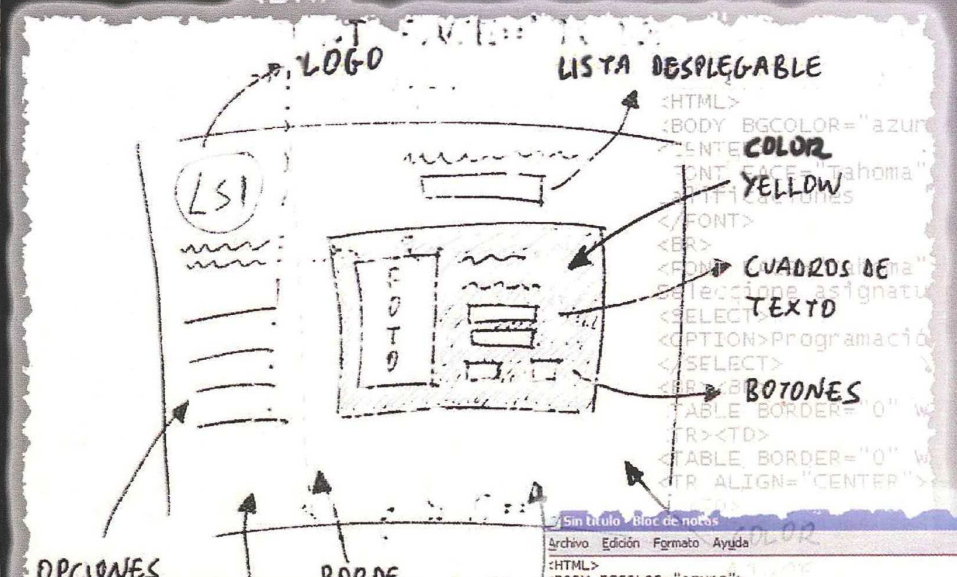
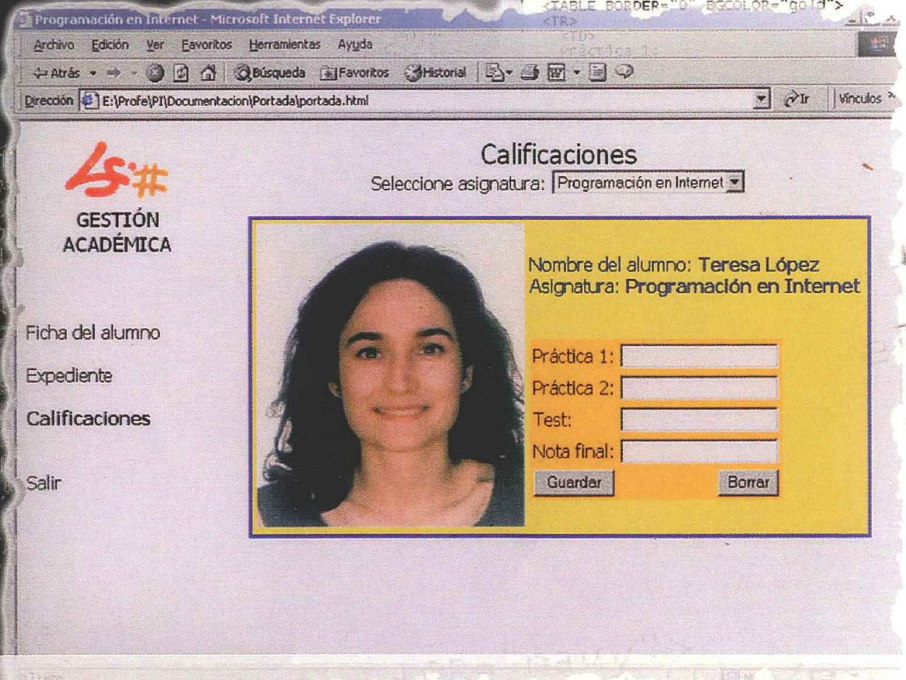
Programación de aplicaciones web: Historia, Principios básicos y Clientes web.

Sergio Luján Mora




```
<HTML>
<BODY BGCOLOR="azure">
<CENTER>
<FONT FACE="Tahoma" SIZE="5">
Calificaciones
</FONT>
<BR>
<FONT FACE="Tahoma">
Seleccione asignatura:
<SELECT>
<OPTION>Programación en Internet
</SELECT>
<BR><BR>
<TABLE BORDER="0" WIDTH="100%" BGCOLOR="blue">
<TR><TD>
<TABLE BORDER="0" WIDTH="100%" BGCOLOR="yellow">
<TR ALIGN="CENTER">
  <TD>
  <IMG SRC="dni.jpg">
  </TD>
  <TD ALIGN="LEFT">
  <FONT FACE="Tahoma" COLOR="navy">
  Nombre del alumno: <B>Teresa López</B>
  <BR>
  Asignatura: <B>Programación en Internet</B>
  <BR><BR>
  <TABLE BORDER="0" BGCOLOR="gold">
  <TR>
```



ECU
EDITORIAL CLUB UNIVERSITARIO

Aunque los inicios de Internet se remontan a los años sesenta, no ha sido hasta los años noventa cuando, gracias a la Web, se ha extendido su uso por todo el mundo. En pocos años la Web ha evolucionado enormemente: se ha pasado de páginas sencillas, con pocas imágenes y contenidos estáticos a páginas complejas con contenidos dinámicos que provienen de bases de datos, lo que permite la creación de "aplicaciones web".

De forma breve, una aplicación Web se puede definir como una aplicación en la cual el usuario por medio de un navegador realiza peticiones a una aplicación remota accesible a través de Internet (o a través de una Intranet) y que recibe una respuesta que se muestra en el propio navegador.

El contenido de este libro se estructura en dos partes. En la primera parte del libro se tratan temas introductorios a la programación de aplicaciones Web: un breve repaso de la historia de Internet y de la web, características de las arquitecturas cliente/servidor, el concepto de aplicación Web y la estructura de un sitio web tanto a nivel físico como lógico.

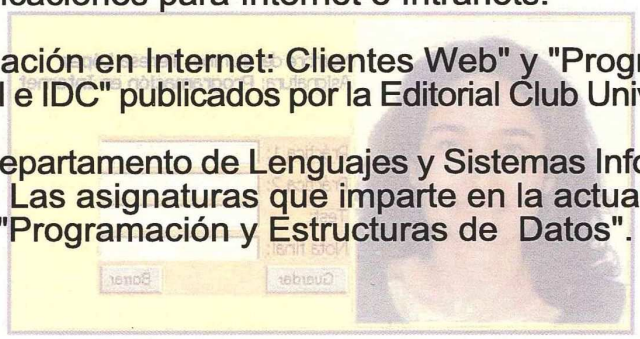
La segunda parte del libro se centra en la programación de la parte cliente de las aplicaciones web. En el "mundo Internet" existen muchas tecnologías que se pueden emplear para programar los clientes web, como ActiveX, applet, Flash, VRML, etc., pero sólo dos son las tecnología más extendidas y se pueden considerar "el estándar": HTML y JavaScript. Este libro se centra en estas dos tecnologías y presta una especial atención a la creación de formularios, la base para cualquier aplicación web.

Sergio Luján Mora es Ingeniero en Informática por la Universidad de Alicante. Ha impartido diversos cursos sobre Internet y las diversas tecnologías que se emplean en la programación de aplicaciones web (HTML, JavaScript, ASP y Java).

Durante un año ha trabajado en el Laboratorio Multimedia (mmlab) de la Universidad de Alicante, desarrollando aplicaciones para Internet e Intranets.

Ha escrito los libros "Programación en Internet: Clientes Web" y "Programación de servidores web con CGI, SSI e IDC" publicados por la Editorial Club Universitario.

Desde 1999 forma parte del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Alicante. Las asignaturas que imparte en la actualidad son "Programación en Internet" y "Programación y Estructuras de Datos".



ECU®
EDITORIAL CLUB UNIVERSITARIO
www.ecu.fm

I.S.B.N.: 84-8454-206-8
9 788484 542063

NOTA DEL AUTOR

Creative Commons

Creative Commons License Deed

Reconocimiento-NoComercial-SinObraDerivada 3.0 España (CC BY-NC-ND 3.0)

Usted es libre de:

copiar, distribuir y comunicar públicamente la obra

Bajo las condiciones siguientes:

Reconocimiento — Debe reconocer los créditos de la obra de la manera especificada por el autor o el licenciador (pero no de una manera que sugiera que tiene su apoyo o apoyan el uso que hace de su obra).

No comercial — No puede utilizar esta obra para fines comerciales.

Sin obras derivadas — No se puede alterar, transformar o generar una obra derivada a partir de esta obra.

Entendiendo que:

Renuncia — Alguna de estas condiciones puede no aplicarse si se obtiene el permiso del titular de los derechos de autor

Dominio Público — Cuando la obra o alguno de sus elementos se halle en el dominio público según la ley vigente aplicable, esta situación no quedará afectada por la licencia.

Otros derechos — Los derechos siguientes no quedan afectados por la licencia de ninguna manera:

- Los derechos derivados de usos legítimosu otras limitaciones reconocidas por ley no se ven afectados por lo anterior.
- Los derechos morales del auto;
- Derechos que pueden ostentar otras personas sobre la propia obra o su uso, como por ejemplo derechos de imagen o de privacidad.

- **Aviso** — Al reutilizar o distribuir la obra, tiene que dejar bien claro los términos de la licencia de esta obra.

Título: Programación de aplicaciones web: historia, principios básicos y clientes web.

Autor: © Sergio Luján Mora

I.S.B.N.: 84-8454-206-8
Depósito legal: A-883-2002

Edita: Editorial Club Universitario Telf.: 96 567 38 45
C/. Cottolengo, 25 - San Vicente (Alicante)
www.ecu.fm
ecu@ecu.fm

Printed in Spain
Imprime: Imprenta Gamma Telf.: 965 67 19 87
C/. Cottolengo, 25 - San Vicente (Alicante)
www.gamma.fm
gamma@gamma.fm

Programación de aplicaciones web: historia, principios básicos y clientes web

Sergio Luján Mora

Prefacio

Internet y la Web han influido enormemente tanto en el mundo de la informática como en la sociedad en general. Si nos centramos en la Web, en poco menos de 10 años ha transformado los sistemas informáticos: ha roto las barreras físicas (debido a la distancia), económicas y lógicas (debido al empleo de distintos sistemas operativos, protocolos, etc.) y ha abierto todo un abanico de nuevas posibilidades. Una de las áreas que más expansión está teniendo en la Web en los últimos años son las aplicaciones web.

Las aplicaciones web permiten la generación automática de contenido, la creación de páginas personalizadas según el perfil del usuario o el desarrollo del comercio electrónico. Además, una aplicación web permite interactuar con los sistemas informáticos de gestión de una empresa, como puede ser gestión de clientes, contabilidad o inventario, a través de una página web.

Las aplicaciones web se encuadran dentro de las arquitecturas cliente/servidor: un ordenador solicita servicios (el cliente) y otro está a la espera de recibir solicitudes y las responde (el servidor). En este libro se aborda la programación de la parte cliente de las aplicaciones web. Existen multitud de tecnologías que se pueden emplear para programar las aplicaciones web, como ActiveX o *applets*, pero no están tan estandarizadas como las dos que se muestran en este libro: HTML y JavaScript.

Este libro se complementa con otro de próxima aparición que tratará la programación de aplicaciones web desde el punto de vista del servidor. En él se mostrarán las tecnologías que se emplean para programar los servidores web: CGI, ASP, JSP, PHP, etc.

Este libro posee diez capítulos y tres apéndices que definen tres partes. La primera parte, formada por los cinco primeros capítulos, aborda una serie de temas "teóricos" como son la historia de Internet y la Web, las arquitecturas cliente/servidor en general, las aplicaciones web como caso particular de las arquitecturas cliente/servidor y la estructura (física y lógica) de un sitio web.

En la segunda parte, formada por los cinco últimos capítulos, se tratan una serie de temas más "prácticos": el lenguaje HTML, una guía de estilo con consejos que ayudan a evitar los errores más comunes a la hora de crear páginas web, los lenguajes de *script* en general y un lenguaje de *script* concreto: JavaScript. Esta segunda parte

finaliza con el modelo de objetos de documento, que permite acceder a los elementos de una página web desde un lenguaje de *script*.

La última parte del libro está formada por tres apéndices donde se resumen las etiquetas de HTML y se explica cómo trabajar con los colores en HTML y cómo depurar errores de JavaScript.

El libro además posee una serie de índices (general, de figuras, de acrónimos, etc.) que facilitan la lectura y la búsqueda de información.

Para finalizar, quisiera mandar un abrazo a mi familia y a Marisa, la gente que quiero y que me apoya en mi trabajo. También me gustaría agradecer a mi compañero Jaume Aragonés del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Alicante la labor que ha realizado de revisión de este libro y sus valiosos comentarios que me han permitido mejorar algunos aspectos confusos.

Alicante, 31 de octubre de 2002 *Sergio Luján Mora*

Índice resumido

Índice general

Índice de cuadros

Índice de figuras

Índice de acrónimos

ADSL *Asymmetric Digital Subscriber Line*
Tecnología de comunicación que permite obtener altas velocidades de transmisión a través de las líneas telefónicas tradicionales. La comunicación es asimétrica porque las velocidades de recepción (128 Kbps hasta 9 Mbps) son mayores que las de transmisión (16 Kbps hasta 640 Kbps).

API *Application Program Interface*
Interfaz de programación de aplicaciones. Conjunto de constantes, funciones y protocolos que permiten programar aplicaciones. Una buena **API** facilita la tarea de desarrollar aplicaciones, ya que facilita todas las piezas y el programador sólo tiene que unirlas para lograr el fin que desea.

ARPA *Advanced Research Projects Agency*
Agencia de Proyectos de Investigación Avanzados. Agencia creada por el Departamento de Defensa de los Estados Unidos de Norteamérica en 1958. También conocida como **DARPA**. A lo largo de los años ha cambiado su nombre varias veces: en 1971 **DARPA**, en 1993 **ARPA** y en 1996 **DARPA** otra vez. El proyecto más conocido de los desarrollados por esta agencia es ARPANET (o ARPAnet), semilla de la actual Internet.

ASP *Active Server Pages*
Tecnología propietaria de MICROSOFT que permite crear páginas web dinámicas en el servidor. Desarrollada con el objetivo de sustituir a la tecnología **CGI**, ofrece una serie de características que facilitan la programación de aplicaciones web. Las páginas **ASP** suelen estar programadas en *VBScript*, aunque también se pueden programar en otros lenguajes, como *JScript*.

ASCII *American Standard Code for Information Interchange*
Código binario utilizado para representar letras, números, símbolos, etc. A cada carácter se le asigna un número del 0 al 127 (7 bits). Por ejemplo, el código **ASCII** para la A mayúscula es 65. Existen códigos **ASCII** extendidos de 256 caracteres (8 bits), que permiten representar caracteres no ingleses como las vocales acentuadas o la eñe. Los caracteres de la parte superior (128 a 255)

de estos códigos **ASCII** extendidos varían de uno a otro. Por ejemplo, uno de los más extendidos es ISO Latin-1 (oficialmente ISO-8859-1).

BMP *Bit-map*
Formato gráfico de mapa de bits estándar en los sistemas operativos Microsoft Windows. Almacena las imágenes en un formato llamado "mapa de bits independiente del dispositivo", que significa que el color de cada punto (*pixel*) se almacena de un modo independiente del método empleado por un dispositivo para representar el color. Existen diversos formatos: 1 bit (blanco y negro), 4 bits (16 colores), 8 bits (256 colores) y 24 bits (16 777 216 colores).

CERN *Conseil Européenne pour le Recherche Nucléaire*
Organización Europea para la Investigación Nuclear. Es el mayor centro científico a nivel mundial dedicado a la física de partículas. Su sede central se encuentra en Ginebra, Suiza. Fundado en 1954 por 12 países, actualmente está formado por 20 países, entre ellos España. Tim Berners-Lee, mientras trabajaba en él a principios de 1990, inventó la **WWW**.

CGI *Common Gateway Interface*
Estándar que permite el intercambio de información entre un servidor y un programa externo al servidor. Un programa **CGI** es un programa preparado para recibir y enviar datos desde y hacia un servidor web según este estándar. Normalmente se programan en *C* o en *Perl*, aunque se puede usar cualquier lenguaje de propósito general.

CSP *Caché Server Pages*
Tecnología propietaria de InterSystems que permite crear páginas web dinámicas en el servidor. Se diferencia de otras tecnologías similares como **ASP** y **JSP** en que la lógica de negocio reside junto con la lógica de datos en el sistema gestor de bases de datos.

CSS *Cascading Style Sheets*
Tecnología empleada en la creación de páginas web, que permite un mayor control sobre el lenguaje **HTML**. Permite crear hojas de estilo que definen como cada elemento, como por ejemplo los encabezados o los enlaces, se tiene que mostrar. El término "en cascada" indica que diferentes hojas de estilo se pueden aplicar sobre la misma página. **CSS** ha sido desarrollada por **W3C**.

DARPA *Defense Advanced Research Projects Agency*
Véase **ARPA**.

DHTML *Dynamic HTML*
Conjunto de extensiones a **HTML** que permiten modificar el contenido de una página web en el cliente sin necesidad de establecer una nueva comunicación con el servidor. Se basa en el uso de **DOM** para acceder al contenido de la página.

DLL *Dynamic Link Library*
Fichero que almacena funciones ejecutables o datos que pueden ser usados por una aplicación en Microsoft Windows. Una **DLL** puede ser usada por varios programas a la vez y se carga en tiempo de ejecución (no en tiempo de compilación).

DOM *Document Object Model*
Especificación que define como se puede acceder a los objetos de un documento **HTML** (ventanas, imágenes, formularios) a través de un lenguaje de *script*. Básicamente define una jerarquía de objetos. **DOM** se encuentra en proceso de estandarización por **W3C**. **DHTML** depende de **DOM** para cambiar dinámicamente el contenido de una página web. Desgraciadamente, los dos navegadores mayoritarios poseen distintos modelos de objetos que en algunas partes son incompatibles entre sí.

ECMA *European Computer Manufacturers Association*
ECMA es una asociación internacional que establece estándares relacionados con sistemas de comunicación y de información.

GIF *Graphics Interchange Format*
Formato gráfico de mapa de bits desarrollado por CompuServe para su servicio de información. Sus principales características son: compresión de datos sin pérdidas (**LZW**), soporte de transparencias y de animaciones. Existen dos versiones de este estándar gráfico: 87A y 89A. Es el formato más adecuado para imágenes con pocos colores, dibujos sencillos o textos.

HTML *HyperText Markup Language*
Lenguaje compuesto de una serie de etiquetas o marcas que permiten definir el contenido y la apariencia de las páginas web. Aunque se basa en el estándar **SGML**, no se puede considerar que sea un subconjunto de él. Existen cientos de etiquetas con diferentes atributos. **W3C** se encarga de su estandarización. El futuro sustituto de **HTML** es **XHTML**.

HTTP *HyperText Transfer Protocol*
Es el protocolo que emplea la **WWW**. Define como se tienen que crear y enviar los mensajes y que acciones debe tomar el servidor y el navegador en respuesta a un comando. Es un protocolo *stateless* (sin estado), porque cada comando se ejecuta independientemente de los anteriores o de los posteriores. Actualmente, la mayoría de los servidores soportan **HTTP** 1.1 (**RFC** 2616 de junio de 1999). Una de las principales ventajas de esta versión es que soporta conexiones persistentes: una vez que el navegador se conecta al servidor, puede recibir múltiples ficheros a través de la misma conexión, lo que aumenta el rendimiento de la transmisión hasta en un 20 %.

IAP *Internet Access Provider*
Véase **ISP**.

IDC *Internet Database Connector*
Conector de bases de datos de Internet. Tecnología propietaria de MICROSOFT que permite generar páginas web dinámicas a partir de la información almacenada en una base de datos. Es el precursor de **ASP**.

IMP *Interface Message Processor*
Máquina encargada del intercambio de paquetes en ARPANET. Sus tareas son: conectar los nodos entre sí, encaminar los mensajes, verificar los mensajes y confirmar la llegada de los mensajes.

ISAPI *Internet Server Application Program Interface*
Un API para el servidor Microsoft Internet Information Server. Permite crear filtros **ISAPI**, programas que se ejecutan en el servidor web en respuesta a determinadas peticiones de los clientes, lo cual facilita la programación de aplicaciones web. Por ejemplo, la tecnología **ASP** es un filtro **ISAPI**.

ISP *Internet Service Provider*
Proveedor de servicios de Internet. Una empresa que proporciona a particulares o empresas acceso a Internet. Para ello, la empresa proporciona un nombre de usuario, una contraseña y un número de teléfono. También se conoce como **IAP**.

ISO *International Organization for Standards*
Organización fundada en 1946, cuyos miembros son las organizaciones nacionales de normalización (estandarización) correspondientes a los países miembros. Entre sus miembros se incluyen ANSI (Estados Unidos), BSI (Gran Bretaña), AFNOR (Francia), DIN (Alemania) y UNE (España).

JPEG *Joint Photographic Experts Group*
Nombre del comité de expertos que desarrolló el formato gráfico con el mismo nombre. Se trata de un formato gráfico de mapa de bits que incorpora compresión de datos con pérdidas y permite trabajar con 24 bits de color (color real o verdadero). El nivel de compresión es variable, por lo que se puede elegir entre mejor calidad y menor compresión o peor calidad y mayor compresión. Este formato se suele emplear con imágenes fotográficas o complejas, pero no es el adecuado para imágenes sencillas, dibujos o textos.

JSP *Java Server Pages*
Tecnología de SUN MICROSYSTEMS que permite crear páginas web dinámicas en el servidor. Equivale a la tecnología **ASP** de MICROSOFT. Se programan en *Java*.

LZW *Lempel Ziv Welch*
Esquema de compresión sin pérdidas empleado en el formato gráfico **GIF** de CompuServe. Desarrollado por J. Ziv and A. Lempel en 1977, y posteriormente mejorado por T. Welch. La patente de **LZW** la ostenta Unisys Corporation. Durante muchos años, Unisys Corporation permitió el uso de **LZW** sin cobrar un canon (la mayoría de la gente no sabía que había sido patentado en 1983). Sin embargo, a partir de 1995 decidió cobrar una tasa y se desató una gran controversia ya que se había extendido ampliamente su uso.

MIME *Multipurpose Internet Mail Extensions*
Se usa en el correo electrónico desde 1992 para enviar y recibir ficheros de distinto tipo. Se puede consultar el estándar en **RFC** 1341, **RFC** 1521 y **RFC** 1522.

MIT *Massachusetts Institute of Technology*
Instituto Tecnológico de Massachusetts. Centro de investigación avanzado situado en los Estados Unidos. Famoso por su "Media Lab", en el cual trabajan investigadores de la talla de Nicholas Negroponte o Marvin Minsky.

MNG *Multiple-image Network Graphics*
Formato gráfico basado en **PNG** que permite usar múltiples imágenes en un fichero, animaciones (como **GIF**) y **JPEG** transparente. Aunque la especificación 1.0 se publicó el 31 de enero de 2001, se encuentra poco extendido y hay pocas herramientas que lo soporten.

NCP *Network Control Protocol*
Protocolo de Control de Red. Primer protocolo *host-to-host* empleado en ARPANET a partir de diciembre de 1970.

NCSA *National Center for Supercomputing Applications*
Centro Nacional para Aplicaciones de Supercomputación. Centro creado en la Universidad de Illinois en enero de 1986. Famoso porque uno de los primeros navegadores web gratuitos, NCSA Mosaic, se creó en sus instalaciones.

NPL *National Physical Laboratory*
Laboratorio Nacional de Física. Centro de investigación del Reino Unido, famoso porque en él se trabajó en la conmutación de paquetes y las redes de área amplia, de forma paralela e independiente al trabajo desarrollado en Estados Unidos. En este centro se acuñaron los términos "paquete" y "conmutación de paquetes".

