JavaScript

Introducción de la informatica de JavaScript.

Autor 1: Cristian Ramírez Naranjo

*Risaralda, Universidad Tecnológica de Pereira, Pereira, Colombia*

Correo-e: cristian.ramirez1@utp.edu.co

***Resumen*— JavaScript es un lenguaje de programación que te permite realizar actividades complejas en una página web —  cada vez más una página web hace más cosas que sólo mostrar información estática — como mostrar actualizaciones de contenido en el momento, interactuar con mapas, animaciones gráficas 2D/3D etc. — puedes estar seguro que JavaScript está involucrado. Es la tercera capa del pastel de los estándares en las tecnologías para la web, dos de las cuales son (**[**HTML**](https://developer.mozilla.org/es/docs/Learn/HTML)**y**[**CSS**](https://developer.mozilla.org/es/docs/Learn/CSS)**), hablaremos de ellas más adelante con más detalle en otra parte de nuestra Área de Aprendizaje.**

**Palabras clave- JavaScript:** **JavaScript es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos**,**basado en prototipos, impererativo, débilmente tipado y dinámico.**

***HTML es un lenguaje de programación que se utiliza para el desarrollo de páginas de Internet. Se trata de la siglas que corresponden a HyperText Markup Language, es decir, Lenguaje de Marcas de Hipertexto.***

**CSS (en inglés Cascading Style Sheets) es lo que se denomina lenguaje de hojas de estilo en cascada y se usa para estilizar elementos escritos en un lenguaje de marcado como HTML. CSS separa el contenido de la representación visual del sitio.**

***Abstract*-- JavaScript is a programming language that allows you to perform complex activities on a web page - more and more a web page does more than just display static information - such as displaying content updates at the moment, interacting with maps, 2D graphic animations / 3D etc. - you can be sure that JavaScript is involved. It is the third layer of standards in web technologies, two of which are (HTML and CSS), we will talk about them later in more detail in another part of our Learning Area.**

**Keywords- JavaScript: JavaScript is an interpreted programming language, dialect of the ECMAScript standard. It is defined as object-oriented, based on prototypes, impera- tive, weakly typed and dynamic.**

**HTML is a programming language that is used for the development of Internet pages. This is the acronym that corresponds to HyperText Markup Language, that is, Hypertext Markup Language.**

**CSS (in English Cascading Style Sheets) is what is called cascading style sheet language and is used to stylize elements** **written in a markup language such as HTML. CSS separates the content of the visual representation of the site.**

1. INTRODUCCIÓN

Este documento es una guía de formato o plantilla. Puede obtenerse una copia de la página del curso, o incluso puede buscar por otras versiones semejantes en internet. La idea de esta sección, es dar una introducción al tema que se tratará en el artículo, de forma concisa y que permita al lector prepararse para los contenidos siguientes.

El máximo número de artículos de un mismo autor en una revista se especifica en la convocatoria respectiva.

En la introducción del artículo también se utiliza letra tipo Times New Roman de 10 puntos.

La introducción puede contener:

Un párrafo que describa la justificación y/o antecedentes del problema o temática.

* Javascript **fue creado por la compañía de software “Netscape Corporation”** para que fuese colocado en su navegador 2.0 y que gracias a su simplicidad, aún continúa siendo una de las herramientas de gran utilidad, para la creación de páginas web que posean algo más que texto.
* Otras de las empresas que desarrollo Java Script fue Microsoft, para su Explorer, que por lo general es similar al Netscape.
* Es importante que quede claro, que Javascript **no es del todo un lenguaje de programación**, sino más bien un [**lenguaje de script**](http://www.internetmania.net/int0/int61.htm) (rutinas o guiones). Por lo tanto, es más parecido a los macros de los procesadores de hojas de cálculo o texto. Sería imposible ejecutar un programa completo con Javascript.
* Los Java Script **ayudan a mejorar la gestión cliente/servidor**; entre sus funciones básicas se encuentran: abrir y cerrar ventanas; cambios eficaces en una página (en lo que respecta a su contenido y aspecto; desarrollo de cadenas de texto; procedimientos aritméticos.
* Dado que su misión es extender el HTML, Java Script es un lenguaje que **contempla ciertas restricciones**, que de manera indirecta, terminan por brindarle seguridad al usuario.( <https://programarfacil.com/blog/programacion/introduccion-al-javascript/>)

1. CONTENIDO

**1.2. Breve historia**

A principios de los años 90, la mayoría de usuarios que se conectaban a Internet lo hacían con módems a una velocidad máxima de 28.8 kbps. En esa época, empezaban a desarrollarse las primeras aplicaciones web y por tanto, las páginas web comenzaban a incluir formularios complejos.

Con unas aplicaciones web cada vez más complejas y una velocidad de navegación tan lenta, surgió la necesidad de un lenguaje de programación que se ejecutara en el navegador del usuario. De esta forma, si el usuario no rellenaba correctamente un formulario, no se le hacía esperar mucho tiempo hasta que el servidor volviera a mostrar el formulario indicando los errores existentes.

