la expresión condicional sea verdadera. Su sintaxis es la siguiente:

```
while (expresión){  
    instrucciones  
}
```

Una variación del bucle `while` es el denominado `do-while`. Su estructura es la siguiente:

```
do{  
    instrucciones  
} while (expresión)
```

De este modo nos asegura que la ejecución del bucle se lleve a cabo al menos una vez y solo al final, se evalúa si se debe seguir ejecutando o no. Sin embargo, podemos realizar este proceso a través de un bucle `while`. Esta es una práctica recomendable visto que el bucle `do-while` se introdujo solo a partir de la versión 1.2 de JavaScript, con lo cual no todos los navegadores soportan este último bucle.

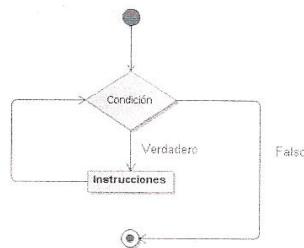


Figura 2.6. Diagrama de flujo de control del bucle `while`

BUCLE FOR

El bucle `for` permite que un navegador ejecute las instrucciones que se encuentren dentro del mismo bucle hasta que la expresión condicional devuelva el valor `false`. Este tipo de sentencia consta de cinco partes: la palabra clave `for`, el valor inicial de la variable de control, la condición basada en dicha variable, el incremento o decremento de la variable y el cuerpo del bucle.

Su sintaxis general es la siguiente:

```
for(valor_inicial_variable; expresión_condicional;
     incremento_o_decremento_de_la_variable) {
    cuerpo_del_bucle
}
```

Tal y como podremos verificar, este bucle es una herramienta bastante potente a la hora de poder realizar diferentes actividades en pocas líneas de código. El diagrama de flujo de este bucle los vemos en la Figura 2.7. En este diagrama, la fase del incremento del contador se efectúa después de llevar a cabo las instrucciones del bucle, sin embargo, este incremento lo podemos realizar inmediatamente después de hacer la comprobación inicial de la condición o en alguna de las mismas instrucciones del bucle.

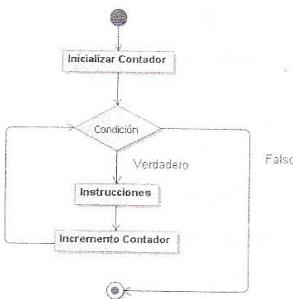


Figura 2.7. Diagrama de flujo de control bucle *for*

Para comprender mejor su funcionamiento, podemos utilizar un ejemplo que muestra en pantalla un valor que se incrementa en cada una de las diez iteraciones del bucle:

```
for(i=0; i<10; i++){
    document.write("El valor de la variable de control es: "
    + i + "<br>")
}
```

ACTIVIDADES 2.6

Abra el fichero Actividad-2.6-if.html e introduzca la información requerida por el navegador. Posteriormente mire el código para que vea un ejemplo del uso de la sentencia condicional if.

RESUMEN DEL CAPÍTULO

En este capítulo hemos presentado una breve introducción de las características principales del lenguaje JavaScript, además de un primer ejemplo clásico denominado “Hola Mundo”, el cual se utiliza en el aprendizaje de cualquier lenguaje de programación. Por otro lado, hemos introducido la sintaxis básica del lenguaje, teniendo en cuenta algunos de los errores más frecuentes en los programadores con poca experiencia.

A continuación, hemos presentado los tipos de datos usados por JavaScript: números, cadenas de texto y valores booleanos. Estos tipos de datos son usados por las variables, las cuales presentan dos fases principales para un correcto funcionamiento: la **declaración** y la **inicialización**.

Posteriormente, hemos presentado todas las categorías de los operadores del lenguaje: operadores aritméticos, operadores lógicos, operadores de asignación, operadores de comparación y operadores condicionales.

Los tipos de datos, las variables y los operadores son conceptos fundamentales en JavaScript. Estos conceptos se pueden definir como la base de cualquier aplicación de este lenguaje.