ODBC *Open DataBase Connectivity*
Conectividad abierta de bases de datos. **ODBC** es un estándar *de facto* para el acceso a bases de datos en entornos cliente/servidor. El objetivo de **ODBC**

es facilitar el acceso a cualquier dato desde cualquier aplicación, independientemente del sistema gestor de bases de datos empleado. Para ello, en **ODBC**se inserta una capa intermedia, llamada controlador (*driver*) de la base de datos, entre la aplicación y el sistema gestor de bases de datos. El propósito de esta capa es traducir las consultas que genera la aplicación en comandos que entienda el sistema gestor de bases de datos. Por tanto, mediante **ODBC**, se puede cambiar la parte servidor (la base de datos) sin tener que cambiar el cliente, siempre que todas las partes sean compatibles con **ODBC**.

OSI *Open System Interconnection*
También conocido como el Modelo de Referencia **OSI** o el Modelo **OSI**. Se trata de un estándar de **ISO** que define un marco para implementar los protocolos de red en siete capas. Los siete niveles, desde el más inferior (1) al superior (7) son: físico, enlace, red, transporte, sesión, presentación y aplicación.

PNG *Portable Network Graphics*
Formato gráfico de mapa de bits similar a **GIF**. **W3C** ha desarrollado este formato gráfico con la idea de sustituir **GIF** por **PNG** debido a que el primero emplea un algoritmo que está patentado, mientras que **PNG** es totalmente gratuito. No permite crear animaciones, pero sí que permite definir distintos niveles de transparencia. Al igual que **GIF**, emplea un esquema de compresión sin perdidas que logra tasas de compresión mayores que **GIF**. Tanto Microsoft Internet Explorer como Netscape Communicator aceptan este formato, aunque no todas sus características.

RFC *Request for Comments*
Medio de publicar propuestas sobre Internet. Cada **RFC** recibe un número. Algunos se convierten en un estándar de Internet.

RGB *Red Green Blue*
Notación de los colores en la que cada color se representa como una combinación de los tres colores básicos (primarios) rojo (*red*), verde (*green*) y azul (*blue*). Se trata de un modelo aditivo (se parte del negro). Mediante la combinación adecuada de los tres colores básicos se consigue todo el espectro de colores. Además de **RGB** existen otras formas de representar los colores. Otra de las más corrientes es CMYK (*cyan*, *magenta*, *yellow*, *black*), que se trata de un modelo sustractivo.

RPC *Remote Procedure Call*
Llamada a procedimiento remoto. Protocolo que permite a un programa en un ordenador (cliente) ejecutar un programa en otro ordenador (servidor). El programa en el cliente envía un mensaje al servidor con los argumentos necesarios y el servidor devuelve un mensaje que contiene los resultados obtenidos al ejecutar el programa con los argumentos recibidos.

SGML *Standard Generalized Markup Language*
Lenguaje que permite organizar y etiquetar los distintos elementos que componen un documento. Se emplea para manejar grandes documentos que sufren constantes revisiones y se imprimen en distintos formatos e idiomas. Desarrollado y estandarizado por **ISO** en 1986.

SLAC *Stanford Linear Accelerator Center*
Centro de investigación nuclear en California (EE.UU.). En diciembre de 1992, fue el primer sitio fuera de Europa en instalar un servidor web.

SSL *Secure Socket Layer*
Protocolo diseñado por NETSCAPE que permite transmisiones seguras de información a través de Internet.

TCP/IP *Transmission Control Protocol/Internet Protocol*
Familia de protocolos que se emplean en las comunicaciones de Internet.

URL *Universal Resource Locator*
También conocido como *Uniform Resource Locator*. Sistema de direccionamiento de máquinas y recursos en Internet. Es decir, se trata de una dirección que permite localizar cualquier máquina o documento que se encuentre accesible a través de Internet.

VPN *Virtual Private Network*
Red Privada Virtual. Red privada de comunicaciones que se basa en el empleo de una red pública. Emplea protocolos de comunicación seguros para establecer canales de comunicación seguros sobre una red pública que es insegura.

VRML *Virtual Reality Modeling Language*
Lenguaje de Modelado de Realidad Virtual. Lenguaje para crear objetos en tres dimensiones en la Web. Los ficheros creados con este lenguaje poseen la extensión `.wrl` (de *world*) y para visualizarlos es necesario emplear un visor adecuado o que el navegador web disponga del correspondiente *plug-in*.

W3C *World Wide Web Consortium*
Consorcio internacional de compañías y organizaciones involucradas en el desarrollo de Internet y en especial de la **WWW**. Su propósito es desarrollar estándares y "poner orden" en Internet.

WAI *Web Accessibility Initiative*
Comité de **W3C** creado con el objetivo de aumentar la usabilidad de la Web de cara a la gente con minusvalías.

WWW *World Wide Web*
También conocida como "la Web" o "la Red". Sistema mundial de servidores web conectados a Internet (no todos los ordenadores conectados a Internet forman

parte de la **WWW**). Su protocolo de comunicación es **HTTP**, su lenguaje de creación de documentos **HTML** y su sistema de direccionamiento de los recursos **URL**. Los navegadores web (*browsers*) permiten navegar por la web.

WYSIWYG *What You See Is What You Get*
Una aplicación es **WYSIWYG** cuando en pantalla se puede visualizar exactamente como se verá un documento cuando se imprima.

XHTML *Extensible HyperText Markup Language*
HTML escrito según las normas que marca **XML**. Por tanto, se trata de una aplicación concreta de **XML** y no tienen que confundirse entre sí.

XML *Extensible Markup Language*
Metalenguaje de etiquetado basado en **SGML**. Diseñado específicamente para la **WWW** por **W3C**. Permite que un usuario diseñe sus propias etiquetas, con sus atributos y las reglas de construcción de documentos (sintaxis).

Capítulo 1

Introducción

En este capítulo se realiza una introducción del libro y se presenta el contenido de cada uno de los capítulos. Además, también se comentan las convenciones tipográficas empleadas para distinguir los acrónimos, nombres de programas, etc.

Índice General

1.1. Introducción

Este libro contempla la programación de la parte cliente de las aplicaciones web. En el "mundo Internet" existen muchas tecnologías que se pueden emplear para programar los clientes web, pero sólo dos son las más extendidas y se pueden considerar "el estándar *de facto*": *HyperText Markup Language* (**HTML**) y *JavaScript*.

Este libro no trata sobre diseño gráfico o artístico de sitios web. Sin embargo, sí que se ofrecen algunos consejos que pueden ayudar a lograr sitios web más fáciles de usar, prácticos y elegantes. Por ello, me gustaría incluir unas palabras de Miguel Ripoll, uno de los diseñadores web de más prestigio mundial:

> A good website has to be fast to download, easy to navigate, appealing to the eye, on brand and on target, offer something different from the rest, have added value, and constitute a unique user experience. Simple, really.

> *Un buen sitio web tiene que ser rápido de descargar, fácil de navegar, atractivo a la vista, centrado en la marca y en el objetivo, ofrecer algo diferente del resto, tener un valor añadido y constituir una experiencia única para el usuario. Simple, en realidad.*
>
> Miguel Ripoll, `http://www.miguelripoll.com/`

En definitiva, se puede decir que las dos reglas básicas que hay que tener en cuenta para desarrollar un sitio web correcto son la simplicidad en el diseño visual y la eficacia de las herramientas de navegación y de búsqueda que incorpore.

1.2. Contenido de los capítulos

Este libro se compone de 10 capítulos y 3 apéndices, además de varios índices (figuras, acrónimos, etc.) que facilitan la búsqueda de información.

El el Capítulo 2 (**Historia de Internet**) se presenta una breve historia de Internet. Se comentan los hechos más importantes y se destacan las personas que más han influido en su desarrollo. Además, se dedica un apartado especial a la historia de la Web y se presentan las cuatro generaciones de sitios web que existen actualmente.

En el Capítulo 3 (**Arquitecturas cliente/servidor**), como las aplicaciones web son un tipo especial de aplicaciones cliente/servidor, se introducen las características básicas de las arquitecturas cliente/servidor.

En el Capítulo 4 (**Qué es una aplicación web**) se comentan los tres niveles típicos de las aplicaciones web: el nivel superior que interacciona con el usuario (el cliente web, normalmente un navegador), el nivel inferior que proporciona los datos (la base de datos) y el nivel intermedio que procesa los datos (el servidor web). En este capítulo se describen el cliente y el servidor web y se comentan los entornos web en los que se ejecutan las aplicaciones web: Internet, intranet y extranet. Además se comentan las principales ventajas que poseen las aplicaciones web. Por último se presenta una metodología de desarrollo de sitios web.

En el Capítulo 5 (**Estructura de un sitio web**) se introducen los conceptos de sitio web, estructura física y estructura lógica (o de navegación). Además, se incluye una guía de estilo con consejos a tener en cuenta cuando se diseñe la navegación de un sitio web. El principal objetivo que se quiere lograr cuando se diseñan la estructura física y la estructura lógica es lograr sitios web que sean fáciles de mantener y de navegar.

En el Capítulo 6 (**HTML**) se presenta el lenguaje de marcas (etiquetas) que se emplea para dar formato a los documentos que se quieren publicar en la *World Wide Web* (**WWW**). Los navegadores pueden interpretar las etiquetas y muestran los documentos con el formato deseado. En este capítulo se presentan los conceptos básicos y avanzados (enlaces, tablas, marcos, etc.) de **HTML**. Además, se realiza un estudio especial de los formularios, ya que son una pieza clave de las aplicaciones web.

En el Capítulo 7 (**Guía de estilo**) se presentan una serie de consejos que ayudan a no cometer los errores más comunes a la hora de crear un sitio web. Además, también se incluyen una serie de consejos sobre accesibilidad, tanto de cara al usuario como al navegador.

En el Capítulo 8 (**Lenguajes de script**) se tratan los lenguajes de script, que permiten incluir "programación" en las páginas web. En este capítulo se explican las tres formas que existen de incluir y ejecutar código en una página web.

En el Capítulo 9 (**JavaScript**) se presenta un lenguaje de script concreto: *JavaScript*. Este lenguaje es el más empleado en Internet y se puede considerar el lenguaje estándar. En este capítulo se estudian sus características básicas, sus diferentes sentencias, las funciones que incorpora y, por último, como validar formularios.

En el Capítulo 10 (**Modelo de objetos de documento**) se trata el Modelo de Objetos de Documento, que es un interfaz que permite acceder y modificar la estructura y contenido de una página web. Este modelo especifica como se puede acceder a los distintos elementos (enlaces, imágenes, formularios, etc.) de una página y como se pueden modificar. En este capítulo se describen los objetos que componen este modelo, sus propiedades, métodos y eventos.

Además, el libro también posee una serie de apéndices que complementan la información tratada a lo largo de los capítulos. En concreto, en el Apéndice A (**Resumen de etiquetas HTML**) se incluye un resumen de la sintaxis de las etiquetas **HTML** que acepta Netscape Communicator 4.0 y posteriores. El objetivo de este apéndice es que sirva como una guía rápida de búsqueda en caso de duda.

En el Apéndice B (**Colores en HTML**) se explica como trabajar con los colores en **HTML**. Además, también se incluyen una serie de consejos sobre el uso de los colores en las páginas web.

Finalmente, en el Apéndice C (**Depuración de errores de JavaScript**)se explica como se puede depurar el código *JavaScript* en cualquier navegador. Además, se comentan algunas herramientas específicas que poseen los navegadores Microsoft Internet Explorer y Netscape Communicator.

1.3. Convenciones tipográficas

Con el fin de mejorar la legibilidad del texto, distintas convenciones tipográficas se han empleado a lo largo de todo el libro.

Los ejemplos, que normalmente están completos y por tanto se pueden escribir y probar, aparecen destacados y numerados dentro de una caja de la siguiente forma (el texto de los ejemplos emplea un `tipo de letra de paso fijo como Courier`):

Ejemplo 1.1

```
1 <!DOCTYPE HTML PUBLIC ".//W3C//DTD HTML 4.0//EN">
2 <HTML>
3 <HEAD>
```

```
Cabecera de la página
</HEAD>
<BODY>
Cuerpo de la página
</BODY>
</HTML>
```

Los estilos empleados a lo largo del texto son:

- Los acrónimos y siglas que figuran en el índice de acrónimos aparecen siempre destacados en **negrita**. Ejemplo: **ASP**, **HTML**, **URL**, etc.
- Cuando un acrónimo aparece por primera vez, se muestra el nombre completo en *cursiva* y entre paréntesis y en **negrita** el acrónimo. Ejemplo: *Graphics Interchange Format* (**GIF**), *World Wide Web* (**WWW**), etc.
- Las palabras no escritas en español aparecen destacadas en *cursiva*. Ejemplo: *bookmarks*, *browser*, *plug-in*, etc.
- Cuando se hace referencia a un programa, el texto de los interfaces que se ven en pantalla aparece en **negrita**. Además, se emplea una flecha para indicar una secuencia de acciones o pulsaciones en un programa. Ejemplo: **Aceptar**, **Inicio** → **Programas** → **Accesorios**.
- Los nombres de las compañías se muestran con un tipo de letra de MAYÚSCULAS PEQUEÑAS. Ejemplo: MICROSOFT, NETSCAPE, etc.
- Los nombres de los programas se muestran con un tipo de letra sin palo (sans serif). Ejemplo: Microsoft Paint, Netscape Navigator, Opera, etc.
- Los lenguajes informáticos se muestran con un tipo de letra *inclinada*. Ejemplo: *C*, *Java*, *Perl*, etc.
- Las extensiones de los ficheros, las palabras clave de los lenguajes de programación y el código incluido dentro del texto se muestra con un `tipo de letra de paso fijo como Courier`. Ejemplo: `.html`, `<IMG>`, `var ciudad = "Elche"`, etc.

Capítulo 2

Historia de Internet

En este capítulo se presenta una breve historia de Internet. Se comentan los hechos más importantes y se destacan las personas que más han influido en su desarrollo. Además, se dedica un apartado especial a la historia de la Web y se presentan las cuatro generaciones de sitios web que existen actualmente.

Índice General

2.1. Historia de Internet

El desarrollo de Internet, como casi todos los avances de la ciencia y la tecnología, no se debe a una persona o a un grupo pequeño de personas, sino que ha sido fruto de las ideas y del trabajo de miles de personas. Sin embargo, en un repaso de la

historia de Internet de unas pocas páginas sólo se pueden nombrar a las personas más importantes.

Como reconocimiento al cambio que Internet ha producido en todos los niveles de la sociedad, el 23 de mayo de 2002[1], Lawrence Roberts, Robert Kahn, Vinton Cerf y Tim Berners-Lee fueron distinguidos con el Premio Príncipe de Asturias de Investigación Científica y Técnica en representación de las "miles de personas y muchas instituciones" que han hecho posible este avance de nuestro tiempo. Según la resolución del jurado, "Se les otorga el premio por haber diseñado y realizado un sistema que está cambiando el mundo al ofrecer posibilidades antes impensables para el progreso científico y social".

A Lawrence (Larry) Roberts se le suele llamar "el padre de Internet", porque fue el director del equipo de ingenieros que crearon ARPANET, el precursor de la actual Internet. A parte de ser el director, también fue el diseñador principal de ARPANET.

En 1972, Robert Kahn fue contratado por Lawrence Roberts para trabajar en ARPA. Trabajó en el desarrollo de un modelo de arquitectura de red abierta, donde cualquier ordenador pudiera comunicarse con cualquier otro, independientemente del hardware o el software particular de cada uno de ellos. Este trabajo le llevó a desarrollar, junto con Vinton Cerf, el protocolo *Transmission Control Protocol/Internet Protocol* (**TCP/IP**).

Vinton Cerf estuvo implicado desde los primeros años en el desarrollo de ARPANET. En 1973, se unió al proyecto de Robert Kahn de interconexión de redes. Su mayor contribución ha sido el desarrollo, junto a Robert Kahn, de **TCP/IP**, el protocolo que gobierna las comunicaciones en Internet y que permite conectar distintas redes independientes entre sí.

Finalmente, Tim Berners-Lee es conocido como "el padre de la Web". Él fue quien creo **HTML**, el lenguaje empleado para crear los documentos de la Web; *HyperText Transfer Protocol* (**HTTP**), el protocolo que emplean los ordenadores para comunicarse en la Web, y *Universal Resource Locator* (**URL**), como medio de localización de los distintos recursos que forman la Web en Internet. Además, también desarrolló el primer servidor web y el primer navegador/editor web.

2.1.1. Hitos en la diseminación de la información

A lo largo de la historia, los medios de comunicación han evolucionado considerablemente. Se suele citar la capacidad de transmitir el conocimiento de una generación a otra como una de las características que distingue a la especie humana de otras especies animales, ya que es la clave en el avance de la humanidad.

Distintos acontecimientos han marcado hitos en la comunicación. Si nos centramos en los más importantes que se han producido desde el siglo XIX, destacan por orden cronológico:

[1] El premio les fue entregado por el Príncipe de Asturias el 25 de octubre de 2002 en la ceremonia que se celebró en Oviedo.

- 1833: distribución en masa de periódicos.
- 1844: primer mensaje telegrafiado.
- 1858: primer intento de tender un cable de comunicaciones a través del Océano Atlántico. Deja de funcionar a los pocos días debido a problemas con el aislamiento del cable.
- 1866: se tiende con éxito un cable de comunicaciones a través del Océano Atlántico.
- 1876: invención del teléfono por Alexander Graham Bell.
- 1901: primera señal de radio enviada a través del Océano Atlántico.
- 1917: primera llamada de teléfono transcontinental.
- 1927: primera película hablada.
- 1939: debut de la televisión en la Feria Mundial celebrada en Nueva York.
- 1950: inicio de las retransmisiones de televisión en color.
- 1958: los laboratorios Bell inventan el módem.
- 1969: se crea ARPANET, el primer "Internet". Es el primer medio de comunicación que engloba todos los medios existentes: permite comunicación escrita, sonora y de vídeo.

Actualmente, Internet es un medio de comunicación que incluye a todos los demás. A través de Internet se puede leer el periódico, se pueden realizar llamadas de teléfono, se puede ver la televisión o escuchar la radio, etc.

2.1.2. El primer "Internet"

Las bases del actual Internet se crearon en la década de los sesenta. De forma paralela, y sin que hubiera conocimiento entre ellos del trabajo de los otros, en tres centros de investigación se desarrollaron estudios sobre la comunicación de ordenadores, la redes distribuidas y la conmutación de paquetes: el *Massachusetts Institute of Technology* (**MIT**) entre 1961 y 1967, The RAND Corporation entre 1962 y 1965, y el *National Physical Laboratory* (**NPL**) entre 1964 y 1967.

El origen de Internet se sitúa en plena Guerra Fría. En 1957, la extinta Unión de Repúblicas Socialistas Soviéticas (URSS) lanzó al espacio el primer satélite: el Sputnik[2]. La URSS estaba ganando la partida a los Estados Unidos de Norteamérica

[2]La colocación permanente en el espacio de un ingenio humano suponía una gran amenaza, ya que si se podía poner un satélite, entonces también se podía poner un arma nuclear.

(EEUU) en el desarrollo de nuevas tecnologías. Como respuesta a la amenaza que suponía la URSS, los EEUU desarrollaron distintas iniciativas. Una de ellas fue crear en 1958 *Advanced Research Projects Agency* (**ARPA**) por orden del presidente de los Estados Unidos Dwight D. Eisenhower, encargada de desarrollar proyectos de investigación avanzada. En 1962 comenzó el programa de investigación computacional de **ARPA** y en 1966 el programa de comunicaciones bajo la dirección de Lawrence G. Roberts que provenía del **MIT**. Dentro del programa de comunicaciones se desarrolló ARPANET, con el objetivo de explorar la distribución y el uso compartido de recursos informáticos y las comunicaciones basadas en conmutación de paquetes.

Por otro lado, la Fuerza Aérea de los Estados Unidos (*U.S. Air Force*) encargó a la organización The RAND Corporation el estudio de sistemas de comunicaciones digitales basados en sistemas distribuidos. El objetivo era desarrollar una red de comunicaciones militar tolerante a "ataques nucleares". Paul Baran ideó un sistema que no dependía de instalaciones centralizadas y que podía funcionar incluso si muchos de sus enlaces y nodos de comunicación eran destruidos. Todos los nodos poseían la misma condición: eran autónomos y capaces de recibir, dirigir y transmitir la información. En el sistema de comunicación ideado por Paul Baran, cada mensaje se dividía en una serie de pequeños trozos de tamaño establecido, y cada trozo se enviaría de forma individual. Además, cada trozo encontraría su camino hasta la dirección de destino: si partes de la red fueran destruidas, como cada nodo era autosuficiente y cada trozo poseía información sobre el origen y el destino, cada nodo establecería caminos alternativos para transmitir la información.

Es debido al trabajo de Paul Baran el mito de que Internet se creó por los militares para hacer frente a un ataque nuclear. Sin embargo, esto es falso, ya que el proyecto ARPANET no tenía unos fines militares, sino facilitar la comunicación entre los científicos.

Finalmente, en el **NPL** del Reino Unido, Donald Davies, Roger Scantlebury y otros investigadores trabajaron en la conmutación de paquetes a mediados de los sesenta. Sin embargo, no fueron capaces de convencer al gobierno británico de que financiase sus experimentos en el desarrollo de redes de área amplia. No obstante, fueron ellos los que acuñaron los términos "paquete" y "conmutación de paquetes".

Los principales acontecimientos en este desarrollo a tres bandas fueron:

- 1961 (julio): Leonard Kleinrock publica "Information Flow in Large Communication Nets", el primer artículo sobre conmutación de paquetes.
- 1962 (agosto): J.C.R. Licklider escribe una serie de informes sobre su "Galactic Network": un conjunto de ordenadores conectados globalmente a través de los cuales cualquiera puede acceder a datos y programas existentes en cualquiera de ellos.
- 1962 (agosto): J.C.R. Licklider es nombrado director del primer programa de investigación computacional en **ARPA**.

- 1964: Leonard Kleinrock publica "Communication Nets", el primer libro sobre conmutación de paquetes y redes de ordenadores.
- 1964 (agosto): Paul Baran (The RAND Corporation) publica "On Distributed Communications", documento donde recoge todo su trabajo desarrollado sobre comunicaciones distribuidas, conmutación de paquetes, etc.
- 1966 (agosto): Lawrence Roberts deja el **MIT** y se incorpora a **ARPA**.
- 1966 (diciembre): Lawrence Roberts comienza el diseño de ARPANET.
- 1967 (octubre): En la *ACM Operating Systems Symposium* en Gatlinberg (Tennessee) se presentan dos artículos clave en el desarrollo de Internet:
  - "Multiple Computer Networks and Intercomputer Communication" de Lawrence Roberts, donde se presenta el diseño de ARPANET.
  - "A Digital Communications Network for Computers", de Donald Davies, Roger Scantlebury y otros, donde se introducen por primera vez los conceptos de paquete y conmutación de paquetes.

  En este conferencia, Lawrence Roberts y Roger Scantlebury se conocen, intercambian ideas y Scantlebury le habla a Roberts de Paul Baran y su trabajo.
- 1968 (diciembre): La empresa Bolt Beranek and Newman gana la licitación para construir el primer *Interface Message Processor* (**IMP**). Proponen emplear un miniordenador Honeywell DDP-516 con 12K de memoria, una velocidad de 1,1 MHz y un peso de unos 400 Kg (Figura 2.1).
- 1969 (abril): Aparece el primer *Request for Comments* (**RFC**) con el título "Host Software", escrito por Steve Crocker.
- 1969 (septiembre): El 1 de septiembre se instala el primer nodo de ARPANET en la Universidad de California en Los Ángeles (UCLA). Se conecta el **IMP** a un ordenador XDS (Xerox Data Systems)[3] Sigma 7. En la Figura 2.2 se muestra un boceto de este primer nodo realizado por Lawrence Roberts.
- 1969 (octubre): El segundo nodo de ARPANET se instala en el Instituto de Investigación de Stanford (*Stanford Research Institute*, SRI). Se conecta el **IMP** a un ordenador XDS 940. Ese mismo día se transmite el primer mensaje de ARPANET.
- 1969 (noviembre): Se instala el tercer nodo de ARPANET en la Universidad de California en Santa Bárbara (UCSB). El **IMP** se conecta a un IBM 360/75.
- 1969 (diciembre): El cuarto nodo se instala en la Universidad de Utah. Se conecta el **IMP** a un DEC PDP-10. En la Figura 2.3 se puede ver un boceto de Lawrence Roberts con los cuatro primeros nodos de ARPANET.