**Brendan Eich**, un programador que trabajaba en Netscape, pensó que podría solucionar este problema adaptando otras tecnologías existentes (como *ScriptEase*) al navegador Netscape Navigator 2.0, que iba a lanzarse en 1995. Inicialmente, Eich denominó a su lenguaje *LiveScript*.

Posteriormente, Netscape firmó una alianza con Sun Microsystems para el desarrollo del nuevo lenguaje de programación. Además, justo antes del lanzamiento Netscape decidió cambiar el nombre por el de JavaScript. La razón del cambio de nombre fue exclusivamente por marketing, ya que Java era la palabra de moda en el mundo informático y de Internet de la época.

La primera versión de JavaScript fue un completo éxito y Netscape Navigator 3.0 ya incorporaba la siguiente versión del lenguaje, la versión 1.1. Al mismo tiempo, Microsoft lanzó JScript con su navegador Internet Explorer 3. JScript era una copia de JavaScript al que le cambiaron el nombre para evitar problemas legales.

Para evitar una guerra de tecnologías, Netscape decidió que lo mejor sería estandarizar el lenguaje JavaScript. De esta forma, en 1997 se envió la especificación JavaScript 1.1 al organismo ECMA *European Computer Manufacturers Association*).

ECMA creó el comité TC39 con el objetivo de *"estandarizar de un lenguaje de script multiplataforma e independiente de cualquier empresa"*. El primer estándar que creó el comité TC39 se denominó **ECMA-262**, en el que se definió por primera vez el lenguaje ECMAScript.

Por este motivo, algunos programadores prefieren la denominación *ECMAScript* para referirse al lenguaje JavaScript. De hecho, JavaScript no es más que la implementación que realizó la empresa Netscape del estándar ECMAScript.

La organización internacional para la estandarización (ISO) adoptó el estándar ECMA-262 a través de su comisión IEC, dando lugar al estándar ISO/IEC-16262.

# 1.3. Especificaciones oficiales

ECMA ha publicado varios estándares relacionados con ECMAScript. En Junio de 1997 se publicó la primera edición del estándar ECMA-262. Un año después, en Junio de 1998 se realizaron pequeñas modificaciones para adaptarlo al estandar ISO/IEC-16262 y se creó la segunda edición.

La tercera edición del estándar ECMA-262 (publicada en Diciembre de 1999) es la versión que utilizan los navegadores actuales y se puede consultar gratuitamente en http://www.ecma-international.org/publications/standards/Ecma-262.htm

Actualmente se encuentra en desarrollo la cuarta versión de ECMA-262, que podría incluir novedades como paquetes, namespaces, definición explícita de clases, etc.

ECMA también ha definido varios estándares relacionados con ECMAScript, como el estándar ECMA-357, que define una extensión conocida como E4X y que permite la integración de JavaScript y XML.

# 1.4. Cómo incluir JavaScript en documentos XHTML

La integración de JavaScript y XHTML es muy flexible, ya que existen al menos tres formas para incluir código JavaScript en las páginas web.

### 1.4.1. Incluir JavaScript en el mismo documento XHTML

El código JavaScript se encierra entre etiquetas <script> y se incluye en cualquier parte del documento. Aunque es correcto incluir cualquier bloque de código en cualquier zona de la página, se recomienda definir el código JavaScript dentro de la cabecera del documento (dentro de la etiqueta <head>):

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />

<title>Ejemplo de código JavaScript en el propio documento</title>

<script type="text/javascript">

alert("Un mensaje de prueba");

</script>

</head>

<body>

<p>Un párrafo de texto.</p>

</body>

</html>

Para que la página XHTML resultante sea válida, es necesario añadir el atributo type a la etiqueta <script>. Los valores que se incluyen en el atributo type están estandarizados y para el caso de JavaScript, el valor correcto es text/javascript.

Este método se emplea cuando se define un bloque pequeño de código o cuando se quieren incluir instrucciones específicas en un determinado documento HTML que completen las instrucciones y funciones que se incluyen por defecto en todos los documentos del sitio web.

El principal inconveniente es que si se quiere hacer una modificación en el bloque de código, es necesario modificar todas las páginas que incluyen ese mismo bloque de código JavaScript.

### 1.4.2. Definir JavaScript en un archivo externo

Las instrucciones JavaScript se pueden incluir en un archivo externo de tipo JavaScript que los documentos XHTML enlazan mediante la etiqueta <script>. Se pueden crear todos los archivos JavaScript que sean necesarios y cada documento XHTML puede enlazar tantos archivos JavaScript como necesite.

Ejemplo:

Archivo codigo.js

alert("Un mensaje de prueba");

Documento XHTML

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />

<title>Ejemplo de código JavaScript en el propio documento</title>

<script type="text/javascript" src="/js/codigo.js"></script>

</head>

<body>

<p>Un párrafo de texto.</p>

</body>

</html>

Además del atributo type, este método requiere definir el atributo src, que es el que indica la URL correspondiente al archivo JavaScript que se quiere enlazar. Cada etiqueta <script> solamente puede enlazar un único archivo, pero en una misma página se pueden incluir tantas etiquetas <script> como sean necesarias.