Por último, hemos introducido las principales sentencias condicionales: `if`, `switch`, `while` y `for`. De cada una de ellas, hemos enseñado las posibles variantes con su respectiva sintaxis. Con estos conceptos se ha aprendido a utilizar un navegador para llevar a cabo tareas de toma de decisiones o ejecución de instrucciones de forma repetitiva.

EJERCICIOS PROPUESTOS

1. Cree un fichero HTML vacío y llámelo EjercicioPropuesto-2.1-Numeros.html. Añada el siguiente código JavaScript en el cuerpo de la página (entre las etiquetas <body> y </body>):

```
<script type="text/javascript">
    var maxValue = Number.MAX_VALUE;
    var minValue = Number.MIN_VALUE;
    alert("Max Value: " + maxValue);
    alert("Min Value: " + minValue);
    alert("Valor especial: " + maxValue*2);
</script>
```

De este modo podremos comprobar el número más grande representable por JavaScript, el número más cercano a cero y el valor especial que representa el infinito.

2. Utilice el siguiente *script* para comprobar el operador lógico &&:

```
<script type="text/javascript">
    var operando_1 =
        eval(prompt("Introduce el primer
                    valor lógico (true o false):", true));
    var operando_2 =
        eval(prompt("Introduce el segundo
                    valor lógico (true o false):", true));
    var resultado_logico = operando_1 &&
    operando_2;
    alert("Resultado: " + resultado_
        logico);
</script>
```

Con este ejercicio pedimos al usuario que introduzca dos valores lógicos y posteriormente compruebe el resultado del operador &&.

3. Utilice el siguiente *script* en un fichero HTML y compruebe su funcionamiento. Posteriormente, cambie la línea countdown-- por countdown++ y concluya explicando por qué el navegador no deja de solicitarnos valores.

```
<script type="text/javascript">
    var countdown = prompt("Introduce un
                           número para
                           iniciar la cuenta atrás: ");
    while (countdown>0){
        alert(countdown+ "... ");
        countdown--;
    }
</script>
```

4. Realice el mismo ejercicio anterior, pero sustituyendo el bucle while por un bucle for.

TEST DE CONOCIMIENTOS

¿Qué navegador web actual incorpora un intérprete para código JavaScript?

- Firefox.
- Safari.
- Chrome.
- Todos los anteriores.

¿El lenguaje JavaScript distingue entre mayúsculas y minúsculas?

- Solamente en el código que se encuentre dentro de las etiquetas `<body>` y `</body>`.
- Solamente en el nombre de las variables.
- Distingue en todo el código JavaScript.
- Solamente en el uso de las palabras clave.

Para terminar una instrucción en JavaScript se utiliza:

- Un punto y coma.
- La palabra clave `end`.
- La etiqueta `</script>`.
- Un punto y coma o un salto de línea.

¿Cuál de los siguientes comentarios es correcto en JavaScript?

```
/ Comentario  
/* Comentario */  
// Comentario  
<!--Comentario-->
```

¿Cuál es el número más grande representable por JavaScript?

- No existe un límite.
- 9.99×10^{308} .
- $1.7976931348623157 \times 10^{308}$.
- Ninguno de los anteriores.

En la expresión `a + b`, ¿qué parte de la expresión son las letras `a` y `b`?

- Operadores.
- Operandos.
- Sumas.
- Incrementos.

¿Cuál es el resultado de la expresión `10<100` ?
'Verdadero' : 'Falso'?

- True.
- `10<100`.
- Falso.
- Verdadero.

¿Para qué sirve la palabra `else` en una sentencia `if-else`?

- Para anidar una sentencia `if`.
- Para definir una nueva expresión condicional.
- Para ejecutar instrucciones en el caso en que la expresión condicional sea falsa.
- Para ejecutar instrucciones en el caso en que la expresión condicional sea verdadera.

¿Qué sentencia condicional permite ejecutar instrucciones independientemente de si una condición sea verdadera o falsa?

- `if`.
- `do-while`.
- `while`.
- `for`.