[3] También conocido como Scientific Data Systems (SDS).

Figura 2.1: Leonard Kleinrock junto al primer IMP

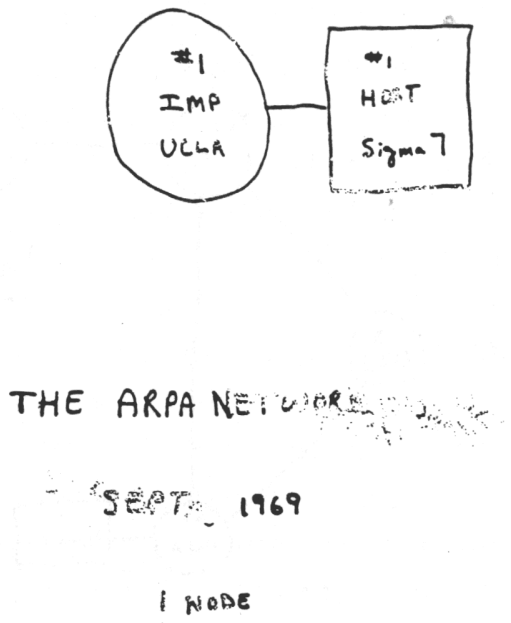


Figura 2.2: El primer nodo de ARPANET

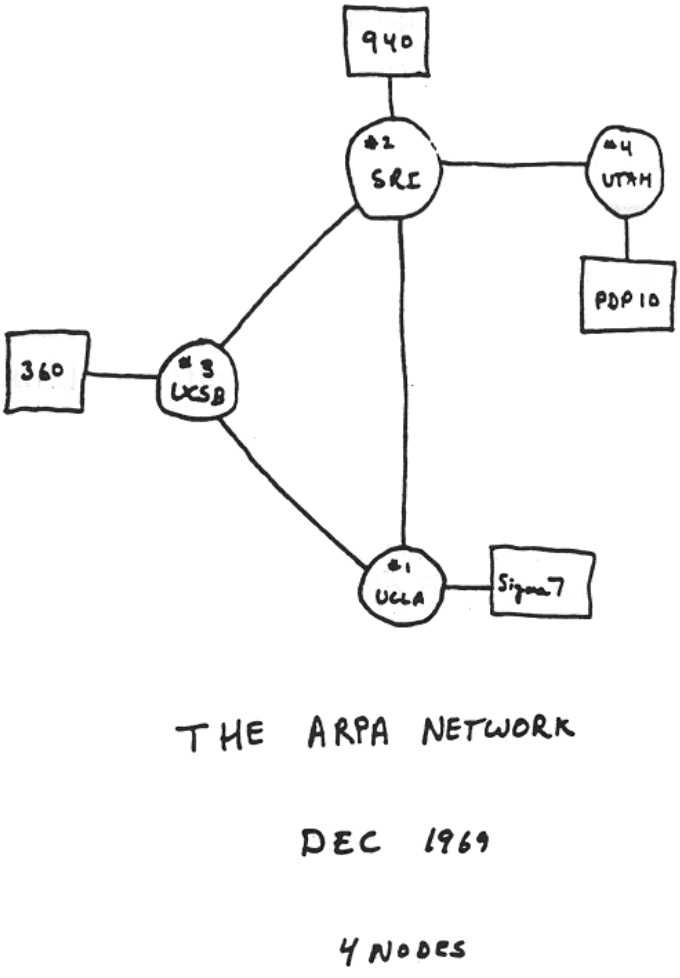


Figura 2.3: Los cuatro primeros nodos de ARPANET

- 1970 (diciembre): Se completa *Network Control Protocol* (**NCP**), el primer protocolo *host-to-host* empleado en ARPANET.

- 1971 (abril): 15 nodos conectados a ARPANET. En la Figura 2.4 se muestra el diseño lógico de ARPANET con los 15 nodos.

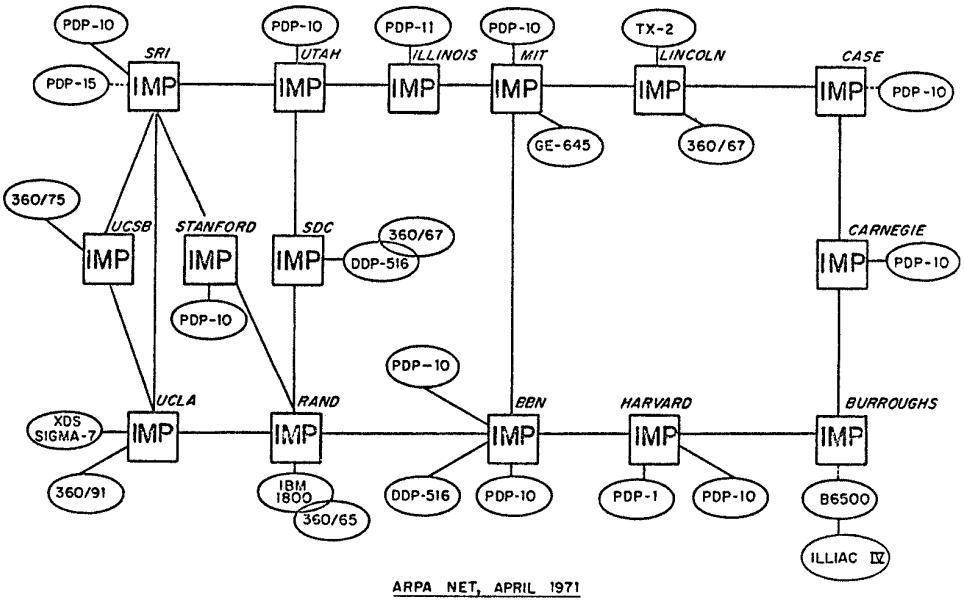


Figura 2.4: Diseño lógico de ARPANET en abril de 1971

- 1972: 37 nodos conectados a ARPANET.

- 1972 (marzo): Ray Tomlinson de Bolt Beranek and Newman crea el primer programa de correo electrónico.

- 1972 (octubre): Primera demostración pública de ARPANET en "International Conference on Computer Communications" en Washington D.C.

- 1973: Primera conexión internacional a ARPANET: *University College of London* (Inglaterra) y *Royal Radar Establishment* (Noruega).

- 1974 (mayo): Robert Kahn y Vinton Cerf publican "A Protocol for Packet Network Interconnection", en *IEEE Transaction on Communications*[4]. En este artículo se presenta el primer protocolo de interconexión de redes (TCP). Además, aparece por primera vez el término *Internet*.
- 1978 (marzo): TCP se separa en **TCP/IP**: TCP se encarga de la comunicación extremo a extremo e IP del proceso de direccionamiento.
- 1983: ARPANET se divide en MILNET (formada 45 nodos de carácter militar) y ARPANET (68 nodos de carácter civil).
- 1983 (enero): A partir del 1 de enero, cualquier máquina conectada a ARPANET debe usar **TCP/IP** (se sustituye **NCP**).
- 1988 (2 de noviembre): El primer gusano ataca Internet.
- 1989: ARPANET se cierra.
- 1990 (noviembre): Se instala el primer servidor web en el *Conseil Européenne pour le Recherche Nucléaire* (**CERN**).

2.1.3. Protocolos de Internet

El éxito de Internet se basa mucho en el empleo de **TCP/IP**, el conjunto de protocolos de comunicación que permiten el intercambio de información de forma independiente de los sistemas en que ésta se encuentra almacenada. **TCP/IP** constituye la solución problema de heterogeneidad de los sistemas informáticos. El 1 de enero de 1983, **TCP/IP** se estableció como el protocolo estándar de comunicación en Internet.

El conjunto de protocolos **TCP/IP**, también llamado la pila de protocolos **TCP/IP**, incluye una serie de protocolos que se encuentran en el nivel 7 o de aplicación de la arquitectura *Open System Interconnection* (**OSI**) y que proporcionan una serie de servicios.

Como un mismo ordenador puede atender varios servicios, cada servicio se identifica con un número llamado puerto. Por tanto, a cada protocolo le corresponde un número de puerto. Los protocolos que se encuentran estandarizados poseen un puerto reservado que no puede emplear ningún otro protocolo.

En el Cuadro 2.1 se muestran los protocolos del nivel 7 más comunes de Internet junto con el número de puerto que emplean.

Además de los anteriores protocolos, existen otros menos conocidos que se encuentran en diferentes niveles de la arquitectura **OSI**, como son: *Address Resolution Protocol* (ARP), *Dynamic Host Configuration Protocol* (DHCP), *Finger*, *Gopher*, *Internet Control Messaging Protocol* (ICMP), *Internet Relay Chat* (IRC), *Network File*

[4] V.G. Cerf y R.E. Kahn. A Protocol for Packet Network Interconnection. *IEEE Transaction on Communications*, 22(5), Mayo 1974, páginas 637-648.

Nombre	Acrónimo	Puerto	Descripción
File Transfer Protocol	FTP	21	Transferencia de ficheros
Telnet		23	Conexión en modo terminal a sistemas remotos
Simple Mail Transfer Protocol	SMTP	25	Envío de correo electrónico
Domain Name System	DNS	53	Resolución de nombres de dominio
HyperText Transfer Protocol	HTTP	80	Transferencia de páginas web
Post Office Protocol v3	POP3	110	Recepción de correo electrónico
Network News Transfer Protocol	NNTP	119	Acceso a foros de discusión

Cuadro 2.1: Protocolos más comunes de Internet

System (NFS), *Network Time Protocol* (NTP), *Routing Information Protocol* (RIP) y *Simple Network Management Protocol* (SNMP).

2.2. Historia de la Web

Al igual que Internet, el desarrollo de la Web no se debe a una única persona. Pero si buscamos un único padre de la Web, ese es Tim Bernes-Lee. A él se deben los tres elementos que fueron clave en el nacimiento de la Web (Figura 2.5):

- **HTML** como lenguaje para crear los contenidos de la Web, basado en *Standard Generalized Markup Language* (**SGML**).
- **HTTP** como protocolo de comunicación entre los ordenadores de la Web, encargado de la transferencia de las páginas web y demás recursos.
- **URL** como medio de localización (direccionamiento) de los distintos recursos en Internet.

Los acontecimientos más importantes en el nacimiento de la Web se remontan a los años cuarenta:

- 1945: Vannevar Bush escribe el artículo "As We May Think" en *The Atlantic Monthly*[5] sobre un dispositivo fotoeléctrico y mecánico, llamado *memex*, capaz

[5]Vannevar Bush. As We May Think. *The Atlantic Monthly.* Volumen 176, nº 1, páginas 101-108, julio 1945.

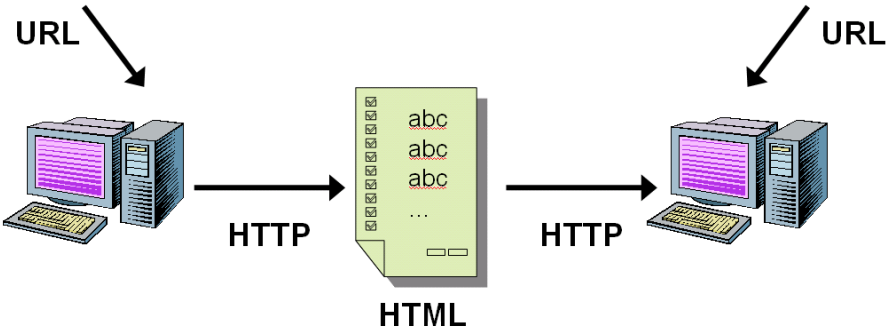


Figura 2.5: Los pilares de la Web

de crear y seguir enlaces entre distintos documentos almacenados en microfichas (en definitiva, un sistema muy parecido a lo que hoy conocemos como hipertexto).

> [...] Consider a future device for individual use, which is a sort of mechanized private file and library. It needs a name, and, to coin on at random, "memex" will do. A memex is a device in which an individual stores all his books, records, and communications, and which is mechanized so that it may be consulted with exceeding speed and flexibility. It is an enlarged intimate supplement to his memory.
>
> [...] All this is conventional, except for the projection forward of present-day mechanisms and gadgetry. It affords an immediate step, however, to associative indexing, the basic idea of which is a provision whereby any item may be caused at will to select immediately and automatically another. This is the essential feature of the memex. The process of tying two items together is the important thing.

> *[...] Considera un dispositivo futuro para uso individual, que es una especie de archivo y biblioteca personal mecanizada. Necesita un nombre, y, acuñando uno al azar, "'memex" servirá. Un memex es un dispositivo en el que un individuo almacena todos sus libros, registros y comunicaciones, y que está mecanizado de forma que se puede consultar con una gran velocidad y flexibilidad. Es un íntimo complemento a su memoria.*
>
> *[...] Todo esto es convencional, excepto por la proyección futura de mecanismos y artilugios actuales. Sin embargo, permite un paso inmediato al indexado asociativo, cuya idea básica es permitir que a*

> *partir de cualquier elemento se pueda seleccionar inmediatamente y automáticamente otro cuando se desee. Esta es la característica fundamental del memex. El proceso de ligar dos elementos juntos es el aspecto importante.*
>
> Vannevar Bush, "As We May Think"

- 1965: Ted Nelson acuña el término "hipertexto" en el artículo "A File Structure for the Complex, the Changing, and the Indeterminate"[6]. Comienza el desarrollo del proyecto Xanadu[7], un sistema basado en hipertexto que nunca llegó a completarse (aún continúa su desarrollo).
- 1967: Andy van Dam y su equipo construyen *Hypertext Editing System* (HES), el primer sistema de hipertexto. Sus principales características son: permite editar grandes cantidades de texto en pantalla, permite teclear cadenas tan largas como el usuario desee y permite enlaces dentro de un documento que conducen a otras partes del mismo documento o a otro documento.
- 1968: Doug Engelbart y su equipo dan a conocer su sistema *On-Line System* (NLS), una herramienta de trabajo en grupo con soporte de enlaces entre documentos.
- 1969: Andy van Dam y su equipo construyen *File Retrieval and Editing System* (FRESS) a partir de su anterior sistema *Hypertext Editing System*. Sus principales características son: permite el empleo de terminales gráficos y, por tanto, el empleo de caracteres no occidentales y cualquier símbolo en pantalla, los enlaces pueden ser bidireccionales y posee la capacidad de "deshacer".
- 1980: mientras trabaja en el **CERN**, Tim Berners-Lee escribe un programa llamado Enquire-Within-Upon-Everything, que permite crear enlaces entre nodos. Un nodo posee un título, un tipo y una lista de enlaces.
- 1989 (marzo): Tim Berners-Lee escribe "Information Management: A Proposal", un informe interno que circula por el **CERN**.
- 1990 (septiembre): Mike Sendall, jefe de Tim Berners-Lee da el visto bueno a la compra del ordenador NeXT, lo que permite a Tim seguir adelante y crear un sistema global de hipertexto.
- 1990 (octubre): Tim Berners-Lee comienza a desarrollar un editor y navegador gráfico de hipertexto para NeXTStep, el sistema operativo con entorno gráfico de los ordenadores NeXT. Elige WorldWideWeb como nombre del programa y "World Wide Web" como nombre del proyecto, después de descartar una serie de nombres: *Information Mesh*, *Mine of Information* e *Information Mine*.

[6] Ted Nelson. A File Structure for the Complex, the Changing, and the Indeterminate. *20th National Conference ACM*, páginas 84-100, New York. Association for Computing Machinery, 1965.

[7] `http://www.xanadu.net/`.

- 1990 (noviembre): se instala el primer servidor web[8] y se publica la primera página web[9].

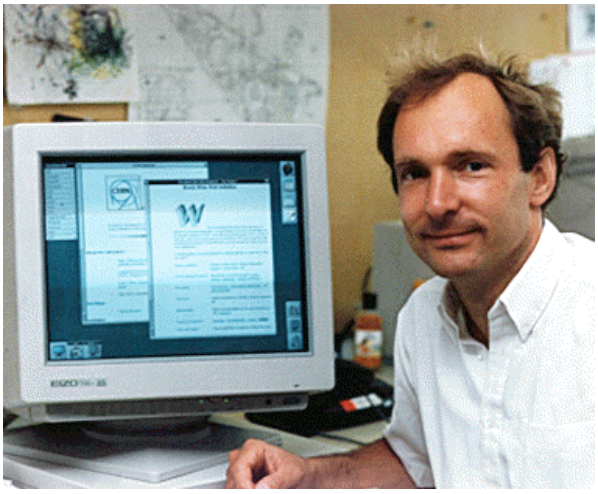


Figura 2.6: Tim Berners-Lee junto a una pantalla de ordenador que muestra su primer navegador web

- 1991 (junio): se celebra un seminario sobre **WWW** en el **CERN**.
- 1991 (agosto): se publican en Internet los ficheros del primer navegador.
- 1991 (diciembre): Paul Kunz instala el primer servidor web fuera de Europa en *Stanford Linear Accelerator Center* (**SLAC**).
- 1992: aparecen los primeros navegadores de terceras partes, Erwise, Viola y Midas.
- 1992: Marc Andreesen y Eric Bina comienzan a trabajar en un nuevo navegador gráfico para Unix en *National Center for Supercomputing Applications* (**NCSA**). Posee nuevas características innovadoras como: la etiqueta `<CENTER> ... </CENTER>`, la inclusión de imágenes en línea (antes se visualizaban aparte), navegación más sencilla a través de hiperenlaces que se pueden pulsar, etc.
- 1993 (febrero): Se publica el navegador gráfico NCSA Mosaic para X-Windows sobre Unix.

[8] `nxoc01.cern.ch`.

[9] `http://nxoc01.cern.ch/hypertext/WWW/TheProject.html`.

- 1993 (abril): Los directores del **CERN** anuncian que la tecnología **WWW** podrá ser usada gratuitamente por cualquiera, sin tener que pagar ningún tipo de licencia o canon.
- 1993 (noviembre): **NCSA** publica versiones de NCSA Mosaic para los sistemas operativos más extendidos: varios Unix, Microsoft Windows y Apple Macintosh.

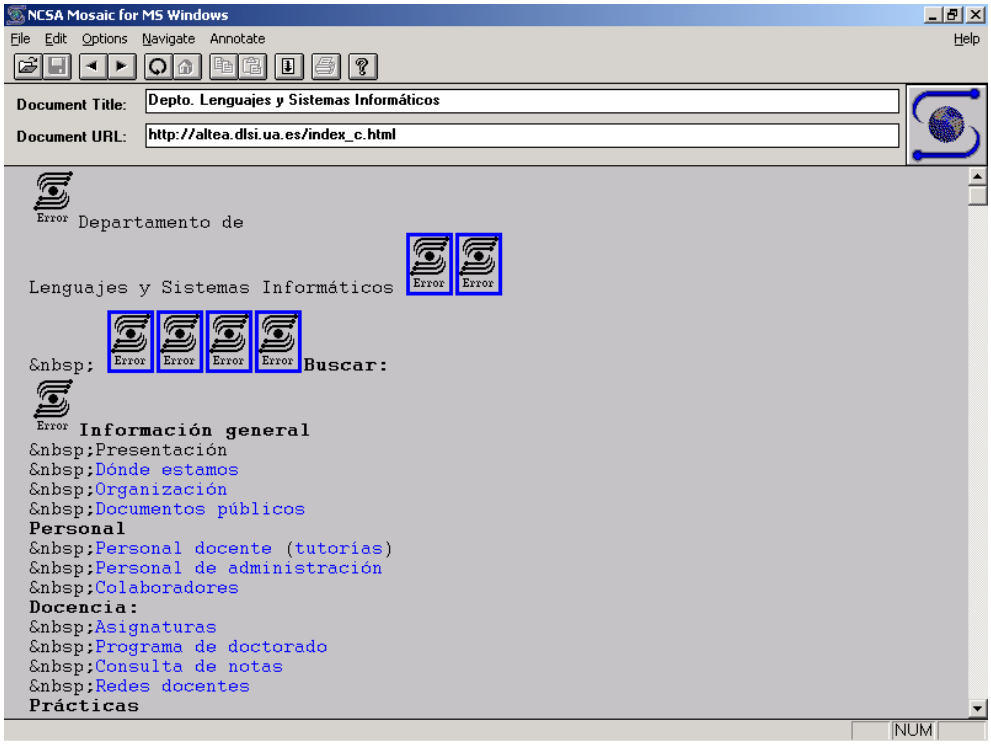


Figura 2.7: Página actual visualizada con Mosaic 1.0

En la Figura 2.7 se puede ver como se muestra la página principal del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Alicante[10] con la versión 1.0 de este navegador para Microsoft Windows 3.0. En la Figura 2.8 se muestra la misma página visualizada con un navegador actual (la versión 4.78 de Netscape Communicator). Como se puede apreciar, la versión 1.0 de NCSA Mosaic no estaba preparada para mostrar imágenes *Joint Photographic Experts Group* (**JPEG**), tablas, color de fondo de la página, etc. En la Figura 2.9 se puede ver el cuadro de diálogo **About** de este navegador.

[10] `http://altea.dlsi.ua.es/index_c.html`.