Los archivos de tipo JavaScript son documentos normales de texto con la extensión .js, que se pueden crear con cualquier editor de texto como Notepad, Wordpad, EmEditor, UltraEdit, Vi, etc.

La principal ventaja de enlazar un archivo JavaScript externo es que se simplifica el código XHTML de la página, que se puede reutilizar el mismo código JavaScript en todas las páginas del sitio web y que cualquier modificación realizada en el archivo JavaScript se ve reflejada inmediatamente en todas las páginas XHTML que lo enlazan.

### 1.4.3. Incluir JavaScript en los elementos XHTML

Este último método es el menos utilizado, ya que consiste en incluir trozos de JavaScript dentro del código XHTML de la página:

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />

<title>Ejemplo de código JavaScript en el propio documento</title>

</head>

<body>

<p onclick="alert('Un mensaje de prueba')">Un párrafo de texto.</p>

</body>

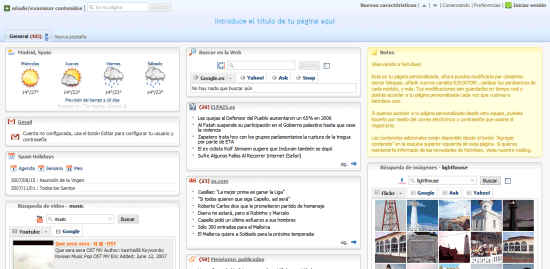
</html>

El mayor inconveniente de este método es que ensucia innecesariamente el código XHTML de la página y complica el mantenimiento del código JavaScript. En general, este método sólo se utiliza para definir algunos eventos y en algunos otros casos especiales, como se verá más adelante.

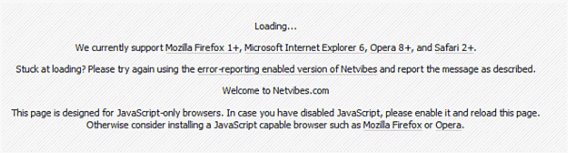
**1.5. Etiqueta noscript**

Algunos navegadores no disponen de soporte completo de JavaScript, otros navegadores permiten bloquearlo parcialmente e incluso algunos usuarios bloquean completamente el uso de JavaScript porque creen que así navegan de forma más segura.

En estos casos, es habitual que si la página web requiere JavaScript para su correcto funcionamiento, se incluya un mensaje de aviso al usuario indicándole que debería activar JavaScript para disfrutar completamente de la página. El siguiente ejemplo muestra una página web basada en JavaScript cuando se accede con JavaScript activado y cuando se accede con JavaScript completamente desactivado.



**Figura 1.1** Imagen de www.Netvibes.com con JavaScript activado



**Figura 1.2** Imagen de www.Netvibes.com con JavaScript desactivado

El lenguaje HTML define la etiqueta <noscript> para mostrar un mensaje al usuario cuando su navegador no puede ejecutar JavaScript. El siguiente código muestra un ejemplo del uso de la etiqueta <noscript>:

<head> ... </head>

<body>

<noscript>

<p>Bienvenido a Mi Sitio</p>

<p>La página que estás viendo requiere para su funcionamiento el uso de JavaScript.

Si lo has deshabilitado intencionadamente, por favor vuelve a activarlo.</p>

</noscript>

</body>

La etiqueta <noscript> se debe incluir en el interior de la etiqueta <body> (normalmente se incluye al principio de <body>). El mensaje que muestra <noscript> puede incluir cualquier elemento o etiqueta XHTML.

# 1.6. Glosario básico

**Script**: 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. A veces se traduce al español directamente como "guión", aunque script es una palabra más adecuada y comúnmente aceptada.

**Sentencia**: cada una de las instrucciones que forman un script.

**Palabras reservadas**: son las palabras (en inglés) que se utilizan para construir las sentencias de JavaScript y que por tanto no pueden ser utilizadas libremente. Las palabras actualmente reservadas por JavaScript son: break, case, catch, continue, default, delete, do, else, finally, for, function, if, in, instanceof, new, return, switch, this, throw, try, typeof, var, void, while, with.

# 1.7. 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 se define 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.

JavaScript define dos tipos de comentarios: los de una sola línea y los que ocupan varias líneas.

Ejemplo de comentario de una sola línea:

*// a continuación se muestra un mensaje*

alert("mensaje de prueba");

Los comentarios de una sola línea se definen añadiendo dos barras oblicuas (//) al principio de la línea.

Ejemplo de comentario de varias líneas:

*/\* Los comentarios de varias líneas son muy útiles*

*cuando se necesita incluir bastante información*

*en los comentarios \*/*

alert("mensaje de prueba");

Los comentarios multilínea se definen encerrando el texto del comentario entre los símbolos /\* y \*/.

# 1.8. Posibilidades y limitaciones

Desde su aparición, JavaScript siempre fue utilizado de forma masiva por la mayoría de sitios de Internet. La aparición de Flash disminuyó su popularidad, ya que Flash permitía realizar algunas acciones imposibles de llevar a cabo mediante JavaScript.

Sin embargo, la aparición de las aplicaciones AJAX programadas con JavaScript le ha devuelto una popularidad sin igual dentro de los lenguajes de programación web.

En cuanto a las limitaciones, JavaScript fue diseñado de forma que se ejecutara en un entorno muy limitado que permitiera a los usuarios confiar en la ejecución de los scripts.

De esta forma, los scripts de JavaScript no pueden comunicarse con recursos que no pertenezcan al mismo dominio desde el que se descargó el script. Los scripts tampoco pueden cerrar ventanas que no hayan abierto esos mismos scripts. Las ventanas que se crean no pueden ser demasiado pequeñas ni demasiado grandes ni colocarse fuera de la vista del usuario (aunque los detalles concretos dependen de cada navegador).

Además, los scripts no pueden acceder a los archivos del ordenador del usuario (ni en modo lectura ni en modo escritura) y tampoco pueden leer o modificar las preferencias del navegador.

Por último, si la ejecución de un script dura demasiado tiempo (por ejemplo por un error de programación) el navegador informa al usuario de que un script está consumiendo demasiados recursos y le da la posibilidad de detener su ejecución.

A pesar de todo, existen alternativas para poder saltarse algunas de las limitaciones anteriores. La alternativa más utilizada y conocida consiste en firmar digitalmente el script y solicitar al usuario el permiso para realizar esas acciones.

# 1.9. JavaScript y navegadores

Los navegadores más modernos disponibles actualmente incluyen soporte de JavaScript hasta la versión correspondiente a la tercera edición del estándar ECMA-262.

La mayor diferencia reside en el dialecto utilizado, ya que mientras Internet Explorer utiliza JScript, el resto de navegadores (Firefox, Opera, Safari, Konqueror) utilizan JavaScript.

# 1.10. JavaScript en otros entornos

La inigualable popularidad de JavaScript como lenguaje de programación de aplicaciones web se ha extendido a otras aplicaciones y otros entornos no relacionados con la web.

Herramientas como Adobe Acrobat permiten incluir código JavaScript en archivos PDF. Otras herramientas de Adobe como Flash y Flex utilizan ActionScript, un dialecto del mismo estándar de JavaScript.

Photoshop permite realizar pequeños scripts mediante JavaScript y la versión 6 de Java incluye un nuevo paquete (denominado javax.script) que permite integrar ambos lenguajes.

Por último, aplicaciones como [Yahoo Widgets](https://www.yahoo.com/) y el [Dashboard de Apple](https://www.apple.com/downloads/dashboard/) utilizan JavaScript para programar sus widgets.

Introducción a JavaScript

Capítulo 2. El primer script

A continuación, se muestra un primer script sencillo pero completo:

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />

<title>El primer script</title>

<script type="text/javascript">

alert("Hola Mundo!");

</script>

</head>

<body>

<p>Esta página contiene el primer script</p>

</body>

</html>

En este ejemplo, el script se incluye como un bloque de código dentro de una página XHTML. Por tanto, en primer lugar se debe crear una página XHTML correcta que incluya la declaración del DOCTYPE, el atributo xmlns, las secciones <head> y <body>, la etiqueta <title>, etc.

Aunque el código del script se puede incluir en cualquier parte de la página, se recomienda incluirlo en la cabecera del documento, es decir, dentro de la etiqueta <head>.

A continuación, el código JavaScript se debe incluir entre las etiquetas <script>...</script>. Además, para que la página sea válida, es necesario definir el atributo type de la etiqueta <script>. Técnicamente, el atributo type se corresponde con "el tipo MIME", que es un estándar para identificar los diferentes tipos de contenidos. El "tipo MIME" correcto para JavaScript es text/javascript.

Una vez definida la zona en la que se incluirá el script, se escriben todas las sentencias que forman la aplicación. Este primer ejemplo es tan sencillo que solamente incluye una sentencia: alert("Hola Mundo!");.

La instrucción alert() es una de las utilidades que incluye JavaScript y permite mostrar un mensaje en la pantalla del usuario. Si se visualiza la página web de este primer script en cualquier navegador, automáticamente se mostrará una ventana con el mensaje "Hola Mundo!".

A continuación se muestra el resultado de ejecutar el script en diferentes navegadores:

Mensaje mostrado con "alert()" en Internet Explorer

Figura 2.1 Mensaje mostrado con "alert()" en Internet Explorer

Mensaje mostrado con "alert()" en Firefox

Figura 2.2 Mensaje mostrado con "alert()" en Firefox

Mensaje mostrado con "alert()" en Opera

Figura 2.3 Mensaje mostrado con "alert()" en Opera

Como se puede observar en las imágenes anteriores, el funcionamiento de la utilidad alert() en los distintos navegadores es idéntico. Sin embargo, existen grandes diferencias visuales en la forma en la que se presentan los mensajes.