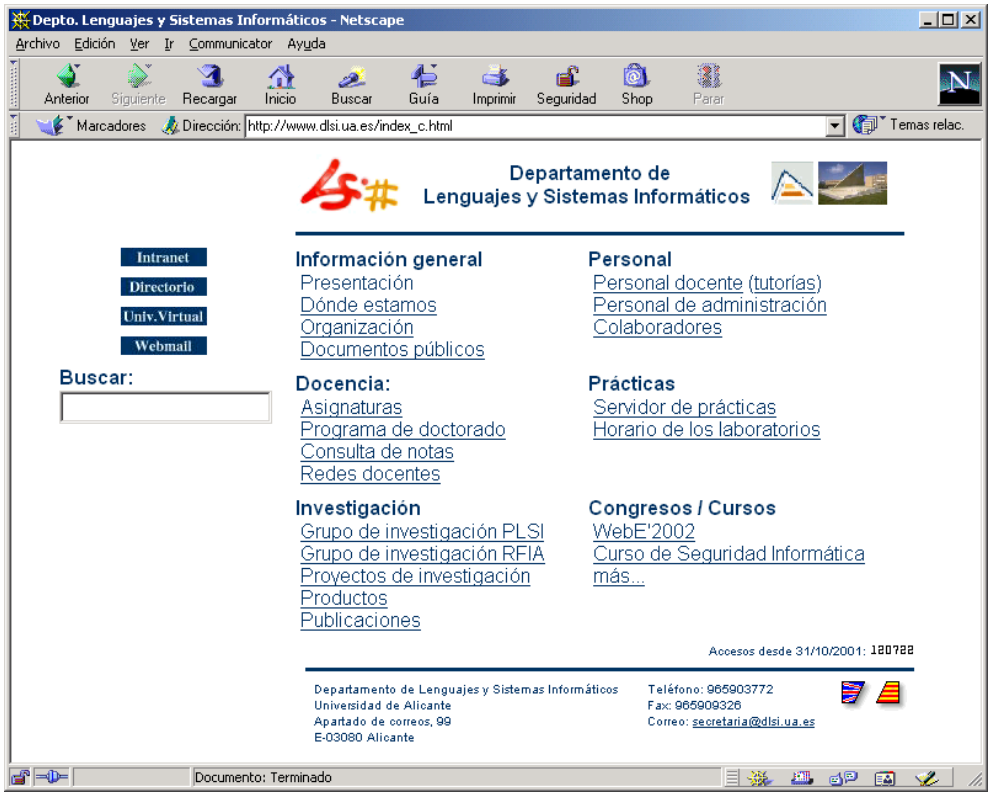


Figura 2.8: Página actual visualizada con Netscape Communicator 4.78

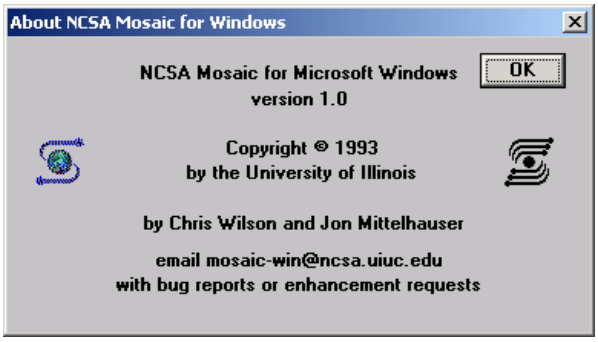


Figura 2.9: Cuadro de diálogo About en NCSA Mosaic 1.0

- 1993 (diciembre): Marc Andreessen abandona **NCSA** y se traslada a California.
- 1994 (marzo): Marc Andreessen y otros compañeros de **NCSA** forman MOSAIC COMMUNICATIONS CORPORATION, que más tarde, por problemas legales, pasará a llamarse NETSCAPE COMMUNICATIONS CORPORATION.
- 1994 (25 a 27 de mayo): Se celebra *First International WWW Conference* en el **CERN** en Ginebra (Suiza). La conferencia es todo un éxito.
- 1994 (agosto): La Universidad de Illinois firma un acuerdo de cesión de los derechos comerciales de NCSA Mosaic con la empresa SPYGLASS.
- 1994 (1 de octubre): Se funda *World Wide Web Consortium* (**W3C**).
- 1994 (17 a 19 de octubre): Se celebra *Second International WWW Conference* en Chicago (EE.UU.). Vuelve a ser un éxito completo.
- 1994 (diciembre): Se lanza al mercado Netscape Navigator 1.0.
- 1995 (10 al 14 de abril): Se celebra *Third International WWW Conference* en Darmstadt (Alemania).
- 1995 (mayo): SUN MICROSYSTEMS anuncia la existencia de *Java* 1.0 y NETSCAPE COMMUNICATIONS CORPORATION lo soportará en sus navegadores a través de los *applets*.
- 1995 (agosto): Coincidiendo con el lanzamiento de Microsoft Windows 95, se presenta Microsoft Internet Explorer 1.0, basado en código licenciado a SPYGLASS (que a su vez es una licencia comercial de NCSA Mosaic). A partir de entonces, comienza la "guerra de los navegadores". En la Figura 2.10 se puede ver el cuadro de diálogo **Acerca de Internet Explorer** del navegador Microsoft Internet Explorer 5.5. Se puede comprobar como aún esta versión se basa en NCSA Mosaic.
- 1995 (noviembre): Se lanza al mercado Microsoft Internet Explorer 2.0.
- 1996 (marzo): Se lanza al mercado Netscape Navigator 2.0. Incorpora nuevas características como elementos de **HTML** 3.0, marcos, la capacidad de ejecutar *applets* programados en *Java*, soporte de *JavaScript*, etc.
- 1996 (agosto): Se lanza al mercado Microsoft Internet Explorer 3.0. Proporciona soporte para marcos y programación con lenguajes de script (*JScript* y *VBScript*).
- 1996 (agosto): Se lanza al mercado Netscape Navigator 3.0.
- 1997 (enero): La versión 3.0 es la última versión de NCSA Mosaic.
- La "guerra de navegadores" continúa ...

Figura 2.10: Cuadro de diálogo Acerca de Internet Explorer en Microsoft Internet Explorer 5.5

2.2.1. El primer navegador

El primer navegador web, que también era editor, fue programado por Tim Berners-Lee a finales de 1990. Al principio lo llamó WorldWideWeb, pero después cambió el nombre por Nexus, ya que empezaba a usarse *World Wide Web* para referirse de forma genérica al sistema de comunicación que había ideado.

Este primer navegador se programó en *Objective-C* en un ordenador NeXT. Según Tim, le llevó un par de meses programarlo, gracias a que el sistema operativo NeXTStep facilitaba la programación al disponer de herramientas para construir los menús, tecnología *What You See Is What You Get* (**WYSIWYG**), etc.

En la Figura 2.11 y 2.12 se muestran dos imágenes del navegador. El navegador que se observa en la Figura 2.12 es una versión del año 1993. Se pueden observar varios aspectos interesantes en esta última imagen:

- El sistema operativo NeXTStep presenta un llamativo entorno gráfico multiventana.
- En la esquina superior izquierda se puede observar el menú del navegador, con su nombre en la primera línea (WorldWideWeb).
- Se pueden ver dos ventanas que muestran imágenes en línea (un libro y el logotipo del **CERN**). La primera versión del navegador mostraba las imágenes

en una ventana aparte.

- El menú **Links** aparece abierto. Se está creando un enlace sobre la palabra **ATLAS** que aparece resaltada en la ventana con título **CERN Experiments** (la ventana que se encuentra en un primer plano).

- La **X** que tienen las ventanas permite cerrarlas (más tarde lo copiaría MICROSOFT).

- La ventana que aparece al fondo con título **Tim's Home Page** tiene la **X** incompleta porque se ha modificado y no se han guardado los cambios.

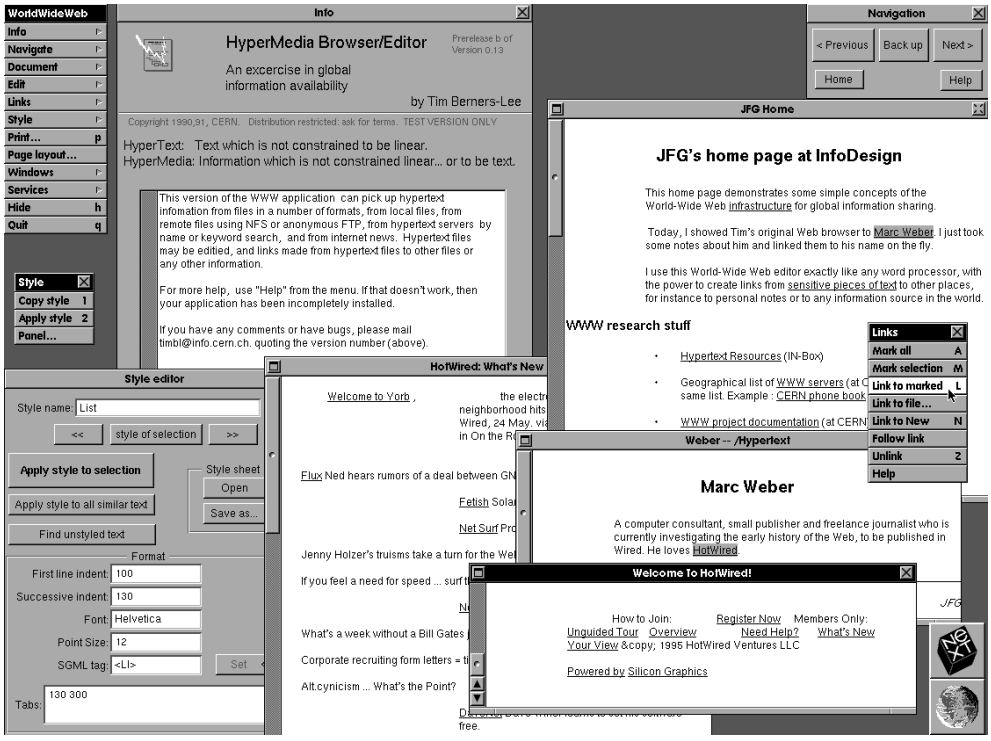


Figura 2.11: El primer navegador web ejecutándose en un ordenador NeXT

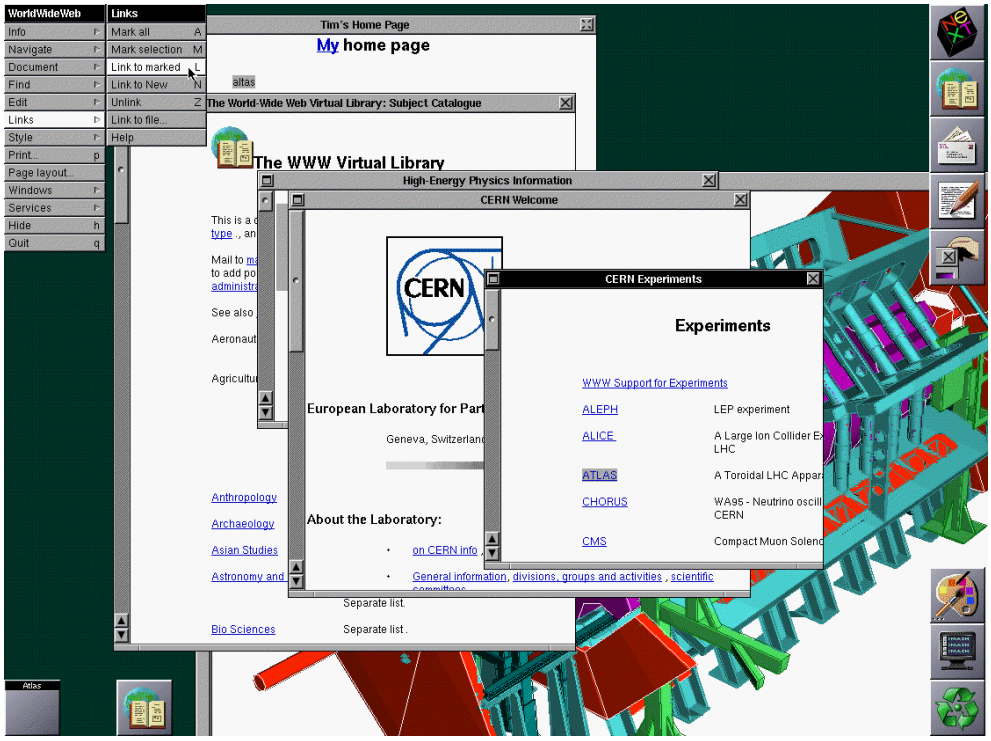


Figura 2.12: El primer navegador web ejecutándose en un ordenador NeXT

2.3. Generaciones de los sitios web

David Siegel, en su libro *Creating Killer Web Sites: The Art of Third-Generation Site Design*[11], estableció tres generaciones de sitios web. Sin embargo desde 1996, fecha en la que estableció su clasificación, ha evolucionado la tecnología empleada en la Web, por lo que se puede añadir una generación más a su clasificación.

Actualmente las cuatro generaciones conviven, aunque ya casi nadie crea sitios web que se clasifiquen en las dos primeras generaciones.

2.3.1. Primera generación

La primera generación abarca desde el nacimiento de la Web (1992) hasta mediados de 1994. La creación de páginas web durante esta generación se ve limitada por diversas razones tecnológicas: ancho de banda limitado (módems de 2.4 Kbps), navegadores poco desarrollados, monitores monocromos, etc.

Las características principales de estas páginas son:

- Tiempo de carga rápido: son páginas basadas en texto, con muy pocas imágenes y ningún recurso multimedia.
- Navegación poco estructurada, con falta de coherencia.
- Páginas largas, que parece que nunca se acaban. La información no se suele organizar en varias páginas, ya que así se reduce el número de transferencias.
- Texto escrito como si fuera una hoja de papel: de lado a lado de la página y desde el principio hasta el final.
- Empleo de saltos de línea como separadores.
- Empleo de líneas horizontales para separar secciones en una misma página.
- Empleo de listas para organizar la información.
- Poco uso de los enlaces entre páginas de un mismo sitio web.
- Como las páginas son muy largas, se emplean muchos enlaces intradocumentales.
- Listas interminables de enlaces a otros sitios web.
- Se pueden visualizar correctamente casi en cualquier navegador (incluso los navegadores no gráficos), pero son aburridas y poco legibles.
- Las páginas web poseen un contenido educativo o científico. Pocas empresas poseen un sitio web.

[11] David Siegel. *Creating Killer Web Sites: The Art of Third-Generation Site Design.* Hayden Books, 1996.

En definitiva, durante este primer periodo, se emplea la Web como si fuera uno de los medios de comunicación tradicionales (libros, revistas, etc.). Aún no se sabe como aprovechar todas las posibilidades que ofrece la Web.

Respecto a la generación de las páginas, no existe generación: las páginas son estáticas. A finales de este primer periodo aparece la tecnología *Common Gateway Interface* (**CGI**), que permite la generación dinámica de páginas web.

En la Figura 2.13[12] y 2.14[13] podemos observar dos ejemplos de páginas pertenecientes a la primera generación. Se pueden apreciar en estas páginas las principales características de esta generación: páginas simples, poco o nulo empleo de elementos gráficos, empleo de listas para organizar la información, empleo de líneas horizontales como separadores, etc.

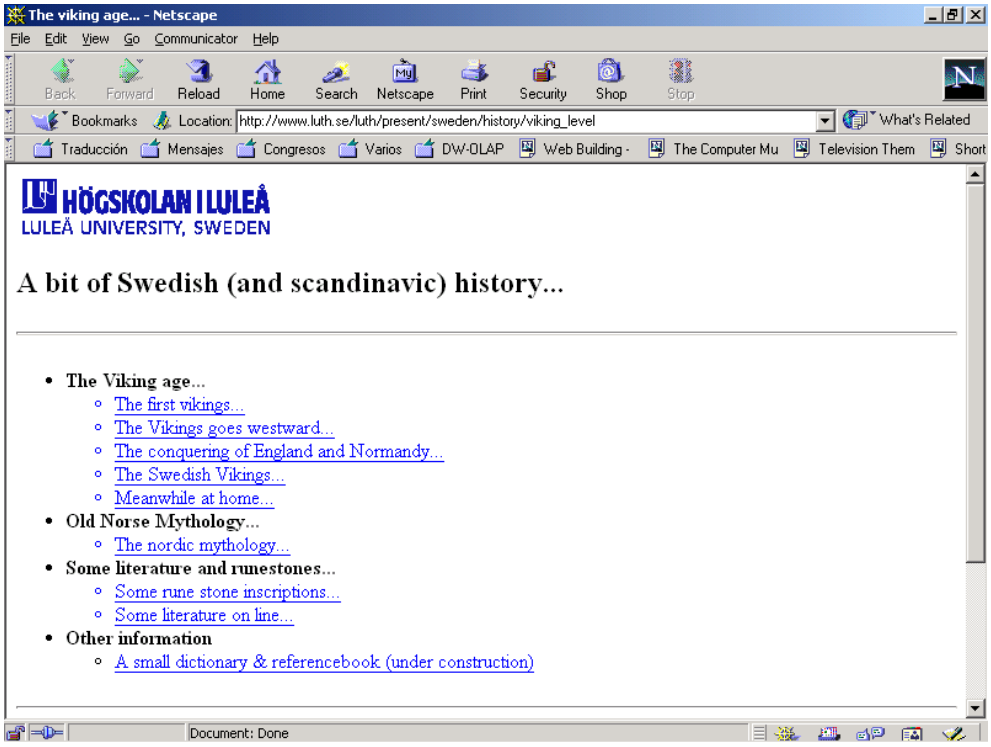


Figura 2.13: Ejemplo de página web de la primera generación

[12] `http://www.luth.se/luth/present/sweden/history/viking_level`.
[13] `http://www.library.cornell.edu/okuref/research/webeval.html`.

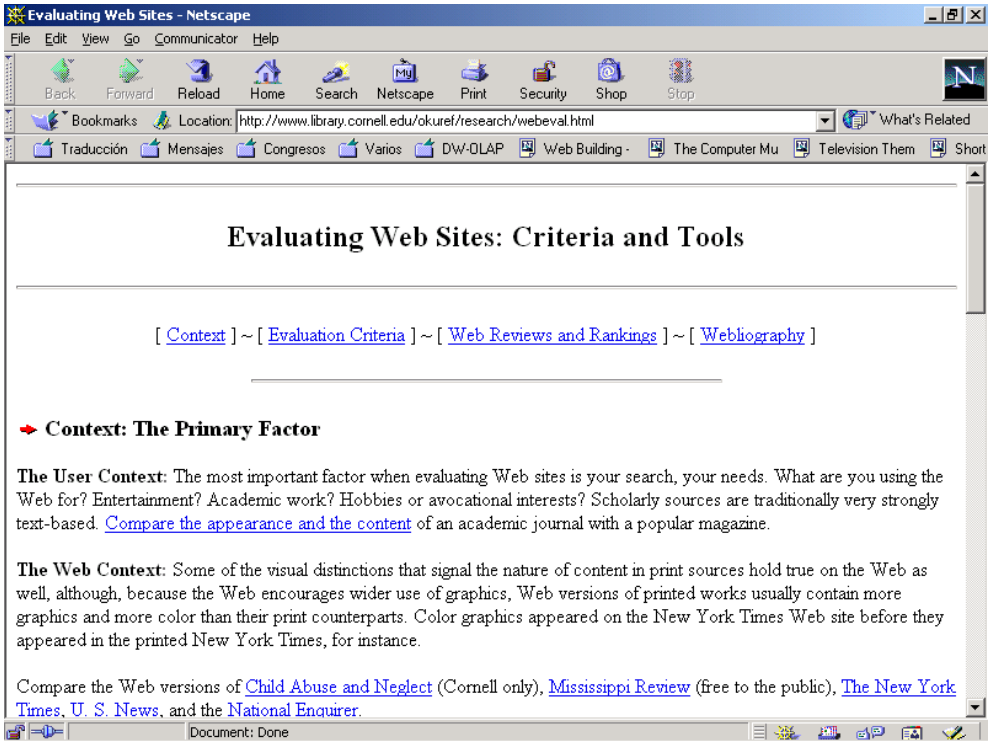


Figura 2.14: Ejemplo de página web de la primera generación

2.3.2. Segunda generación

La segunda generación se extiende desde 1995 hasta la actualidad. La diferencia principal con las páginas web de la generación anterior es la masiva incorporación de elementos gráficos en las páginas web:

- Los iconos sustituyen a las palabras.
- El color de fondo se sustituye por una imagen de fondo.
- Los *banners* sustituyen a los encabezados de las páginas.
- Las listas normales se sustituyen por listas con topos[14] (*bullets*) gráficos.

Sus características principales son:

- Tiempo de carga lento: se emplean imágenes con multitud de colores y animaciones en exceso, debido a la novedad de su uso. No se comprueba el rendimiento de las páginas con conexiones lentas: no se tiene en cuenta al usuario final.
- El color de fondo de las páginas deja de ser el blanco o el gris. Incluso, se emplean imágenes como fondo de las páginas.
- Empleo de tablas, aunque no con el propósito de situar el contenido (tablas invisibles), sino para mostrar datos tabulados.
- Las páginas todavía poseen una estructura de arriba a abajo.
- La navegación suele ser jerárquica, a partir de una página principal. Sin embargo, no existe una filosofía de planificación de la navegación.
- Aparecen tecnologías multimedia propietarias, que necesitan la instalación de un *plug-in* para su visualización. Prima el uso de tecnologías (imágenes y sonidos), aunque luego el público no pueda visualizar correctamente las páginas.

En definitiva, las páginas web de esta generación se caracterizan porque prima el uso de la tecnología, sin tener en cuenta el propósito del sitio web. Además, no se tiene en cuenta la legibilidad o claridad de la presentación de la información.

Respecto a la generación de las páginas, la mayoría siguen siendo estáticas, aunque cada vez se emplea más la tecnología **CGI**. El uso de esta tecnología abre un abanico de posibilidades enorme: la creación de aplicaciones web que acceden a bases de datos. Las primeras aplicaciones que se desarrollan son pequeñas y sencillas: libro de visitas, formulario de más información, etc. Si se necesita almacenar información de forma persistente, se emplean ficheros en vez de bases de datos.

[14] Los topos o bolos son caracteres de imprenta o elementos gráficos que representan un figura geométrica y que se emplean para destacar el comienzo de un párrafo o apartado, un sumario, las acepciones de un diccionario, etc.

En la Figura 2.15[15] y 2.16[16] se pueden observar dos ejemplos de páginas pertenecientes a esta segunda generación. Se pueden apreciar las principales características de esta generación: empleo de imágenes como fondo de la página, uso excesivo de elementos gráficos (imágenes, iconos), listas con topos gráficos, empleo de tablas, etc.

Figura 2.15: Ejemplo de página web de la segunda generación

2.3.3. Tercera generación

La tercera generación aparece a mediados de 1996. Las páginas pertenecientes a esta generación son las más comunes en la actualidad. Se caracterizan por:

- Tiempo de carga rápido: los creadores de las páginas se centran en el contenido y no en la presentación. Se minimiza el tiempo de carga mediante un uso minimalista de los recursos gráficos, el uso de *Cascading Style Sheets* (**CSS**) y

[15] `http://www.ua.es/ursua/`.
[16] `http://www.cehipar.es/index.htm`.

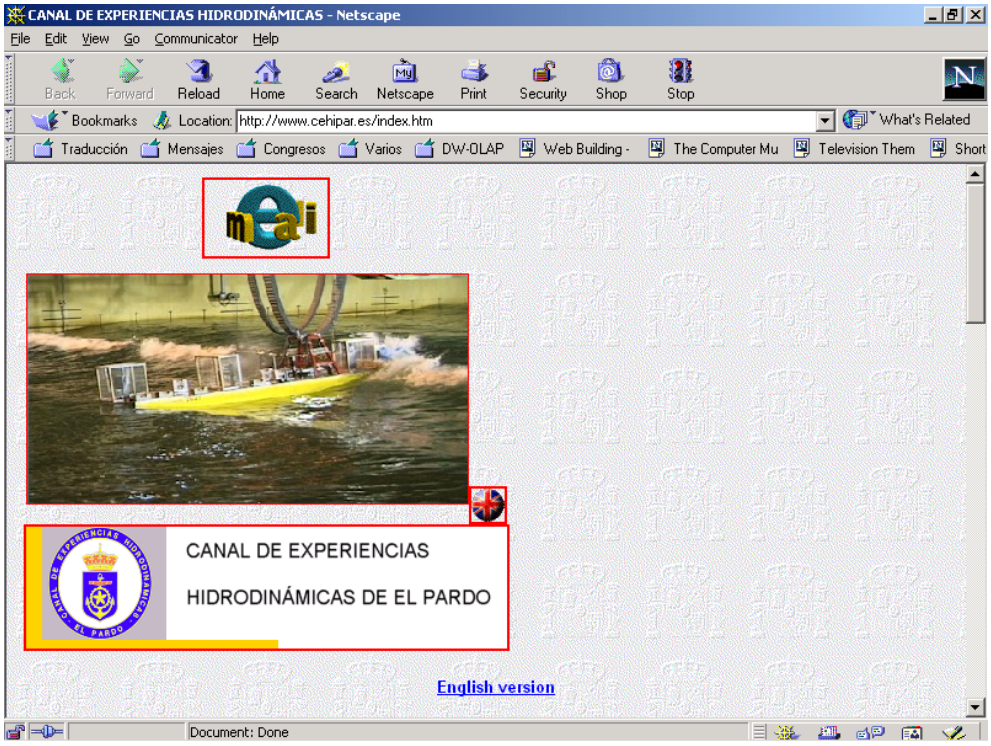


Figura 2.16: Ejemplo de página web de la segunda generación

la optimización del código **HTML**. El rendimiento de las páginas se verifica empleando conexiones a distintas velocidades.

- Las páginas se limitan para que se puedan visualizar completamente en una pantalla, sin tener que realizar desplazamiento (*scroll*).
- Los sitios web se crean teniendo en cuenta los posibles usuarios y el objetivo del sitio (informar, vender, ofrecer servicios, etc.).
- Se limita el número de enlaces, se simplifica la navegación. Se organiza la información a partir de una página inicial hasta una página final, ofreciendo distintos caminos.
- Se tienen en cuenta principios tipográficos y de organización visual de la información.
- Se emplean metáforas y temas visuales para seducir y guiar al usuario, creando una experiencia completa desde la primera página hasta la última.
- Se incorporan los principios de usabilidad y accesibilidad.
- Se comprueba con usuarios reales el funcionamiento de los sitios web.
- En los sitios web de las empresas cobra importancia la creación de una identidad corporativa. Se emplean de forma coherente los colores, las imágenes, los símbolos e iconos, los tipos de letra, etc.

La característica principal de las páginas web pertenecientes a la tercera generación es la planificación: los diseñadores invierten tiempo en analizar los posibles caminos que los visitantes tomarán al visitar un sitio web, y en función de ello diseñan los sitios web. La estructura del sitio web cobra una gran importancia.