# Capítulo 3. Programación básica

Antes de comenzar a desarrollar programas y utilidades con JavaScript, es necesario conocer los elementos básicos con los que se construyen las aplicaciones. Si ya sabes programar en algún lenguaje de programación, este capítulo te servirá para conocer la sintaxis específica de JavaScript.

Si nunca has programado, este capítulo explica en detalle y comenzando desde cero los conocimientos básicos necesarios para poder entender posteriormente la programación avanzada, que es la que se utiliza para crear las aplicaciones reales.

**3.1. 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.

Si no existieran variables, un programa que suma dos números podría escribirse como:

resultado = 3 + 1

El *programa* anterior es tan poco útil que sólo sirve para el caso en el que el primer número de la suma sea el 3 y el segundo número sea el 1. En cualquier otro caso, el *programa* obtiene un resultado incorrecto.

Sin embargo, el programa se puede rehacer de la siguiente manera utilizando variables para almacenar y referirse a cada número:

numero\_1 = 3

numero\_2 = 1

resultado = numero\_1 + numero\_2

Los elementos numero\_1 y numero\_2 son **variables** que almacenan los valores que utiliza el programa. El resultado se calcula siempre en función del valor almacenado por las variables, por lo que este programa funciona correctamente para cualquier par de números indicado. Si se modifica el valor de las variables numero\_1 y numero\_2, el programa sigue funcionando correctamente.

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 numero\_1 = 3;

var numero\_2 = 1;

var resultado = numero\_1 + numero\_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 numero\_1 = 3;

var numero\_2 = 1;

var resultado = var numero\_1 + var numero\_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 numero\_1;

var numero\_2;

numero\_1 = 3;

numero\_2 = 1;

var resultado = numero\_1 + numero\_2;

Una de las características más sorprendentes de JavaSript 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 numero\_1 = 3;

var numero\_2 = 1;

resultado = numero\_1 + numero\_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. De la misma forma, también sería correcto el siguiente código:

numero\_1 = 3;

numero\_2 = 1;

resultado = numero\_1 + numero\_2;

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;

Sin embargo, las siguientes variables tienen identificadores incorrectos:

var 1numero; *// Empieza por un número*

var numero;1\_123; *// Contiene un carácter ";"*

# 3.2. Tipos de variables

Aunque todas las variables de JavaScript se crean de la misma forma (mediante la palabra reservada var), la forma en la que se les asigna un valor depende del tipo de valor que se quiere almacenar (números, textos, etc.)

### 3.2.1. 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*

### 3.2.2. 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';

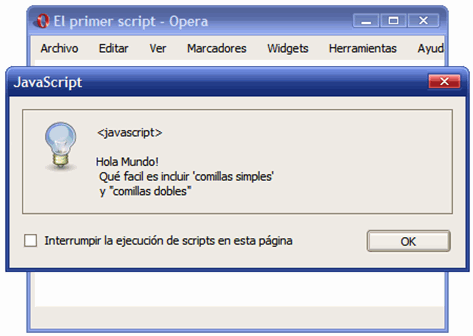
var texto2 = "Una frase con \"comillas dobles\" dentro";

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

**Ejercicio 2**

Modificar el primer script del capítulo anterior para que:

1. El mensaje que se muestra al usuario se almacene en una variable llamada mensaje y el funcionamiento del script sea el mismo.
2. El mensaje mostrado sea el de la siguiente imagen:



**Figura 3.1** Nuevo mensaje que debe mostrar el script

[Ver solución](http://www.librosweb.es/javascript/capitulo_11/ejercicio_2.html)

### 3.2.3. 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 0y 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.

**Ejercicio 3**

Crear un array llamado meses y que almacene el nombre de los doce meses del año. Mostrar por pantalla los doce nombres utilizando la función alert().

[Ver solución](http://www.librosweb.es/javascript/capitulo_11/ejercicio_3.html)

### 3.2.4. 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.

Los únicos valores que pueden almacenar estas variables son true y false, por lo que no pueden utilizarse los valores verdadero y falso. A continuación se muestra un par de variables de tipo booleano:

var clienteRegistrado = false;

var ivaIncluido = true;

# 3.3. 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.

### 3.3.1. 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 numero1 = 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 numero1 = 3;

var numero2 = 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;

### 3.3.2. 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).

### 3.3.3. 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.

#### 3.3.3.1. 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 trueen 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*

#### 3.3.3.2. 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*

#### 3.3.3.3. 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*

### 3.3.4. 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*

### 3.3.5. 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.)

# 3.4. Estructuras de control de flujo

Los programas que se pueden realizar utilizando solamente variables y operadores son una simple sucesión lineal de instrucciones básicas.