Durante este periodo tiene lugar una "explosión" en el número de herramientas informáticas relacionadas con la Web.

En cuanto a la generación de las páginas, este periodo supone la consolidación de la generación de páginas web dinámicas. El uso de **CGI** está muy extendido, pero debido a sus limitaciones aparecen nuevas tecnologías. Las primeras soluciones relevantes provienen de MICROSOFT, primero con *Internet Database Connector* (**IDC**) y luego con *Active Server Pages* (**ASP**), que supone una verdadera revolución en la creación de páginas web dinámicas. A partir de ahí aparecen nuevas tecnologías ColdFusion, PHP o *Java Server Pages* (**JSP**) basada en Java.

En la Figura 2.17[17] y 2.18[18] se pueden observar dos ejemplos de páginas pertenecientes a la tercera generación. En estas páginas las principales características que se pueden apreciar son: el tamaño de las páginas se limita para que quepan en el área de una ventana, se limita el número de enlaces, se simplifica la navegación, se tienen en cuenta principios tipográficos y de organización de la información, etc.

[17] `http://www.renault.es`.
[18] `http://www.ua.es/es/index.html`.

Figura 2.17: Ejemplo de página web de la tercera generación

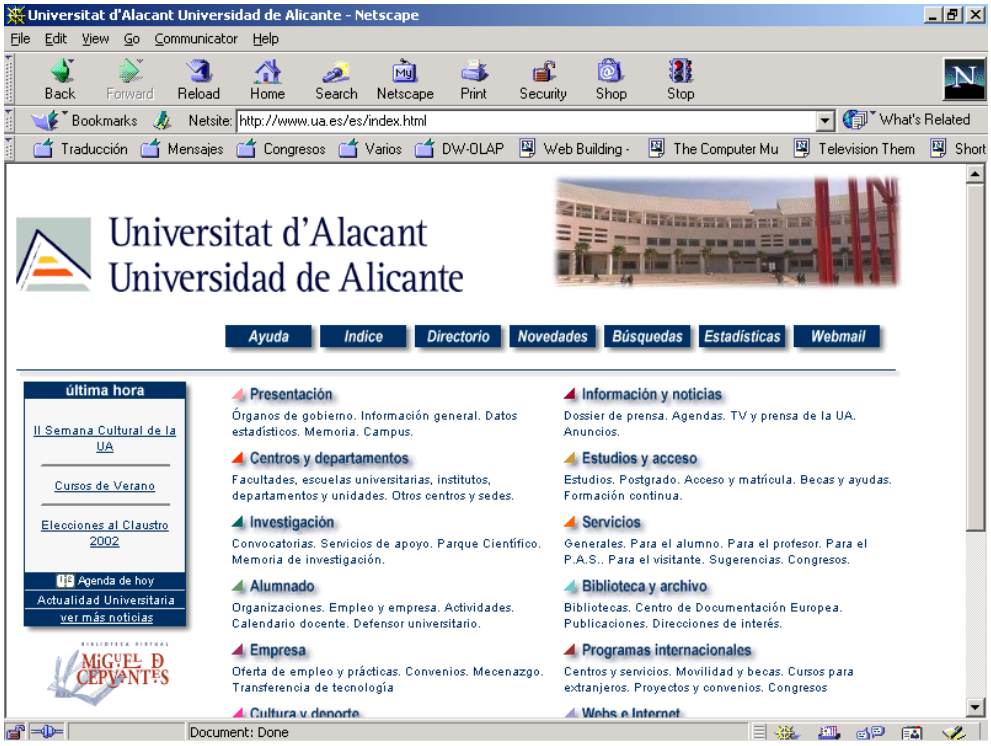


Figura 2.18: Ejemplo de página web de la tercera generación

2.3.4. Cuarta generación

La cuarta y última generación empieza a desarrollarse plenamente en 1999 y discurre hasta la actualidad. Sus características principales son:

- Se vuelven a emplear en exceso los recursos gráficos.
- En muchos casos se intenta aprovechar hasta el último pixel de la página para presentar información.
- **HTML** evoluciona: se extiende el uso de tecnologías poco empleadas hasta ese momento, como **CSS**, y aparecen nuevas tecnologías, como *Dynamic HTML* (**DHTML**). Estas tecnologías permiten un mayor control sobre la visualización de las páginas web, pero a costa de incompatibilidades entre distintos navegadores.
- Uso de nuevas tecnologías multimedia (como Macromedia Flash[19]): se puede crear un sitio web sin tener que emplear **HTML**.
- Los principios empleados en la creación de CD-ROM interactivos se aplican en la creación de páginas web.
- Un equipo interdisciplinar (informático, experto en contenidos, diseñador gráfico, etc.) desarrolla los sitios web.
- El aumento del ancho de banda permite *streaming* de video y audio en tiempo real.
- El objetivo al desarrollar un sitio web es crear una experiencia completa desde que el visitante visualiza la primera página hasta que abandona el sitio web.

Respecto a la generación de páginas web dinámicas supone la consolidación de las tecnologías de generación dinámica. La mayoría de las páginas web pertenecientes a esta generación se crean a partir de información almacenada en bases de datos.

En la Figura 2.19[20], 2.20[21] y 2.21[22] se pueden observar varios ejemplos de páginas pertenecientes a la cuarta generación. En estas páginas podemos detectar las principales características de esta generación: uso excesivo de elementos gráficos, empleo de tecnologías multimedia como Macromedia Flash, aprovechamiento hasta el último pixel de la página (en la página de la Figura 2.19), etc.

[19] En diciembre de 1996, Macromedia Inc. compró a Jonathan Gay su herramienta de animación FutureWave Software, que pasó a llamarse Macromedia Flash.

[20] `http://www.terra.es`.

[21] `http://www.repsolypf.com/esp/home/home.asp`.

[22] `http://www.mde.es`.

Figura 2.19: Ejemplo de página web de la cuarta generación

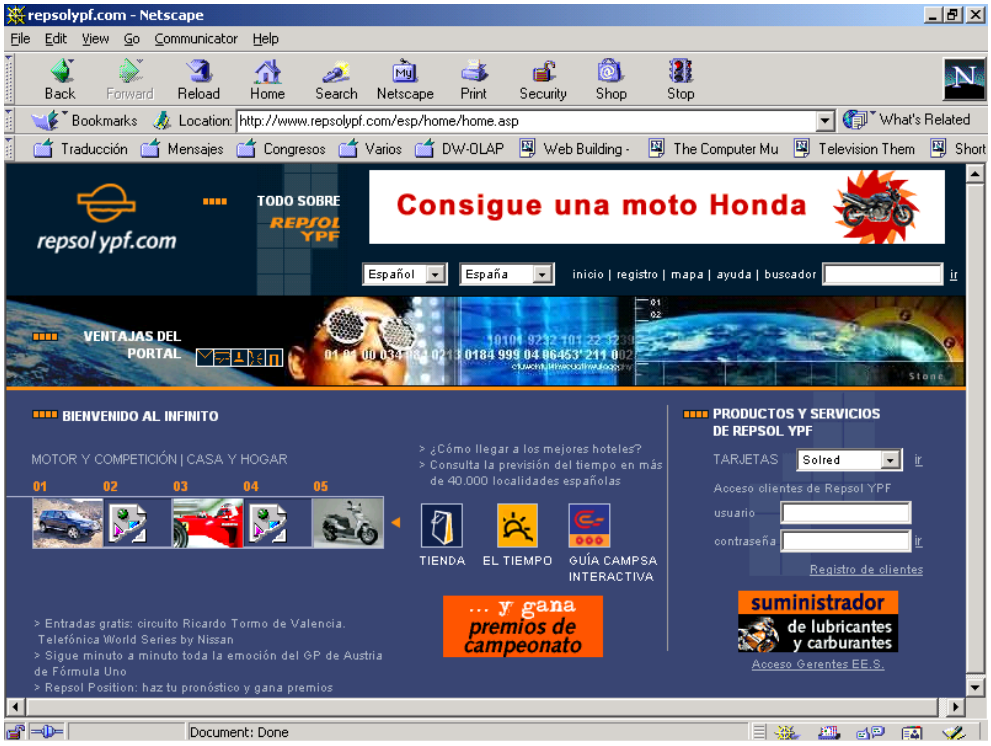


Figura 2.20: Ejemplo de página web de la cuarta generación

Figura 2.21: Ejemplo de página web de la cuarta generación

Capítulo 3

Arquitecturas cliente/servidor

Las aplicaciones web son un tipo especial de aplicaciones cliente/servidor. Antes de aprender a programar aplicaciones web conviene conocer las características básicas de las arquitecturas cliente/servidor.

Índice General

3.1. Introducción

Cliente/servidor es una arquitectura de red[1] en la que cada ordenador o proceso en la red es **cliente** o **servidor**[2]. Normalmente, los servidores son ordenadores potentes dedicados a gestionar unidades de disco (servidor de ficheros), impresoras (servidor

[1] Otro tipo de arquitectura de red es *peer-to-peer* (entre pares o de igual a igual), en la que cada ordenador de la red posee responsabilidades equivalentes.

[2] Aunque un mismo ordenador puede ser cliente y servidor simultáneamente, se establece una separación lógica según las funciones que realiza.

de impresoras), tráfico de red (servidor de red), datos (servidor de bases de datos) o incluso aplicaciones (servidor de aplicaciones), mientras que los clientes son máquinas menos potentes y usan los recursos que ofrecen los servidores.

Dentro de los clientes se suelen distinguir dos clases: los clientes inteligentes (*rich client*) y los clientes tontos (*thin client*). Los primeros son ordenadores completos, con todo el hardware y software necesarios para poder funcionar de forma independiente. Los segundos son terminales que no pueden funcionar de forma independiente, ya que necesitan de un servidor para ser operativos.

Esta arquitectura implica la existencia de una relación entre procesos que solicitan servicios (**clientes**) y procesos que responden a estos servicios (**servidores**). Estos dos tipos de procesos pueden ejecutarse en el mismo procesador o en distintos.

La arquitectura cliente/servidor permite la creación de aplicaciones distribuidas. La principal ventaja de esta arquitectura es que facilita la separación de las funciones según su servicio, permitiendo situar cada función en la plataforma más adecuada para su ejecución. Además, también presenta las siguientes ventajas:

- Las redes de ordenadores permiten que múltiples procesadores puedan ejecutar partes distribuidas de una misma aplicación, logrando concurrencia de procesos.
- Existe la posibilidad de migrar aplicaciones de un procesador a otro con modificaciones mínimas en los programas.
- Se obtiene una escalabilidad de la aplicación. Permite la ampliación horizontal o vertical de las aplicaciones. La **escalabilidad horizontal** se refiere a la capacidad de añadir o suprimir estaciones de trabajo que hagan uso de la aplicación (clientes), sin que afecte sustancialmente al rendimiento general (Figura 3.1). La **escalabilidad vertical** se refiere a la capacidad de migrar hacia servidores de mayor capacidad o velocidad, o de un tipo distinto de arquitectura sin que afecte a los clientes (Figura 3.2).
- Posibilita el acceso a los datos independientemente de donde se encuentre el usuario.

3.2. Separación de funciones

La arquitectura cliente/servidor nos permite la separación de funciones en tres niveles, tal como se muestra en la Figura 3.3:

- **Lógica de presentación**. Se encarga de la entrada y salida de la aplicación con el usuario. Sus principales tareas son: obtener información del usuario, enviar la información del usuario a la lógica de negocio para su procesamiento, recibir los resultados del procesamiento de la lógica de negocio y presentar estos resultados al usuario.

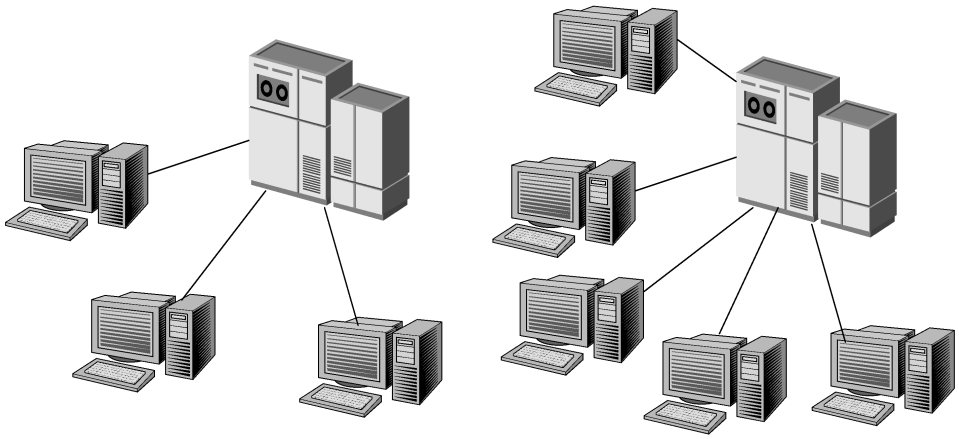


Figura 3.1: Escalabilidad horizontal

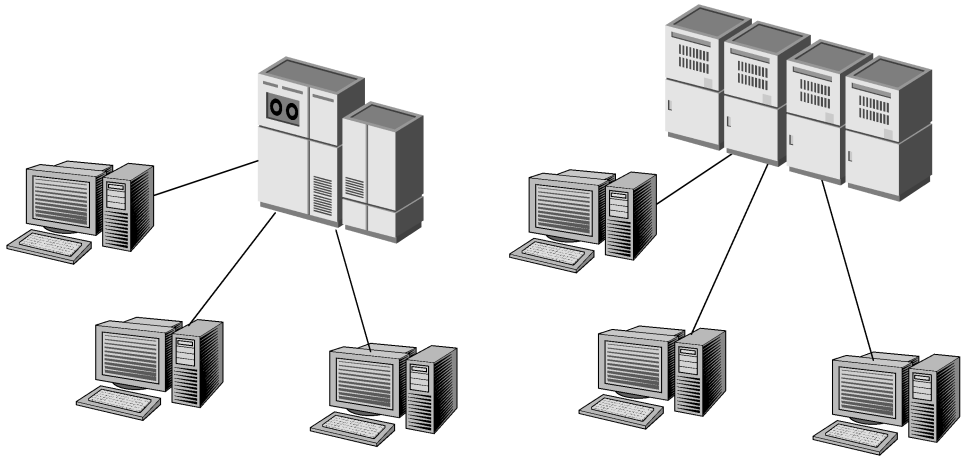


Figura 3.2: Escalabilidad vertical

- **Lógica de negocio (o aplicación)**. Se encarga de gestionar los datos a nivel de procesamiento. Actúa de puente entre el usuario y los datos. Sus principales tareas son: recibir la entrada del nivel de presentación, interactuar con la lógica de datos para ejecutar las reglas de negocio (*business rules*) que tiene que cumplir la aplicación (facturación, cálculo de nóminas, control de inventario, etc.) y enviar el resultado del procesamiento al nivel de presentación.

- **Lógica de datos**. Se encarga de gestionar los datos a nivel de almacenamiento. Sus principales tareas son: almacenar los datos, recuperar los datos, mantener los datos y asegurar la integridad de los datos.

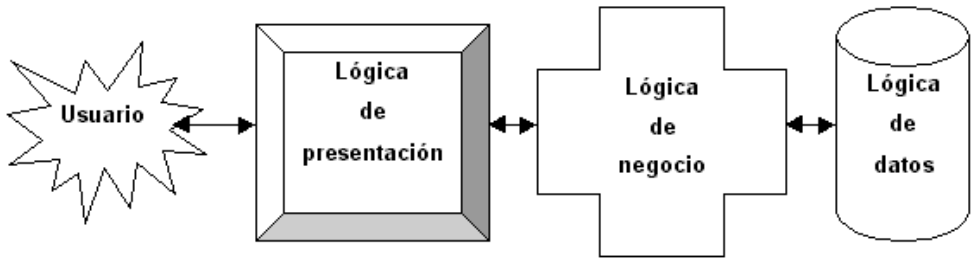


Figura 3.3: Separación de funciones

Si un sistema distribuido se diseña correctamente, los tres niveles anteriores pueden distribuirse y redistribuirse independientemente sin afectar al funcionamiento de la aplicación.

3.3. Modelos de distribución en aplicaciones cliente/-servidor

Según como se distribuyan las tres funciones básicas de una aplicación (presentación, negocio y datos) entre el cliente y el servidor, podemos contemplar tres modelos: presentación distribuida, aplicación distribuida y datos distribuidos.

3.3.1. Presentación distribuida

El cliente sólo mantiene la presentación, el resto de la aplicación se ejecuta remotamente (Figura 3.4). La presentación distribuida, en su forma más simple, es una interfaz gráfica de usuario a la que se le pueden acoplar controles de validación de datos, para evitar la validación de los mismos en el servidor.

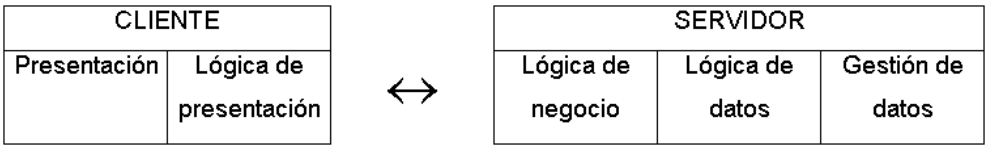


Figura 3.4: Presentación distribuida

3.3.2. Aplicación distribuida

Es el modelo que proporciona máxima flexibilidad, puesto que permite tanto a servidor como a cliente mantener la lógica de negocio realizando cada uno las funciones que le sean más propias, bien por organización, o bien por mejora en el rendimiento del sistema (Figura 3.5).

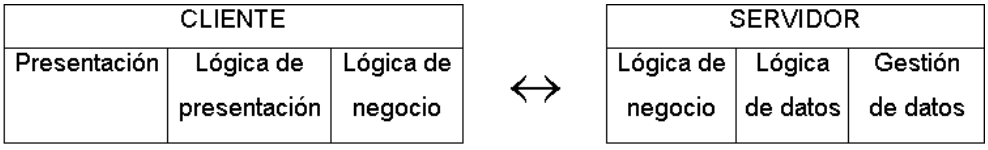


Figura 3.5: Aplicación distribuida

3.3.3. Datos distribuidos

Los datos son los que se distribuyen, por lo que la lógica de datos es lo que queda separada del resto de la aplicación (Figura 3.6). Se puede dar de dos formas: ficheros distribuidos o bases de datos distribuidas.

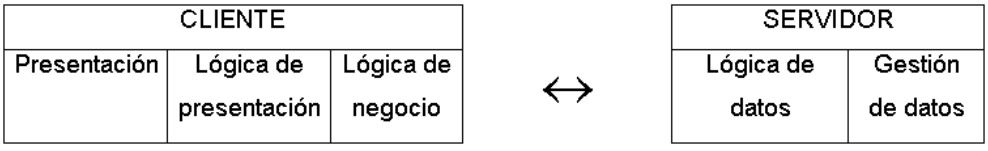


Figura 3.6: Datos distribuidos

3.4. Arquitecturas de dos y tres niveles

La diferencia entre las arquitecturas de dos y tres niveles (o capas) estriba en la forma de distribución de la aplicación entre el cliente y el servidor.

Aunque todos los modelos de distribución en aplicaciones cliente/servidor que se han comentado antes se basan en arquitecturas de dos capas, normalmente cuando se habla de aplicaciones de dos niveles se está haciendo referencia a una aplicación donde el cliente mantiene la lógica de presentación, de negocio, y de acceso a los datos, y el servidor únicamente gestiona los datos. Suelen ser aplicaciones cerradas que supeditan la lógica de los procesos cliente al gestor de bases de datos que se está usando.

En las arquitecturas de tres niveles, la lógica de presentación, la lógica de negocio y la lógica de datos están separadas, de tal forma que mientras la lógica de presentación se ejecutará normalmente en la estación cliente, la lógica de negocio y la de datos pueden estar repartidas entre distintos procesadores. En este tipo de aplicaciones suelen existir dos servidores: uno contiene la lógica de negocio y otro la lógica de datos. En la Figura 3.7 se muestra un esquema de este tipo de arquitectura.

El objetivo de aumentar el número de niveles en una aplicación distribuida es lograr una mayor independencia entre un nivel y otro, lo que facilita la portabilidad en entornos heterogéneos y la escalabilidad en caso de incorporación de nuevos clientes.

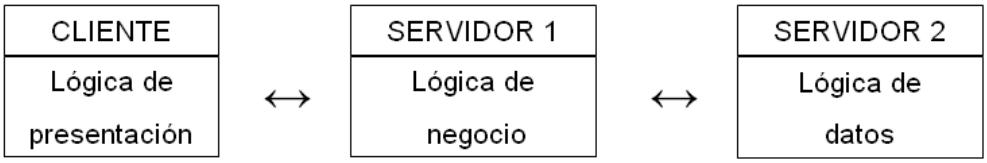


Figura 3.7: Arquitectura de tres niveles

3.5. Descripción de un sistema cliente/servidor

Un sistema cliente/servidor suele presentar las siguientes características:

1. Una combinación de la parte cliente (también llamada *front-end*) que interactúa con el usuario (hace de interfaz entre el usuario y el resto de la aplicación) y la parte servidor (o *back-end*) que interactúa con los recursos compartidos (bases de datos, impresoras, módems).

2. La parte cliente y servidor tienen diferentes necesidades de recursos a la hora de ejecutarse: velocidad de procesador, memoria, velocidad y capacidad de los discos duros, dispositivos de entrada/salida, etc.

3. El entorno suele ser heterogéneo y multivendedor[3]. El hardware y sistema operativo del cliente y el servidor suelen diferir. El cliente y el servidor se suelen comunicar a través de una *Application Program Interface* (**API**) y *Remote Procedure Call* (**RPC**) conocidas (por ejemplo, *Open DataBase Connectivity* (**ODBC**) para acceder a bases de datos).

4. Normalmente la parte cliente se implementa haciendo uso de una interfaz gráfica de usuario, que permite la introducción de datos a través de teclado, ratón, lápiz óptico, etc.

[3]Se emplea hardware y software de distintos fabricantes.

Capítulo 4

Qué es una aplicación web

En las aplicaciones web suelen distinguirse tres niveles (como en las arquitecturas cliente/servidor de tres niveles): el nivel superior que interacciona con el usuario (el cliente web, normalmente un navegador), el nivel inferior que proporciona los datos (la base de datos) y el nivel intermedio que procesa los datos (el servidor web). En este capítulo se describen el cliente y el servidor web y se comentan los entornos web en los que se ejecutan las aplicaciones web: Internet, intranet y extranet. Además se comentan las principales ventajas que poseen las aplicaciones web. También se describen las arquitecturas típicas de las aplicaciones web. Por último, se presenta una metodología de desarrollo de sitios web.

Índice General

4.1. Introducción

Una aplicación web (*web-based application*) es un tipo especial de aplicación cliente/servidor, donde tanto el **cliente** (el navegador, explorador o visualizador[1]) como el **servidor** (el servidor web) y el **protocolo** mediante el que se comunican (**HTTP**) están estandarizados y no han de ser creados por el programador de aplicaciones (Figura 4.1).

El protocolo **HTTP** forma parte de la familia de protocolos de comunicaciones **TCP/IP**, que son los empleados en Internet. Estos protocolos permiten la conexión de sistemas heterogéneos, lo que facilita el intercambio de información entre distintos ordenadores. **HTTP** se sitúa en el nivel 7 (aplicación) del modelo **OSI**.

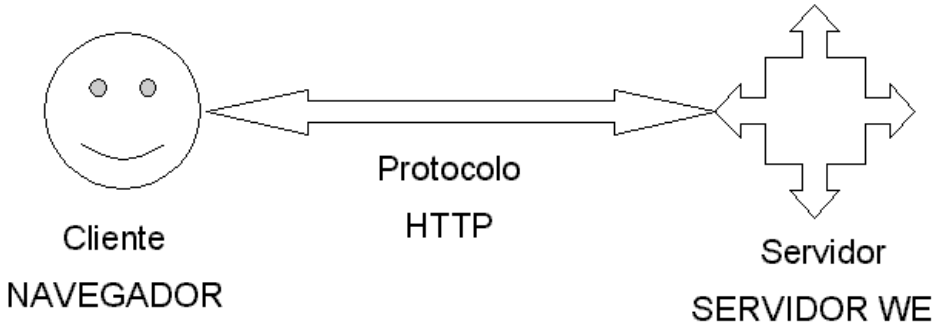


Figura 4.1: Esquema básico de una aplicación web

4.1.1. El cliente

El cliente web es un programa con el que interacciona el usuario para solicitar a un servidor web el envío de los recursos que desea obtener mediante **HTTP**[2].

La parte cliente de las aplicaciones web suele estar formada por el código **HTML** que forma la página web más algo de código ejecutable realizado en lenguaje de *script* del navegador (*JavaScript* o *VBScript*) o mediante pequeños programas (*applets*) realizados en *Java*. También se suelen emplear *plug-ins*[3] que permiten visualizar otros

[1] En inglés se le suele denominar *browser*.

[2] Los clientes web también suelen actuar como clientes de transferencia de archivos (FTP), lectores de correo (SMTP y POP) y grupos de noticias (NNTP), etc.

[3] Un *plug-in* (o *add-in*) es un módulo de software que se instala como un añadido a un programa o sistema y que proporciona nuevas características o servicios al programa o sistema. En los navegadores, suelen permitir la reproducción de diferentes tipos de recursos de audio o vídeo.

contenidos multimedia (como Macromedia Flash[4]), aunque no se encuentran tan extendidos como las tecnologías anteriores y plantean problemas de incompatibilidad entre distintas plataformas. Por tanto, la misión del cliente web es interpretar las páginas **HTML** y los diferentes recursos que contienen (imágenes, sonidos, etc.).

Las tecnologías que se suelen emplear para programar el cliente web son:

- **HTML**.
- **CSS**.
- **DHTML**.
- Lenguajes de script: *JavaScript*, *VBScript*, etc.
- *ActiveX*.
- *Applets* programados en *Java*.
- Distintas tecnologías que necesitan la existencia de un *plug-in* en el navegador: Adobe Acrobat Reader, Autodesk MapGuide, Live Picture PhotoVista, Macromedia Flash, Macromedia Shockwave, *Virtual Reality Modeling Language* (**VRML**), etc.

4.1.2. El servidor

El servidor web es un programa que está esperando permanentemente las solicitudes de conexión mediante el protocolo **HTTP** por parte de los clientes web. En los sistemas Unix suele ser un "demonio" y en los sistemas Microsoft Windows un servicio.

La parte servidor de las aplicaciones web está formada por:

- Páginas estáticas (documentos **HTML**) que siempre muestran el mismo contenido.
- Recursos adicionales (multimedia, documentos adicionales, etc.) que se pueden emplear dentro de las páginas o estar disponibles para ser descargados y ejecutados (visualizados) en el cliente.
- Programas o *scripts* que son ejecutados por el servidor web cuando el navegador del cliente solicita algunas páginas. La salida de este *script* suele ser una página **HTML** estándar que se envía al navegador del cliente. Tradicionalmente este programa o *script* que es ejecutado por el servidor web se basa en la tecnología **CGI**. En algunos casos pueden acceder a bases de datos.

[4] Tecnología de animación vectorial independiente de la plataforma, creada por Macromedia Inc.

La programación del servidor mediante **CGI** es compleja y laboriosa. El protocolo **HTTP** no almacena el estado entre una conexión y la siguiente (es un protocolo sin estado), por lo que es el programador el que se tiene que encargar de conservarlo. Esto conduce a que el programador tenga que dedicar parte de su tiempo a programar tareas ajenas al fin de la aplicación, lo que suele ser origen de diversos problemas.

Sin embargo, con la entrada en 1995 de MICROSOFT en el mundo Internet y la salida al mercado de su servidor web (Microsoft Internet Information Server) se abrió un nuevo campo para las aplicaciones web: *Internet Server Application Program Interface* (**ISAPI**). Se trata de un conjunto de funciones que el servidor web pone a disposición de los programadores de aplicaciones web. Con **ISAPI**, los programadores pueden crear *Dynamic Link Library* (**DLL**) con funciones que son invocadas para determinados archivos (se ejecutan cuando el cliente solicita un archivo con una determinada extensión).

Por ejemplo, todo el sistema **ASP**, no es más que una **DLL** del tipo **ISAPI** que es invocada automáticamente para los archivos cuya extensión sea `.asp`[5]. La **DLL ASP** preprocesa el archivo `.asp` interpretando su código como un *script* a ejecutar en el servidor. Sin embargo, ella no interpreta directamente el código, sino que en función del lenguaje en el que está escrito (*VBScript*, *JavaScript*, etc.), invoca a otra **DLL** que se encarga de ejecutar el *script*. Después recoge la salida y se la envía al servidor web, el cual a su vez la reenvía al cliente.

Las ventajas que presenta **ASP** frente a **CGI** son:

- Las páginas basadas en **CGI** resultan difíciles de mantener, ya que las instrucciones **HTML** se encuentran insertadas en el propio código del programa **CGI**, mezclándose sus funcionalidades.
- La ejecución de un programa **CGI** es muy ineficiente, debido al proceso de carga del código en memoria que se realiza cada vez que un usuario requiere su ejecución. La existencia de múltiples clientes simultáneos supone múltiples copias del programa en memoria del servidor.
- La unión de **ISAPI** con el servidor web es más "fuerte" (están más integrados), su ejecución es más rápida, con lo que se logra que las aplicaciones basadas en **ISAPI** tengan un mayor rendimiento que las basadas en **CGI**.
- La tecnología **ASP** ofrece una serie de mecanismos (gestión de sesiones, variables globales, etc.) que facilitan la programación de aplicaciones web.

Además de **ASP**, existen otras tecnologías destinadas a programar la parte servidor de las aplicaciones web: ColdFusion, **JSP**, *servlets*, PHP, etc. Todas ellas son muy similares, se basan en los mismos principios y ofrecen prestaciones y resultados equivalentes.

[5] En concreto, se trata del fichero `asp.dll` que tiene poco más de 338 Kb en su versión 3.0 y que normalmente se instala en `C:\WINDOWS\system32\inetsrv`.

En la Figura 4.2 se han resumido las tecnologías que se emplean en la actualidad para programar el cliente y el servidor de las aplicaciones web.

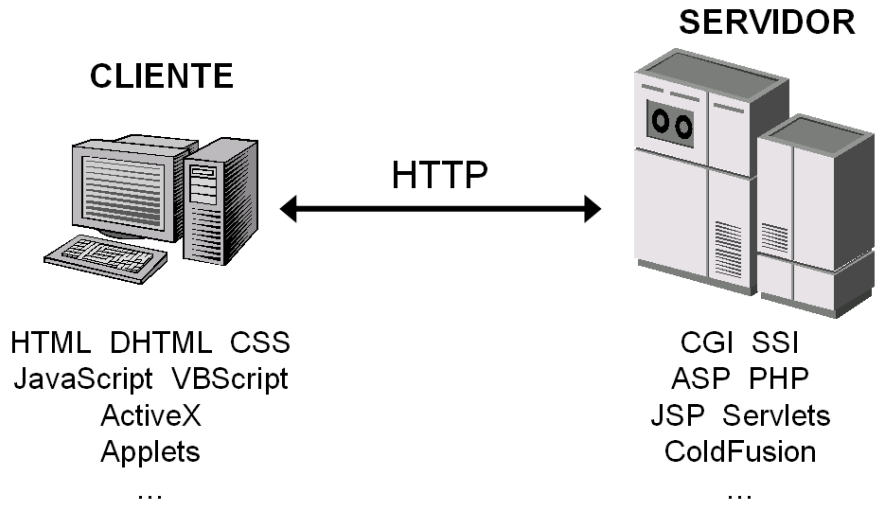


Figura 4.2: Tecnologías empleadas en el cliente y en el servidor web

4.2. Transferencia de páginas web

El proceso completo, desde que el usuario solicita una página, hasta que el cliente web (navegador) se la muestra con el formato apropiado, es el siguiente:

1. El usuario especifica en el cliente web la dirección de la página que desea consultar: el usuario escribe en el navegador la dirección (**URL**) de la página que desea visitar o pulsa un enlace.
2. El cliente establece una conexión con el servidor web.
3. El cliente solicita la página o el objeto deseado.
4. El servidor envía dicha página u objeto (o, si no existe, devuelve un código de error).
5. Si se trata de una página **HTML**, el cliente inicia sus labores de interpretación de los códigos **HTML**. Si el cliente web encuentra instrucciones que hacen referencia a otros objetos que se tienen que mostrar con la página (imágenes, sonidos, animaciones multimedia, etc.), establece automáticamente comunicación con el servidor web para solicitar dichos objetos.

6. Se cierra la conexión entre el cliente y el servidor.

7. Se muestra la página al usuario.

Obsérvese que siempre se libera la conexión, por lo que ésta sólo tiene la duración correspondiente a la transmisión de la página solicitada. Esto se hace así para no desperdiciar innecesariamente el ancho de banda de la red mientras el usuario lee la página recibida.

Cuando el usuario activa un enlace de la página, se establece una nueva conexión para recibir otra página o elemento multimedia. Por ello, el usuario tiene la sensación de que está disfrutando de una conexión permanente cuando realmente no es así.

Un detalle importante es que para cada objeto que se transfiere por la red se realiza una conexión independiente. Por ejemplo, si el cliente web solicita una página que contiene dos imágenes integradas, se realizan tres conexiones: una para el documento **HTML** y dos para los archivos de las imágenes[6].

4.3. Entornos web

Las aplicaciones web se emplean en tres entornos informáticos muy similares que suelen confundirse entre sí: *Internet*, *intranet* y *extranet*.

4.3.1. Internet

En 1998, la Internet tenía más de 100 millones de usuarios en todo el mundo, en diciembre de 2000 unos 400 millones, en junio de 2002 unos 600 millones y el número sigue creciendo rápidamente. Más de 100 países están conectados a este nuevo medio para intercambiar todo tipo de información.

Al contrario que otros servicios *online*, que se controlan de forma centralizada, la Internet posee un diseño descentralizado. Cada ordenador (*host*) en la Internet es independiente. Sus operadores pueden elegir que servicio de Internet usar y que servicios locales quieren proporcionar al resto de la Internet. Asombrosamente, este diseño anárquico funciona satisfactoriamente.

Existe una gran variedad de formas de acceder a la Internet. El método más común es obtener acceso a través de Proveedores de servicios de Internet (*Internet Service Provider* (**ISP**)).

Cuando se emplea la palabra internet en minúsculas, nos referimos a un conjunto de dos o más redes de ordenadores interconectadas entre sí.

[6] Mediante **HTTP** 1.1, que permite mantener conexiones abiertas (*Keep-alive connections*), se puede emplear la misma conexión para transmitir varios ficheros.

4.3.2. Intranet

Una intranet es una red de ordenadores basada en los protocolos que gobiernan Internet (**TCP/IP**) que pertenece a una organización y que es accesible únicamente por los miembros de la organización, empleados u otras personas con autorización.

Una intranet puede estar o no conectada a Internet. Un sitio web en una intranet es y actúa como cualquier otro sitio web, pero los cortafuegos (*firewall*) lo protegen de accesos no autorizados (su acceso está limitado a un ámbito local).

Al igual que Internet, las intranets se usan para distribuir y compartir información. Las intrantes hoy en día componen el segmento con mayor crecimiento dentro de Internet, porque son menos caras de montar y de administrar que las redes privadas que se basan en protocolos propietarios.

4.3.3. Extranet

Una extranet es una intranet a la que pueden acceder parcialmente personas autorizadas ajenas a la organización o empresa propietaria de la intranet.

Mientras que una intranet reside detrás de un cortafuego y sólo es accesible por las personas que forman parte de la organización propietaria de la intranet, una extranet proporciona diferentes niveles de acceso a personas que se encuentran en el exterior de la organización. Esos usuarios pueden acceder a la extranet sólo si poseen un nombre de usuario y una contraseña con los que identificarse. La identidad del usuario determina que partes de la extranet puede visualizar. Además, para acceder a una extranet se suelen emplear medios de comunicación seguros, como *Secure Socket Layer* (**SSL**) y *Virtual Private Network* (**VPN**).

Las extranets se están convirtiendo en un medio muy usado por empresas que colaboran para compartir información entre ellas. Se emplean como medio de comunicación de una empresa con sus clientes, proveedores o socios. Las extranets son la base del comercio electrónico entre empresas (*business to business*, B2B).

4.4. Ventajas y desventajas

El desarrollo explosivo de Internet y en especial de la **WWW** se debe a la aceptación por todo el mundo de los estándares y tecnologías que emplea: medio de transporte común (**TCP/IP**), servidor (**HTTP**) y lenguaje de creación de páginas (**HTML**) estandarizados.

Muchas empresas han descubierto que las anteriores tecnologías se pueden emplear en las aplicaciones cliente/servidor que emplean. De esta forma nace el concepto de intranet: usar las tecnologías de Internet para implementar las tradicionales aplicaciones cliente/servidor dentro de una empresa. Además, una vez que se tiene una aplicación que funciona en una intranet, aparece la posibilidad de permitir su uso a

través de Internet, lo que facilita el teletrabajo o la movilidad de los empleados de una empresa[7].

Una ventaja clave del uso de aplicaciones web es que el problema de gestionar el código en el cliente se reduce drásticamente. Suponiendo que existe un navegador o explorador estándar en cada cliente, todos los cambios, tanto de interfaz como de funcionalidad, que se deseen realizar a la aplicación se realizan cambiando el código que resida en el servidor web. Compárese esto con el coste de tener que actualizar uno por uno el código en cada uno de los clientes (imaginemos que tenemos 2.000 ordenadores clientes). No sólo se ahorra tiempo porque reducimos la actualización a una sólo máquina, sino que no hay que desplazarse de un puesto de trabajo a otro (la empresa puede tener una distribución geográfica amplia).

Una segunda ventaja, relacionada con la anterior, es que se evita la gestión de versiones. Se evitan problemas de inconsistencia en las actualizaciones, ya que no existen clientes con distintas versiones de la aplicación.

Una tercera ventaja es que si la empresa ya está usando Internet, no se necesita comprar ni instalar herramientas adicionales para los clientes.

Otra ventaja, es que de cara al usuario, los servidores externos (Internet) e internos (intranet) aparecen integrados, lo que facilita el aprendizaje y uso.

Una última ventaja, pero no menos importante, es la independencia de plataforma. Para que una aplicación web se pueda ejecutar en distintas plataformas (*hardware* y sistema operativo), sólo se necesita disponer de un navegador para cada una de las plataformas, y no es necesario adaptar el código de la aplicación a cada una de ellas. Además, las aplicaciones web ofrecen una interfaz gráfica de usuario independiente de la plataforma (ya que la plataforma de ejecución es el propio navegador).

Una desventaja, que sin embargo está desapareciendo rápidamente, es que la programación en la web no es tan versátil o potente como la tradicional. El lenguaje **HTML** presenta varias limitaciones, como es el escaso repertorio de controles disponibles para crear formularios. Por otro lado, al principio las aplicaciones web eran básicamente de "solo lectura": permitían una interacción con el usuario prácticamente nula. Sin embargo, con la aparición de nuevas tecnologías de desarrollo como *Java*, *JavaScript* y **ASP**, esta limitación tiende a desaparecer.

4.5. Arquitecturas de las aplicaciones web

Las aplicaciones web se basan en una arquitectura cliente/servidor: por un lado está el cliente (el navegador, explorador o visualizador) y por otro lado el servidor (el servidor web). Existen diversas variantes de la arquitectura básica según como se implementen las diferentes funcionalidades de la parte servidor. Las arquitecturas más comunes son:

[7] Pensemos, por ejemplo, en los ejecutivos que tienen que desplazarse entre distintos países, pero que necesitan acceder a las aplicaciones de su empresa.

1. Todo en un servidor (Figura 4.3): un único ordenador aloja el servicio de **HTTP**, la lógica de negocio y la lógica de datos y los datos. El software que ofrece el servicio de **HTTP** gestiona también la lógica de negocio. Las tecnologías que emplean esta arquitectura son **ASP** y PHP.

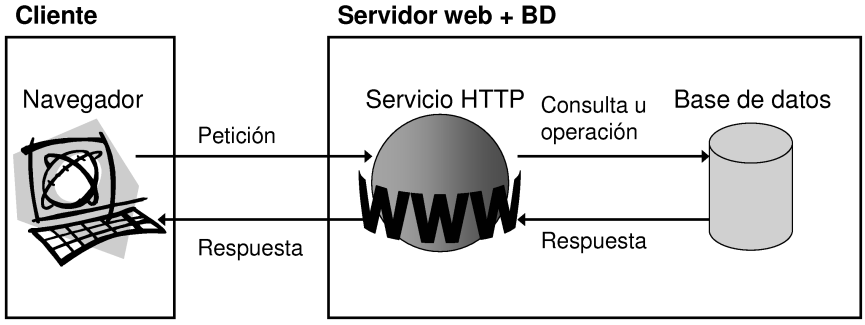


Figura 4.3: Arquitectura de las aplicaciones web: todo en un servidor

2. Servidor de datos separado (Figura 4.4): a partir de la arquitectura anterior, se separa la lógica de datos y los datos a un servidor de bases de datos específico. Las tecnologías que emplean esta arquitectura son **ASP** y PHP.

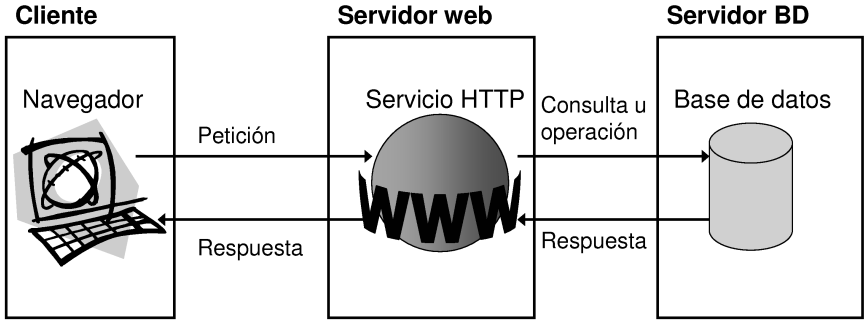


Figura 4.4: Arquitectura de las aplicaciones web: separación servidor de datos

3. Todo en un servidor, con servicio de aplicaciones (Figura 4.5): en la arquitectura número 1 se se separa la lógica de negocio del servicio de **HTTP** y se incluye el

servicio de aplicaciones para gestionar los procesos que implementan la lógica de negocio. La tecnología que emplea esta arquitectura es **JSP**.

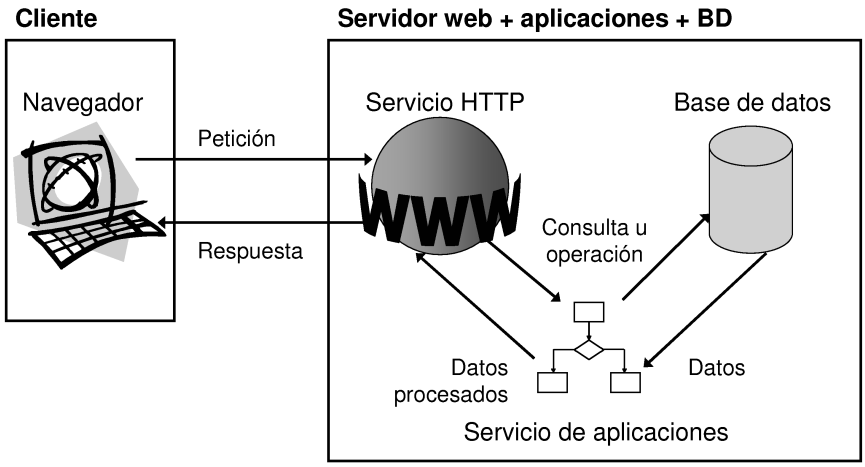


Figura 4.5: Arquitectura de las aplicaciones web: todo en un servidor, con servicio de aplicaciones

4. Servidor de datos separado, con servicio de aplicaciones (Figura 4.6): a partir de la arquitectura anterior, se separa la lógica de datos y los datos a un servidor de bases de datos específico. La tecnología que emplea esta arquitectura es **JSP**.

5. Todo separado (Figura 4.7): las tres funcionalidades básicas del servidor web se separan en tres servidores específicos. La tecnología que emplea esta arquitectura es **JSP**.

El objetivo de separar las distintas funcionalidades (servicio de **HTTP**, lógica de negocio y lógica de datos) en distintos servidores es aumentar la escalabilidad del sistema de cara a obtener un mayor rendimiento. Al separar las distintas funcionales en distintos servidores, cada uno de ellos se puede configurar (dimensionar) de forma adecuada a los requisitos que presenta cada uno de ellos. Por ejemplo, para ofrecer el servicio de **HTTP** hace falta un ordenador con una buena conexión a Internet, rápido pero sin grandes necesidades de almacenamiento. Sin embargo, para el servidor de bases de datos hace falta un ordenador con mucha memoria y con un disco duro de alta capacidad de almacenamiento y rápido para mantener todos los datos.

Otra ventaja que se obtiene al separar las funcionalidades, es que al aislar la lógica de negocio y la lógica de datos en servidores separados que no están conectados

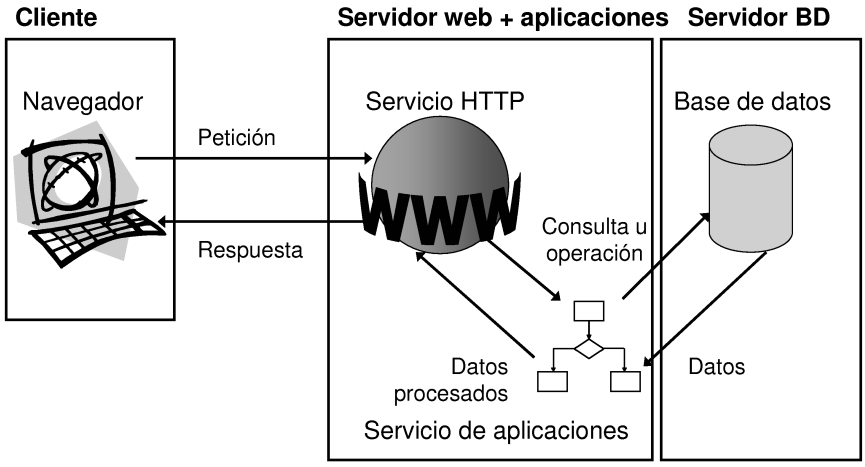


Figura 4.6: Arquitectura de las aplicaciones web: separación servidor de datos, con servicio de aplicaciones

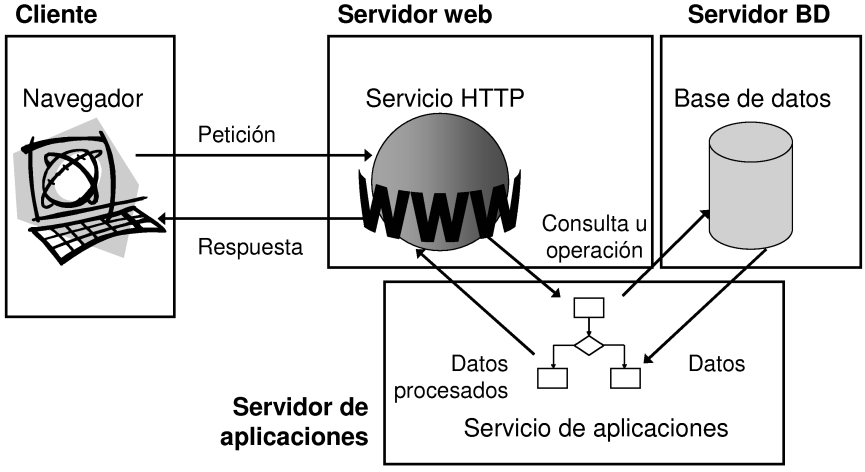


Figura 4.7: Arquitectura de las aplicaciones web: todo separado

directamente a Internet se aumenta el nivel de seguridad, ya que no es tan fácil acceder a ellos.