Sin embargo, no se pueden realizar programas que muestren un mensaje si el valor de una variable es igual a un valor determinado y no muestren el mensaje en el resto de casos. Tampoco se puede repetir de forma eficiente una misma instrucción, como por ejemplo sumar un determinado valor a todos los elementos de un array.

Para realizar este tipo de programas son necesarias las **estructuras de control de flujo**, que son instrucciones del tipo "si se cumple esta condición, hazlo; si no se cumple, haz esto otro". También existen instrucciones del tipo "repite esto mientras se cumpla esta condición".

Si se utilizan estructuras de control de flujo, los programas dejan de ser una sucesión lineal de instrucciones para convertirse en programas inteligentes que pueden tomar decisiones en función del valor de las variables.

### 3.4.1. 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().

**Ejercicio 5**

Completar las condiciones de los if del siguiente script para que los mensajes de los alert() se muestren siempre de forma correcta:

var numero1 = 5;

var numero2 = 8;

if(...) {

alert("numero1 no es mayor que numero2");

}

if(...) {

alert("numero2 es positivo");

}

if(...) {

alert("numero1 es negativo o distinto de cero");

}

if(...) {

alert("Incrementar en 1 unidad el valor de numero1 no lo hace mayor o igual que numero2");

}

[Ver solución](http://www.librosweb.es/javascript/capitulo11/ejercicio_5.html)

### 3.4.2. 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().

**Ejercicio 6**

El cálculo de la letra del Documento Nacional de Identidad (DNI) es un proceso matemático sencillo que se basa en obtener el resto de la división entera del número de DNI y el número 23. A partir del resto de la división, se obtiene la letra seleccionándola dentro de un array de letras.

El array de letras es:

var letras = ['T', 'R', 'W', 'A', 'G', 'M', 'Y', 'F', 'P', 'D', 'X', 'B', 'N', 'J', 'Z', 'S', 'Q', 'V', 'H', 'L', 'C', 'K', 'E', 'T'];

Por tanto si el resto de la división es 0, la letra del DNI es la T y si el resto es 3 la letra es la A. Con estos datos, elaborar un pequeño script que:

1. Almacene en una variable el número de DNI indicado por el usuario y en otra variable la letra del DNI que se ha indicado. (Pista: si se quiere pedir directamente al usuario que indique su número y su letra, se puede utilizar la función prompt())
2. En primer lugar (y en una sola instrucción) se debe comprobar si el número es menor que 0 o mayor que 99999999. Si ese es el caso, se muestra un mensaje al usuario indicando que el número proporcionado no es válido y el programa no muestra más mensajes.
3. Si el número es válido, se calcula la letra que le corresponde según el método explicado anteriormente.
4. Una vez calculada la letra, se debe comparar con la letra indicada por el usuario. Si no coinciden, se muestra un mensaje al usuario diciéndole que la letra que ha indicado no es correcta. En otro caso, se muestra un mensaje indicando que el número y la letra de DNI son correctos.

[Ver solución](http://www.librosweb.es/javascript/capitulo_11/ejercicio_6.html)

### 3.4.3. 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]);

}

**Ejercicio 7**

El factorial de un número entero n es una operación matemática que consiste en multiplicar todos los factores n x (n-1) x (n-2) x ... x 1. Así, el factorial de 5 (escrito como 5!) es igual a: 5! = 5 x 4 x 3 x 2 x 1 = 120

Utilizando la estructura for, crear un script que calcule el factorial de un número entero.

[Ver solución](http://www.librosweb.es/javascript/capitulo_11/ejercicio_7.html)

### 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.

# 3.5. Funciones y propiedades básicas de JavaScript

JavaScript incorpora una serie de herramientas y utilidades (llamadas funciones y propiedades, como se verá más adelante) para el manejo de las variables. De esta forma, muchas de las operaciones básicas con las variables, se pueden realizar directamente con las utilidades que ofrece JavaScript.

### 3.5.1. Funciones útiles para cadenas de texto

A continuación se muestran algunas de las funciones más útiles para el manejo de cadenas de texto:

length, calcula la longitud de una cadena de texto (el número de caracteres que la forman)

var mensaje = "Hola Mundo";

var numeroLetras = mensaje.length; *// numeroLetras = 10*

+, se emplea para concatenar varias cadenas de texto

var mensaje1 = "Hola";

var mensaje2 = " Mundo";

var mensaje = mensaje1 + mensaje2; *// mensaje = "Hola Mundo"*

Además del operador +, también se puede utilizar la función concat()

var mensaje1 = "Hola";

var mensaje2 = mensaje1.concat(" Mundo"); *// mensaje2 = "Hola Mundo"*

Las cadenas de texto también se pueden unir con variables numéricas:

var variable1 = "Hola ";

var variable2 = 3;

var mensaje = variable1 + variable2; *// mensaje = "Hola 3"*

Cuando se unen varias cadenas de texto es habitual olvidar añadir un espacio de separación entre las palabras:

var mensaje1 = "Hola";

var mensaje2 = "Mundo";

var mensaje = mensaje1 + mensaje2; *// mensaje = "HolaMundo"*

Los espacios en blanco se pueden añadir al final o al principio de las cadenas y también se pueden indicar forma explícita:

var mensaje1 = "Hola";

var mensaje2 = "Mundo";

var mensaje = mensaje1 + " " + mensaje2; *// mensaje = "Hola Mundo"*

toUpperCase(), transforma todos los caracteres de la cadena a sus correspondientes caracteres en mayúsculas:

var mensaje1 = "Hola";

var mensaje2 = mensaje1.toUpperCase(); *// mensaje2 = "HOLA"*

toLowerCase(), transforma todos los caracteres de la cadena a sus correspondientes caracteres en minúsculas:

var mensaje1 = "HolA";

var mensaje2 = mensaje1.toLowerCase(); *// mensaje2 = "hola"*

charAt(posicion), obtiene el carácter que se encuentra en la posición indicada:

var mensaje = "Hola";

var letra = mensaje.charAt(0); *// letra = H*

letra = mensaje.charAt(2); *// letra = l*

indexOf(caracter), calcula la posición en la que se encuentra el carácter indicado dentro de la cadena de texto. Si el carácter se incluye varias veces dentro de la cadena de texto, se devuelve su primera posición empezando a buscar desde la izquierda. Si la cadena no contiene el carácter, la función devuelve el valor -1:

var mensaje = "Hola";

var posicion = mensaje.indexOf('a'); *// posicion = 3*

posicion = mensaje.indexOf('b'); *// posicion = -1*

Su función análoga es lastIndexOf():

lastIndexOf(caracter), calcula la última posición en la que se encuentra el carácter indicado dentro de la cadena de texto. Si la cadena no contiene el carácter, la función devuelve el valor -1:

var mensaje = "Hola";

var posicion = mensaje.lastIndexOf('a'); *// posicion = 3*

posicion = mensaje.lastIndexOf('b'); *// posicion = -1*

La función lastIndexOf() comienza su búsqueda desde el final de la cadena hacia el principio, aunque la posición devuelta es la correcta empezando a contar desde el principio de la palabra.

substring(inicio, final), extrae una porción de una cadena de texto. El segundo parámetro es opcional. Si sólo se indica el parámetro inicio, la función devuelve la parte de la cadena original correspondiente desde esa posición hasta el final:

var mensaje = "Hola Mundo";

var porcion = mensaje.substring(2); *// porcion = "la Mundo"*

porcion = mensaje.substring(5); *// porcion = "Mundo"*

porcion = mensaje.substring(7); *// porcion = "ndo"*

Si se indica un inicio negativo, se devuelve la misma cadena original:

var mensaje = "Hola Mundo";

var porcion = mensaje.substring(-2); *// porcion = "Hola Mundo"*

Cuando se indica el inicio y el final, se devuelve la parte de la cadena original comprendida entre la posición inicial y la inmediatamente anterior a la posición final (es decir, la posición inicio está incluida y la posición final no):

var mensaje = "Hola Mundo";

var porcion = mensaje.substring(1, 8); *// porcion = "ola Mun"*

porcion = mensaje.substring(3, 4); *// porcion = "a"*

Si se indica un final más pequeño que el inicio, JavaScript los considera de forma inversa, ya que automáticamente asigna el valor más pequeño al inicio y el más grande al final:

var mensaje = "Hola Mundo";

var porcion = mensaje.substring(5, 0); *// porcion = "Hola "*

porcion = mensaje.substring(0, 5); *// porcion = "Hola "*

split(separador), convierte una cadena de texto en un array de cadenas de texto. La función parte la cadena de texto determinando sus trozos a partir del carácter separador indicado:

var mensaje = "Hola Mundo, soy una cadena de texto!";

var palabras = mensaje.split(" ");

*// palabras = ["Hola", "Mundo,", "soy", "una", "cadena", "de", "texto!"];*

Con esta función se pueden extraer fácilmente las letras que forman una palabra:

var palabra = "Hola";

var letras = palabra.split(""); *// letras = ["H", "o", "l", "a"]*

### 3.5.2. Funciones útiles para arrays

A continuación se muestran algunas de las funciones más útiles para el manejo de arrays:

length, calcula el número de elementos de un array

var vocales = ["a", "e", "i", "o", "u"];

var numeroVocales = vocales.length; *// numeroVocales = 5*

concat(), se emplea para concatenar los elementos de varios arrays

var array1 = [1, 2, 3];

array2 = array1.concat(4, 5, 6); *// array2 = [1, 2, 3, 4, 5, 6]*

array3 = array1.concat([4, 5, 6]); *// array3 = [1, 2, 3, 4, 5, 6]*

join(separador), es la función contraria a split(). Une todos los elementos de un array para formar una cadena de texto. Para unir los elementos se utiliza el carácter separador indicado

var array = ["hola", "mundo"];

var mensaje = array.join(""); *// mensaje = "holamundo"*

mensaje = array.join(" "); *// mensaje = "hola mundo"*

pop(), elimina el último elemento del array y lo devuelve. El array original se modifica y su longitud disminuye en 1 elemento.