En algunos casos, las arquitecturas donde se separan el servicio de **HTTP** del resto de servicios es la única opción disponible. Por ejemplo, en los sistemas heredados (*legacy systems*), donde ya existe una lógica de negocio en un servidor de aplicaciones y una lógica de datos en un servidor de bases de datos, la única forma de acceder desde Internet al sistema heredado es a través de un servidor que ofrezca el servicio de **HTTP** y se comunique internamente con el servidor de aplicaciones.

En la Figura 4.8 se presenta una comparativa de las tecnologías de generación de páginas web más comunes: **ASP**, PHP, **JSP** y *Caché Server Pages* (**CSP**).

Las tecnologías **ASP** y PHP se clasifican dentro de la categoría *Web Server Scripting*. Estas tecnologías pueden emplear las arquitecturas 1 y 2. En estas tecnologías, el servidor web ofrece el servicio de **HTTP** y además se encarga de ejecutar los procesos que poseen la lógica de negocio. La comunicación entre la lógica de negocio y los datos se suele realizar a través de métodos estandarizados como **ODBC**, lo que introduce una penalización en el tiempo de ejecución. En este tipo de tecnologías, parte de la lógica de negocio se puede desplazar hacia el servidor de bases de datos al emplear procedimientos almacenados (*stored procedures*) y disparadores (*triggers*). De este modo se puede lograr una cierta escalabilidad en el sistema.

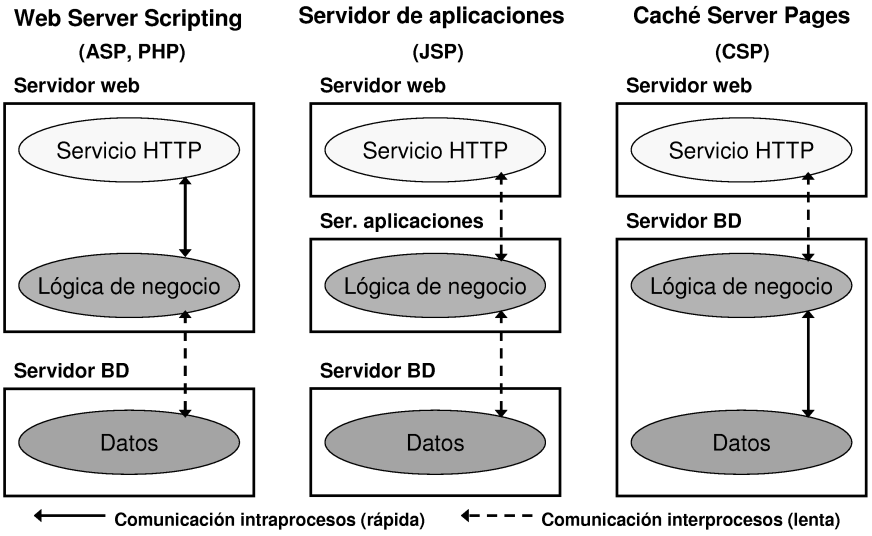


Figura 4.8: Arquitectura de las aplicaciones web: todo separado

Las tecnologías que emplean servidor de aplicaciones como **JSP**, ofrecen la máxima flexibilidad, ya que se pueden separar completamente el servicio **HTTP**, la lógica de

negocio y la lógica de datos. Estas tecnologías pueden emplear las arquitecturas 3, 4 y 5.

Por último, en la tecnología **CSP**, la lógica de negocio y la lógica de datos residen en el sistema gestor de bases de datos (es el encargado de ejecutar los procesos que implementan la lógica de negocio). Gracias a ello, la comunicación entre los procesos de la lógica de negocio y los datos es muy rápida, ya entre ambos se puede establecer una comunicación interproceso.

4.6. Metodología de desarrollo de sitios web

No existe en la actualidad una metodología de desarrollo de sitios web ampliamente aceptada. Sin embargo, una posible metodología es la que se presenta a continuación. Algunas de las fases de esta metodología se pueden realizar en paralelo o no acabar hasta el final del desarrollo del sitio web:

1. Se estudian los requisitos y especificaciones del sitio web: cuál es el contenido del sitio web, qué se pretende conseguir, a quién se destina y número de visitas previsto, qué inversión se desea realizar, de cuánto tiempo se dispone, etc.

2. A partir de los requisitos se decide la arquitectura y tecnología del sitio web: empleo de un servidor web propio o alojamiento (hospedaje) en un servidor alquilado, ancho de banda de la comunicación del servidor web con Internet, páginas estáticas o tecnología de generación dinámica de páginas (**ASP**, **CGI**, etc.), datos almacenados en ficheros o en un servidor de bases de datos, etc.

3. A continuación se diseña la estructura lógica o de navegación del sitio web: página inicial, página principal, empleo de marcos, los menús, división en secciones, relación entre las distintas secciones, página de novedades, etc.

4. Se define la estructura física, que puede ser igual a la lógica o totalmente independiente.

5. Se crean los contenidos del sitio web. Si se emplea una base de datos, se realiza la carga de datos.

6. Se realiza el diseño gráfico y ergonómico: colores, montaje, tipografía, botones de navegación, logotipos y demás elementos gráficos, etc.

7. Se crean las páginas estáticas y los elementos multimedia.

8. Desarrollo de los *scripts* y páginas dinámicas.

9. Por último, se verifica el correcto funcionamiento del sitio web: se comprueba la conexión con la base de datos, se verifica que no existan enlaces rotos, se confirma que todos los recursos empleados (imágenes, ficheros con código de script,

etc.) se encuentran en el sitio web y en su lugar correspondiente, etc. Además, se comprueba el sitio web con distintos navegadores para asegurar su compatibilidad. También se realizan pruebas de carga para evaluar el rendimiento.

10. Puesta en marcha.

Capítulo 5

Estructura de un sitio web

En este capítulo se introducen los conceptos de sitio web, estructura física y estructura lógica (o de navegación). Además se incluye una guía de estilo con consejos a tener en cuenta cuando se diseñe la navegación de un sitio web. Una de las principales premisas que se tienen que tener en cuenta cuando se diseñan la estructura física y la estructura lógica es lograr sitios web que sean fáciles de mantener y de navegar.

Índice General

5.1. Qué es un sitio web

Un sitio web es un conjunto de páginas web relacionadas entre sí. Se entiende por página web tanto el fichero que contiene el código **HTML** como todos los recursos que se emplean en la página (imágenes, sonidos, código *JavaScript*, etc.).

En todo sitio web se suelen distinguir dos páginas especiales: la página inicial (o página de entrada) y la página principal (o página menú). La página inicial, conocida como *splash page* en inglés, es la primera página que un usuario ve al visitar un sitio web. Normalmente, la página inicial se emplea para promocionar la compañía u organización a la que pertenece el sitio web, o para dar a conocer un producto o servicio particular (por ejemplo, para promocionar unos productos en oferta). También se suele emplear para informar al usuario de los requisitos (tipo y versión de navegador, resolución mínima, etc.) necesarios para visualizar correctamente el resto de páginas del sitio web. A menudo, la página inicial es la más vistosa del sitio web, ya que tiene el objetivo de atraer y "atrapar" al visitante. La mayoría de las páginas iniciales poseen las siguientes características:

- Poco texto, pero muchas imágenes, gráficos animados, sonidos o incluso vídeos.
- Algunas pasan ("saltan") automáticamente a la página principal, pero en otras el usuario tiene que pulsar en un enlace para cargar la página principal.
- En algunos casos la página inicial se convierte en un "tunel de entrada": una presentación que dura bastante tiempo (más de 15 segundos), que suele estar realizada con múltiples páginas o con una sola página que emplea tecnología multimedia (como Macromedia Flash). En estos casos, suele existir un enlace para evitar el tunel de entrada y "saltar" directamente a la página principal.

Además de usarse como "tarjeta de presentación", la página inicial también se puede emplear para disminuir el tiempo necesario para cargar las páginas posteriores. Mientras el usuario está visualizando la página inicial, las imágenes que se emplearán en las siguientes páginas se pueden cargar en la memoria caché del navegador mediante un proceso en segundo plano del que el usuario no es consciente.

En algunos sitios web se prescinde de la página inicial y directamente se muestra al usuario la página principal.

Por otro lado, la página principal, conocida como *home page*, *root page*, *entry page*, *front page* o *main page* en inglés, es la página que funciona como índice o tabla de contenidos del sitio web. A través de esta página, el resto de documentos del sitio web es accesible de una forma directa o indirecta. Por tanto, la página principal tiene la función de guiar y dirigir al usuario a otras páginas del sitio web. La página principal tiene que ser clara y no crear confusión con infinidad de opciones (si las opciones son abundantes significa que la clasificación de la información no ha sido la correcta).

5.2. Contenido de un sitio web

Posiblemente, el aspecto más importante (y al mismo tiempo uno de los más descuidados) de un sitio web sea su contenido (y por extensión, sus servicios): un sitio web con un buen diseño pero con un pobre contenido es poco probable que triunfe, sin embargo, un sitio web con un pobre diseño pero con un buen contenido sí que puede triunfar (por ejemplo, los buscadores Yahoo o Google o la tienda Amazon no destacan por su diseño, pero han triunfando en Internet gracias a su contenido o a los servicios que ofrecen).

El contenido de un sitio web se suele organizar en una serie de secciones que facilitan su búsqueda y localización. En general, los contenidos se pueden clasificar en dos tipos: comunes y específicos.

Los contenidos comunes son aquellos que se pueden encontrar en la mayoría de los sitios web pertenecientes a una misma categoría. Por ejemplo:

- Grandes compañías:
  - ¿Quienes somos?
  - Información de contacto.
  - Historia de la compañía.
  - Objetivos.
  - Cartera de clientes.
  - Productos o servicios.
  - ...
- Periódicos:
  - Editorial.
  - Noticias internacionales.
  - Noticias nacionales.
  - Deportes.
  - ...
- Centros educativos:
  - Personal.
  - Profesores.
  - Estudiantes.
  - Planes de estudio.
  - ...

- Portal genérico:
  - Noticias.
  - Canales.
  - Correo.
  - Chat.
  - ...

Por otro lado, los contenidos específicos son aquellos que incorpora cada sitio web como propios y que no tienen porqué encontrarse en otros sitios web de la misma categoría. Por ejemplo, un periódico digital puede decidir ofrecer a sus lectores una cuenta de correo electrónico gratuita, un servicio que no se suele encontrar normalmente en los periódicos digitales, pero que sí que es común en los portales.

En general, lo más adecuado es mostrar primero la información más general y luego, si así lo solicita el usuario, facilitar información más detallada.

El contenido de un sitio web se tiene que cuidar mucho. Supongamos una empresa que vende productos. En el mundo "real", el cliente puede probar el producto que desea comprar: por ejemplo, si se desea comprar un televisor, se puede encender, escuchar su calidad de sonido y ver su imagen, juguetear con sus funciones y en definitiva comprobar el grado de satisfacción que se obtiene. Además, el cliente tiene un trato directo con el personal de la empresa. Sin embargo, en el mundo "virtual" de Internet todo eso es imposible: el usuario no puede saber si le satisface un producto hasta que lo haya recibido. Por ello, la decisión de comprar un producto se basa también en criterios que son ajenos al propio producto, como puede ser la imagen que proyecta la empresa, el aspecto de la página web, o las recomendaciones de otras personas que hayan quedado satisfechas con el producto. Este último aspecto es muy importante en Internet: la fidelización del cliente. Gracias a la personalización de un sitio web, se puede adecuar el sitio web al perfil de cada cliente.

Un mismo contenido se puede clasificar en distintas categorías. Por ejemplo en el caso de un sitio web que sea un buscador de servicios, ¿un cibercafé se clasificaría en restaurantes o en entretenimiento? ¿En cafeterías o en tiendas de informática? En caso de duda, lo mejor es incluir la misma información en las distintas categorías posibles y no exclusivamente en una sola. Hay que tener en cuenta que la gente suele emplear distintos nombres para el mismo concepto y un mismo concepto lo pueden clasificar en distintas categorías.

5.3. Estructura física

La estructura física de un sitio web es la forma en que se almacenan los distintos recursos (ficheros) que forman un sitio web en el sistema de archivos del servidor web. ¿Se almacenan todos los ficheros en un único directorio (carpeta)? ¿O están

almacenados según el tipo de fichero en distintos directorios? Al definir la estructura física de un sitio web hay que tener en cuenta las preguntas anteriores.

Una estructura física eficiente reduce los costes de mantenimiento: cada vez que se tenga que actualizar el sitio web, el tiempo necesario para localizar el recurso que se desea actualizar se reducirá.

La estructura física se debe de planificar antes de iniciar el desarrollo del sitio web. Modificar la estructura física una vez que se han creado muchas páginas es costoso y propenso a cometer errores, ya que para ello hay que actualizar los enlaces entre las páginas, las referencias a las imágenes, etc., para que reflejen la nueva estructura del sitio web.

Si el sitio web contiene un número pequeño de ficheros (5 o 10 ficheros), quizás no suponga ningún problema tenerlos todos juntos en el mismo directorio. Pero conforme aumente el número de ficheros, si se ordenan en subcategorías se simplificará su localización y su cambio.

No existe una forma perfecta de organizar físicamente un sitio web. Todo depende de la persona que tenga que mantener el sitio web: una estructura física puede tener sentido para una persona, pero no para otra. Lo importante es que en el futuro, cuando haya que realizar cambios en un sitio web, la persona que tenga que hacerlo pueda encontrar lo que quiere modificar rápidamente.

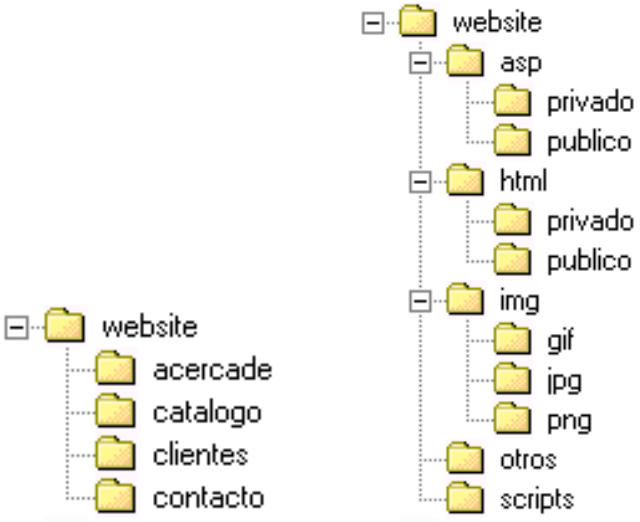
Existen diferentes alternativas de organización de un sitio web en directorios. Algunas de las más usuales son:

- Por el tipo de fichero. Por ejemplo, ficheros **HTML**, ficheros gráficos, ficheros de vídeos, etc.
- Por el nivel de acceso (visibilidad). Por ejemplo, parte pública, parte privada de los clientes, parte privada de los administradores, etc.
- Por el contenido. Por ejemplo, una empresa que vende productos puede organizar los ficheros según la información que contienen: los ficheros de la familia de productos A en un directorio, los ficheros de la familia de productos B en otro directorio, etc.
- Por la fecha. Por ejemplo, el sitio web de una revista puede organizar los ficheros según la fecha de publicación.
- Por su propietario. Por ejemplo, una empresa puede organizar los ficheros según el departamento al que pertenecen: departamento A, departamento B, etc.
- Por su estructura lógica o de navegación. La estructura física puede ser una copia de la estructura lógica.

Por ejemplo, en la Figura 5.1 (a), el sitio web se ha organizado físicamente en función de su contenido: información sobre la empresa ("acercade"), el catálogo, los clientes e información de contacto. En la Figura 5.1 (b), el sitio web se ha organizado

en función del tipo de fichero (asp, html, etc.) en un primer nivel y en función del nivel de acceso (privado y público) y del tipo de fichero (gif, jpg y png) en un segundo nivel.

En la Figura 5.2 (a), el sitio web se ha organizado en función de su contenido en un primer nivel y en función del tipo de fichero en un segundo nivel. En la Figura 5.2 (b), el sitio web se ha organizado en función del nivel de acceso en un primer nivel y en función del tipo de fichero en un segundo nivel.



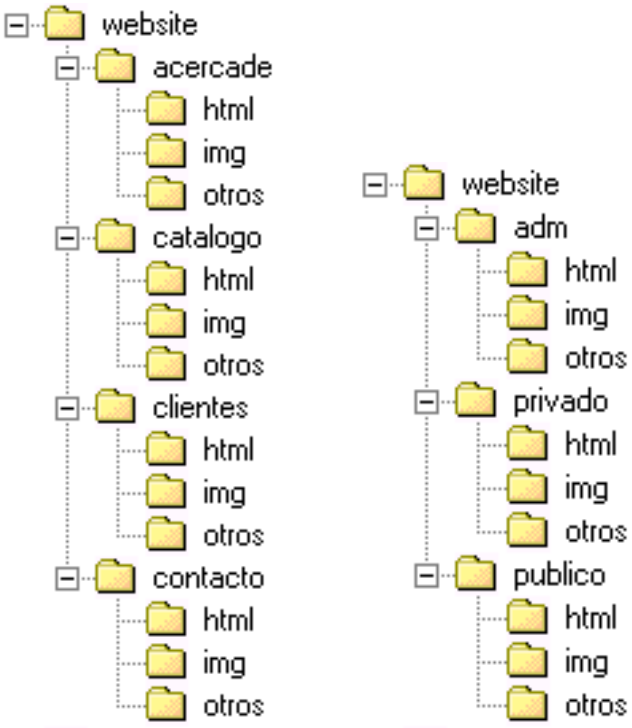
(a) (b)

Figura 5.1: Distintos tipos de estructuras físicas

Sin embargo, no hay que abusar de los directorios y crear estructuras de directorios muy profundas, ya que cada directorio que se crea añade unas letras más a la **URL**: tanto el creador del sitio web como el usuario final tendrán problemas a la hora de recordar una **URL**.

5.3.1. Nombres de los directorios y de los ficheros

El nombre de los directorios y de los ficheros es un aspecto que también hay que tener en cuenta, ya que es importante tanto para el desarrollador del sitio web como para los futuros visitantes (usuarios). Por un lado, se tienen que elegir nombres intuitivos, que expresen el contenido o la función de un fichero, ya que así el desarrollador cuando tenga que mantener el sitio web no tendrá que pararse a pensar que contiene cada directorio o fichero.



(a) (b)

Figura 5.2: Distintos tipos de estructuras físicas

Por otro lado, también es importante para la gente que visite el sitio web. ¿Por qué? Aunque los nombres de los directorios y de los ficheros sólo se van a ver en la **URL** y se puede creer que nadie se fija en ella, a veces los usuarios toman nota de ella (la copian por escrito) o incluso la emplean para navegar libremente.

Por ejemplo, imaginemos que un usuario ha realizado una búsqueda en un buscador y ha llegado a la siguiente página de un sitio web de venta de electrodomésticos: `http://www.electro-abc.es/electronica/videos/sonic1430.html`.

En esta página se muestran las características del vídeo Sonic 1430. El usuario puede ser que modifique la **URL** y escriba:
`http://www.electro-abc.es/electronica/videos/`
con la esperanza de obtener una página con una lista de todos los vídeos, o que incluso escriba:
`http://www.electro-abc.es/electronica/`
para obtener una lista completa de todos los dispositivos electrónicos.

Si la **URL** no es intuitiva, estaremos creando una barrera entre los visitantes y el contenido que desean obtener, y tales barreras se tienen que evitar siempre que se pueda.

Por otro lado, hay que tener mucho cuidado con los caracteres que se emplean en los nombres de los directorios y de los ficheros, ya que cada sistema operativo permite un conjunto de caracteres distintos. Nunca se deben usar espacios en blanco o caracteres especiales como `&`, `%`, `#`, etc. Además, sólo se debe usar el punto para la extensión de los ficheros.

Un problema muy común con los sistemas operativos Microsoft Windows es que no distinguen mayúsculas y minúsculas, aunque sí que permiten escribir los nombres de los ficheros mezclando mayúsculas y minúsculas. Es decir, el nombre de un fichero puede ser `algo.txt` o `algoMas.TXT`, pero en un mismo directorio no pueden existir los ficheros `algo.txt`, `Algo.txt` o `ALGO.TXT`, ya que para el sistema operativo representan el mismo nombre y, por tanto, el mismo fichero. Esta característica suele originar problemas al mover un sitio web de un sistema operativo Microsoft Windows a Unix. Por ejemplo, si en un enlace se escribe el nombre de un fichero en mayúsculas, pero realmente el nombre del fichero está en minúsculas, en el servidor web con Microsoft Windows el enlace funciona correctamente, pero en el que tiene Unix no, ya que en este último sistema operativo sí que se distinguen completamente las mayúsculas y las minúsculas. Por tanto, es conveniente ser consistente y escribir siempre los nombres de los directorios y de los ficheros en minúsculas (lo más adecuado) o en mayúsculas, pero nunca mezclar las dos formas.

5.3.2. Enlaces

Los enlaces[1] son un elemento clave en la web, ya que son los que permiten crear el hipertexto. Los enlaces se pueden clasificar de distintas formas. Atendiendo al ámbito de referencia, se pueden dividir en enlaces que hacen referencia a recursos del propio sitio web y enlaces que hacen referencia a recursos de otro sitio web. Además, los primeros también se pueden clasificar en enlaces dentro del propio documento (intradocumentales) y enlaces a otro documento (extradocumentales).

Por otro lado, los enlaces también se pueden clasificar en absolutos y relativos. Cuando se crea un enlace desde un sitio web a otro sitio web, siempre se emplea un enlace absoluto de la forma:
`http://www.otrositio.es/dir/recurso.xxx`

Sin embargo, cuando se crea un enlace a un recurso en el mismo sitio web, se puede hacer absoluto (dos tipos) o relativo:

- Absoluto 1: similiar al enlace a otro sitio web. Se indica el nombre del sitio actual y la ruta completa al recurso. Todos los enlaces de este tipo comienzan por `http://`. Por ejemplo:
  `http://www.sitioactual.es/dir/recurso.xxx`

- Absoluto 2: se indica la ruta completa al recurso. Como no se indica el nombre del sitio web, se emplea el actual. Todos los enlaces de este tipo comienzan por `/`. Por ejemplo:
  `/dir/recurso.xxx`

- Relativo: no se indica la ruta completa, sino la posición relativa del recurso respecto al fichero que contiene el enlace. Por ejemplo, si un enlace en la página `a.html` del directorio `dir` queremos que apunte a la página `b.html` del mismo directorio, simplemente escribiremos:
  `b.html`

Los enlaces relativos no se limitan a recursos que se encuentren en el mismo directorio. Por ejemplo, si un enlace en la página `a.html` del directorio `dir-a` queremos que apunte a la página `b.html` del directorio `dir-b`, y ambos directorios se encuentran en el mismo directorio padre, escribiremos:
`../dir-b/b.html`

Muy importante: en las direcciones de Internet (la **URL**), la barra que se emplea en la rutas para separar directorios es / y no \. Hay que tener cuidado de no confundir la barra que se emplea en las rutas en los sistemas operativos Microsoft Windows con la barra que se emplea en los sistemas Unix y en la **URL**.

Dos son las ventajas que se obtienen al emplear los enlaces relativos. Por un lado, hay que escribir menos en los enlaces (las **URL** son más cortas). Por otro lado, se

[1] Enlace se refiere a cualquier referencia a un fichero desde una página, ya sea mediante la etiqueta `<A> ... </A>`, `<SCRIPT> ... </SCRIPT>`, `<IMG>`, etc.

facilita el transporte del sitio web entre distintos servidores o directorios: los enlaces siguen funcionando.

En la Figura 5.3 se resumen los distintos tipos de enlace que se han presentado.