var array = [1, 2, 3];

var ultimo = array.pop();

*// ahora array = [1, 2], ultimo = 3*

push(), añade un elemento al final del array. El array original se modifica y aumenta su longitud en 1 elemento. (También es posible añadir más de un elemento a la vez)

var array = [1, 2, 3];

array.push(4);

*// ahora array = [1, 2, 3, 4]*

shift(), elimina el primer elemento del array y lo devuelve. El array original se ve modificado y su longitud disminuida en 1 elemento.

var array = [1, 2, 3];

var primero = array.shift();

*// ahora array = [2, 3], primero = 1*

unshift(), añade un elemento al principio del array. El array original se modifica y aumenta su longitud en 1 elemento. (También es posible añadir más de un elemento a la vez)

var array = [1, 2, 3];

array.unshift(0);

*// ahora array = [0, 1, 2, 3]*

reverse(), modifica un array colocando sus elementos en el orden inverso a su posición original:

var array = [1, 2, 3];

array.reverse();

*// ahora array = [3, 2, 1]*

### 3.5.3. Funciones útiles para números

A continuación se muestran algunas de las funciones y propiedades más útiles para el manejo de números.

NaN, (del inglés, "Not a Number") JavaScript emplea el valor NaN para indicar un valor numérico no definido (por ejemplo, la división 0/0).

var numero1 = 0;

var numero2 = 0;

alert(numero1/numero2); *// se muestra el valor NaN*

isNaN(), permite proteger a la aplicación de posibles valores numéricos no definidos

var numero1 = 0;

var numero2 = 0;

if(isNaN(numero1/numero2)) {

alert("La división no está definida para los números indicados");

}

else {

alert("La división es igual a => " + numero1/numero2);

}

Infinity, hace referencia a un valor numérico infinito y positivo (también existe el valor –Infinity para los infinitos negativos)

var numero1 = 10;

var numero2 = 0;

alert(numero1/numero2); *// se muestra el valor Infinity*

toFixed(digitos), devuelve el número original con tantos decimales como los indicados por el parámetro digitos y realiza los redondeos necesarios. Se trata de una función muy útil por ejemplo para mostrar precios.

var numero1 = 4564.34567;

numero1.toFixed(2); *// 4564.35*

numero1.toFixed(6); *// 4564.345670*

numero1.toFixed(); *// 4564(*<https://uniwebsidad.com/libros/javascript/capitulo-1/breve-historia>)

TABLA I

Imagen ejemplo 1

Imagen que contiene captura de pantalla

Descripción generada automáticamente

Imagen ejemplo 2.

Imagen que contiene captura de pantalla

Descripción generada automáticamente

Imagen ejemplo 3.

Imagen que contiene captura de pantalla

Descripción generada automáticamente

Imágenes tomadas de (<https://www.google.com/search?q=programas+hechos+en+javascript&rlz=1C1CHBF_esCO844CO844&source=lnms&tbm=isch&sa=X&ved=0ahUKEwjN-IWdw7_iAhXRxFkKHanaD8sQ_AUIDygC&biw=1366&bih=657#imgrc=pZJ52H018EFQPM:>

)

1. CONCLUSIONES

JavaScript es una ayuda indispensable para realizar páginas en las que queramos una alta interacción con el usuario, esto nos proporciona una serie de eventos que reconoce, unas librerias de funciones y métodos y todo lo que queramos hacer con un lenguaje: bucles de condición, variable,etc. Todo con la ventaja de que se ejecuta en el cliente y no tiene que estar llamando continuamente al servidor por lo que descarga el trabajo de este y hace que la navegación sea más rápida (<https://es.slideshare.net/guest044583b/javascript-1562899>)

RECOMENDACIONES

Agradezco al profesor de introducción José Gilberto Vargas Cano por enseñarme y guiarme a la realización de un buen paper y por su entrega como educador de la Universidad Tecnológica de Pereira.

REFERENCIAS

Referencias de internet

<http://www.maestrosdelweb.com/que-es-javascript/>

<https://uniwebsidad.com/libros/javascript/capitulo-1/breve-historia>

<https://programarfacil.com/blog/programacion/introduccion-al-javascript/>

<https://www.google.com/search?q=programas+hechos+en+javascript&rlz=1C1CHBF_esCO844CO844&source=lnms&tbm=isch&sa=X&ved=0ahUKEwjN-IWdw7_iAhXRxFkKHanaD8sQ_AUIDygC&biw=1366&bih=657#imgrc=pZJ52H018EFQPM:>

<https://www.hostinger.co/tutoriales/que-es-javascript-introduccion-basica/>

<https://es.slideshare.net/guest044583b/javascript-1562899>

**Observaciones generales:**