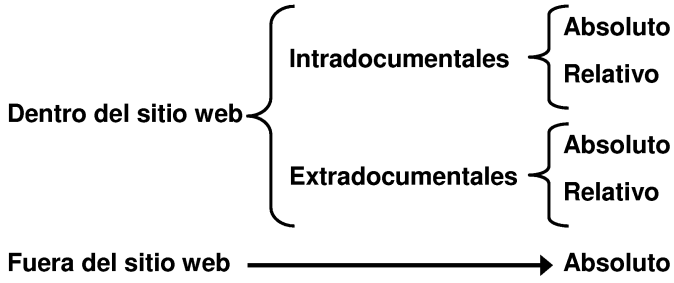


Figura 5.3: Tipos de enlaces

5.4. Estructura lógica

La estructura lógica o de navegación define como un visitante se va a mover (navegar) de una zona a otra de un sitio web. La estructura lógica y la física son totalmente independientes, aunque se pueden definir de forma que una sea una proyección de la otra. Según el tipo de navegación que se permita en un sitio web, los usuarios tendrán una sensación de "poco libertad" (navegación controlada) o "mucha libertad" (navegación libre).

La estructura lógica también se tiene que planear con cuidado y antes de comenzar a desarrollar el sitio web, con el fin de asegurarse de que todo el mundo podrá navegar por el sitio fácilmente. Para logralo, es conveniente dar respuesta a una serie de preguntas:

- ¿Cuál es el propósito de este sitio web?
- ¿Qué contendrá?
- ¿Cuál es la audiencia a la que está destinado?
- ¿Qué esperamos obtener?
- ¿Qué esperamos que los visitantes obtengan?

Por ejemplo, un sitio web destinado a proporcionar noticias (el sitio web de un periódico) tendrá una apariencia y una navegación distintas de las de un sitio web destinado a vender productos.

Existen distintos tipos de estructuras lógicas de navegación. Las más comunes son la secuencial, en rejilla, en árbol y en red, aunque lo normal es que en un sitio web se de una combinación de todas ellas (estructura mixta).

5.4.1. Estructura secuencial

La estructura secuencial (*sequence structure*) es la más simple de todas. En ella, el usuario comienza en la página principal y sólo puede navegar en una dirección (hacia adelante o hacia atrás). En la Figura 5.4 se puede ver una representación simplificada de esta estructura.

Esta estructura se suele emplear en los siguientes casos:

- Un curso o tutorial.
- Una visita (*tour*) guiada.
- Un asistente (*wizard*) que explica como se realiza algo.
- Un proceso determinado compuesto de una serie de pasos secuenciales, como por ejemplo una compra o el registro de un usuario en un servicio de correo electrónico gratuito.

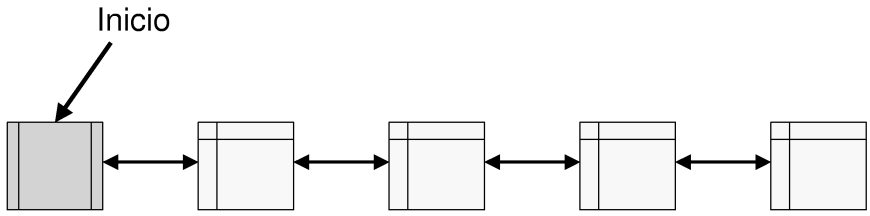


Figura 5.4: Estructura secuencial

Un claro ejemplo de estructura secuencial se puede encontrar en el sitio web de *Yahoo! España*[2]. Este sitio web posee una guía para nuevos usuarios llamada **Tutorial de Yahoo! entrada y registro** donde se explica el proceso de obtención de un identificador para emplear los servicios de YAHOO!. Esta guía presenta una estructura secuencial, ya que el usuario sólo la puede visualizar en el orden secuencial establecido por su creador. Por ejemplo, en la Figura 5.5 podemos ver la primera página web donde se presenta la guía y aparece un enlace en la esquina superior derecha (**¿Cómo obtengo una ID?**). Al pulsar sobre ese enlace, se pasa a la segunda página que se

[2] `http://www.yahoo.es`.

muestra en la Figura 5.6; en esta página existe un enlace para pasar a la siguiente página (**Condiciones de servicio**) y otro enlace para volver a la anterior (**Regístrate en Yahoo!**). En la Figura 5.7 y la Figura 5.8 se muestran las dos páginas siguiente: se puede apreciar como las páginas presentan la misma estructura (un enlace para volver a la página anterior y otro para pasar a la siguiente), pero además incorporan un enlace para volver al principio de la guía (**Inicio del Tour**). Por tanto, no se trata de una estructura secuencial pura, ya que existe la posibilidad de volver directamente a la página inicial.

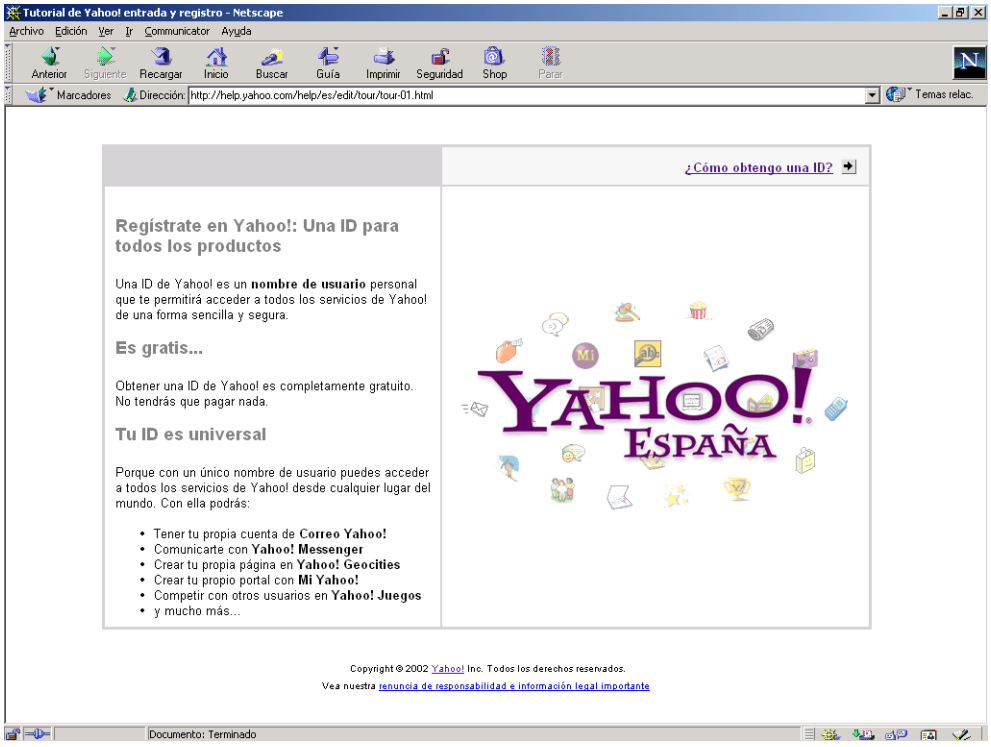


Figura 5.5: Ejemplo de estructura secuencial

5.4.2. Estructura en rejilla

La estructura en rejilla (*grid structure*) se emplea cuando existen estructuras secuenciales paralelas. Casos típicos de esta estructura son los sitios web escritos en varios idiomas, de forma que la misma página se puede consultar en otros idiomas, o los sitios web que ofrecen cada página en distintos formatos (por ejemplo, formato de

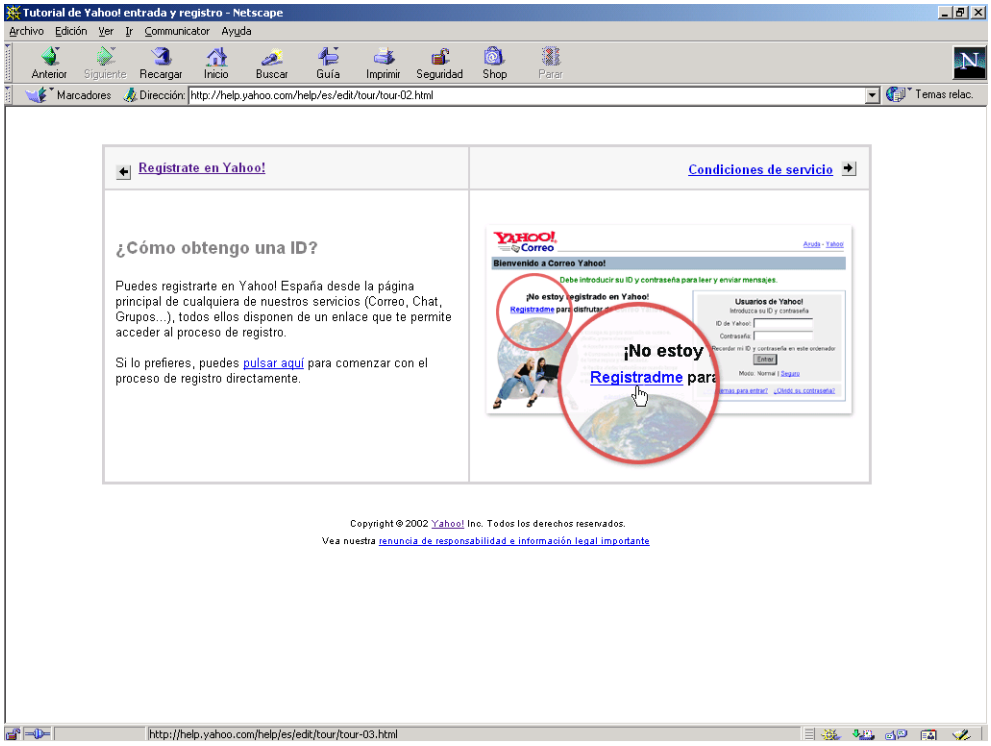


Figura 5.6: Ejemplo de estructura secuencial

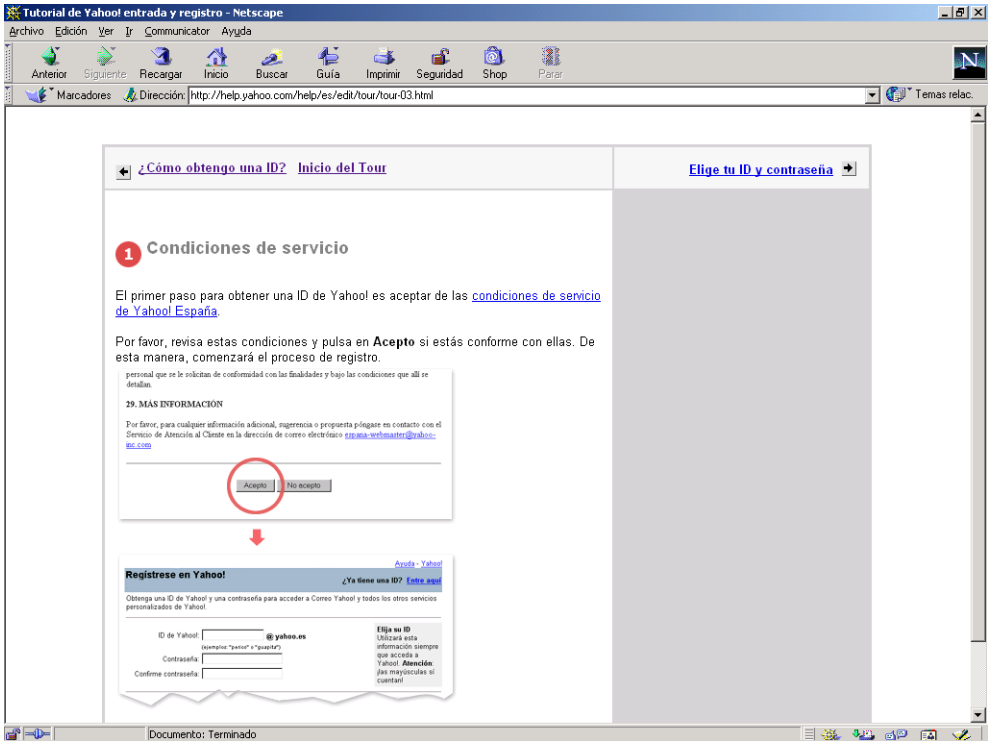


Figura 5.7: Ejemplo de estructura secuencial

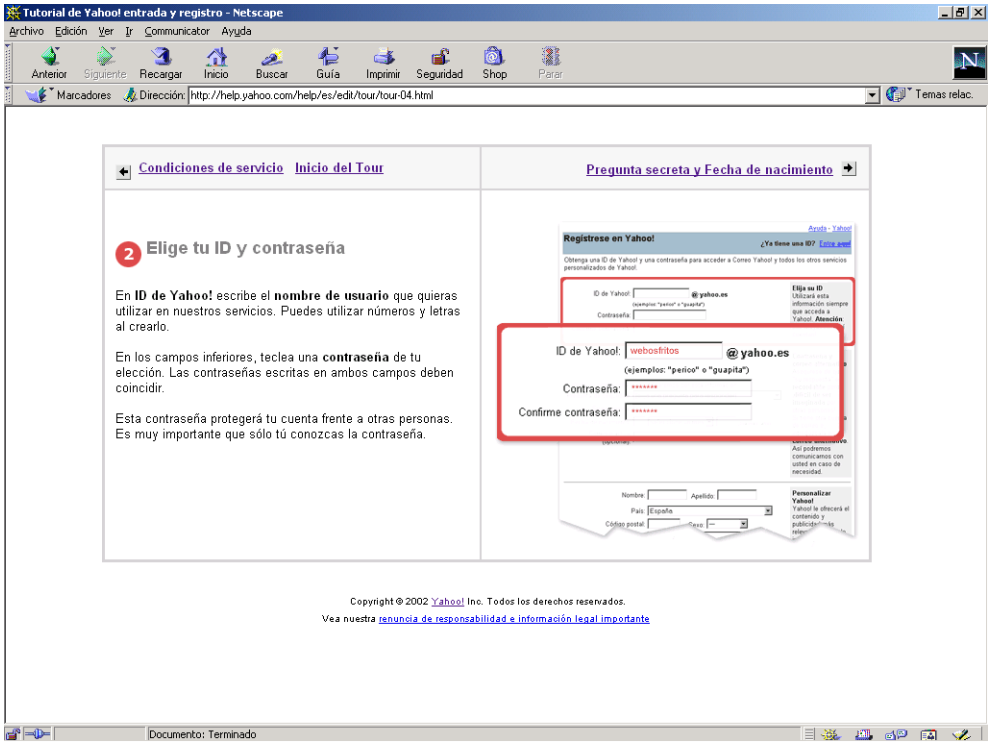


Figura 5.8: Ejemplo de estructura secuencial

pantalla, formato de impresión, formato para discapacitados, etc.). En la Figura 5.9 se muestra un esquema de esta estructura.

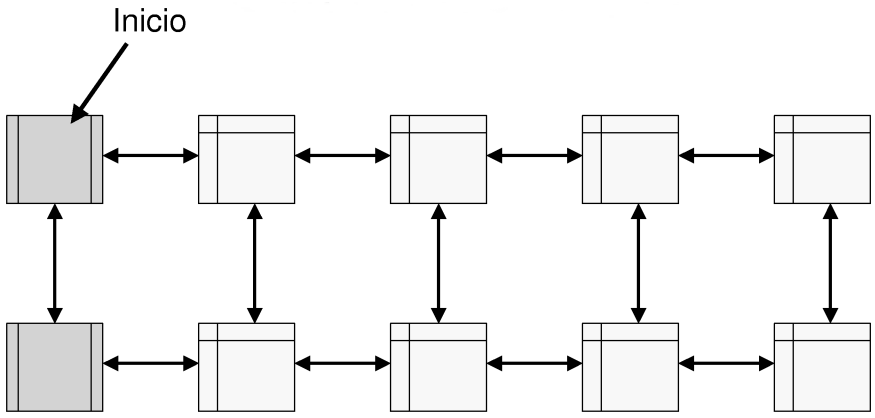


Figura 5.9: Estructura en rejilla

Un claro ejemplo de esta estructura se puede encontrar en la revista *Consumer*[3]. En ella, toda página web posee varias versiones: la normal (Figura 5.10), la preparada para imprimir (Figura 5.11) y la adaptada a gente discapacitada (Figura 5.12). Además, también existe la misma página en castellano, gallego, euskera, catalán y valenciano.

5.4.3. Estructura en árbol

La estructura en árbol (*tree structure*) es una de las más comunes, ya que permite estructurar el contenido de una forma jerárquica con distintos niveles, lo que facilita la búsqueda de información y la navegación (Figura 5.13).

Sin embargo, este tipo de estructuras pueden presentar un problema: el árbol puede degenerar a estructuras poco profundas y muy anchas, o muy profundas y poco anchas (Figura 5.14).

5.4.4. Estructura en red

La estructura en red (*web structure*) es la típica de Internet. En este tipo de estructura no existe un orden establecido, de forma que cada página puede estar enlazada con todas las páginas que componen el sitio web. Además, en este tipo de

[3] `http://revista.consumer.es`.

Figura 5.10: Ejemplo de estructura en rejilla: versión normal

Figura 5.11: Ejemplo de estructura en rejilla: versión para imprimir

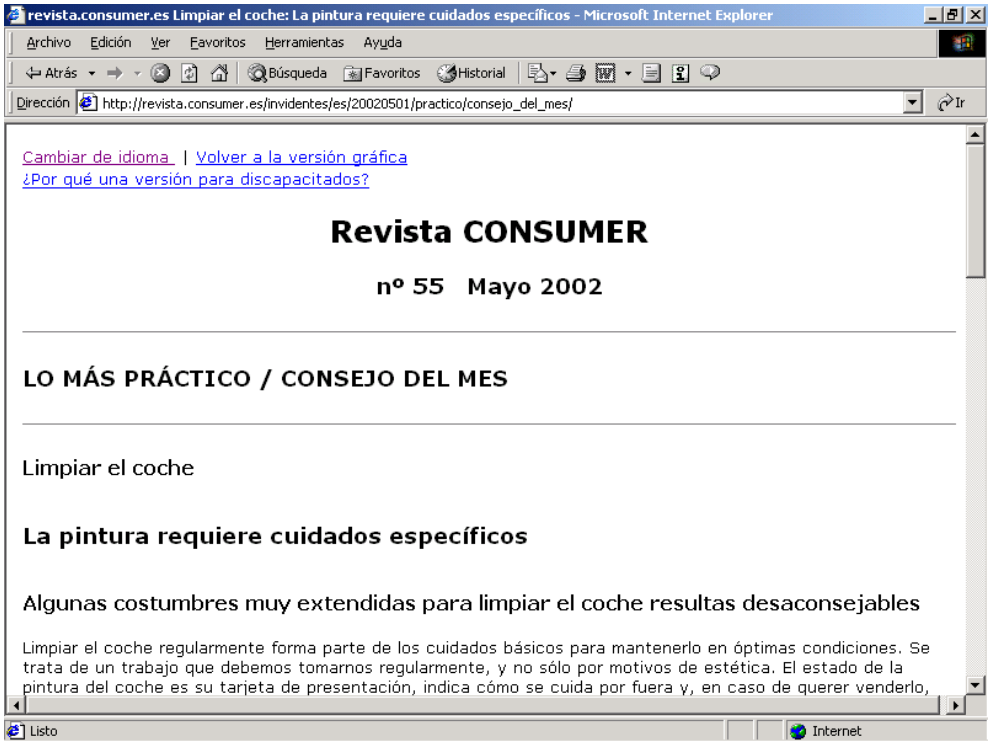


Figura 5.12: Ejemplo de estructura en rejilla: versión para discapacitados

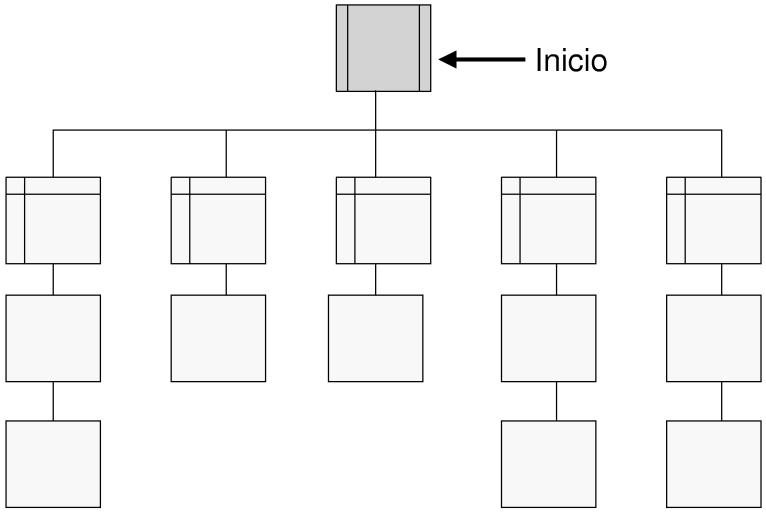


Figura 5.13: Estructura en árbol

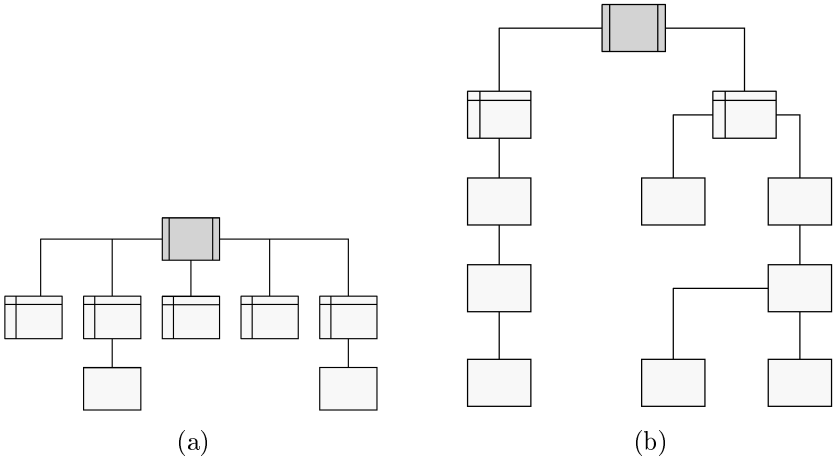


Figura 5.14: Problemas en las estructuras en árbol

estructuras, el sitio web suele tener varias páginas de inicio, tal como se observa en la Figura 5.15.

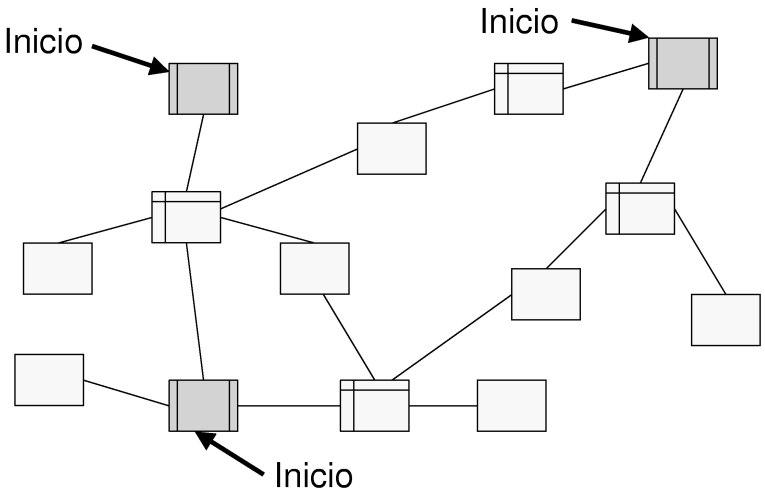


Figura 5.15: Estructura en red

5.4.5. Estructura mixta

En realidad, las cuatro estructuras que se han comentado son ideales: en el mundo real suelen aparecer mezcladas entre sí, tal como se muestra en la Figura 5.16. Por ello, lo más común es que la estructura de un sitio web sea mixta.

5.4.6. Comparativa

Las cuatro estructuras básicas (secuencial, en rejilla, en árbol y en red) se pueden comparar en función de distintas características. En la Figura 5.17 se muestran la cuatro estructuras clasificadas en función del poder de expresividad y de la capacidad de navegación que presentan. Esta comparación es cualitativa y no cuantitativa (por ejemplo, no se puede afirmar que la estructura en árbol posea un 40 % más de expresividad respecto a la estructura secuencial).

La expresividad se mueve desde un nivel bajo en el cual la información contenida en un sitio web se percibe de igual forma por todos los visitantes, a un nivel alto en el cual cada visitante puede percibir la misma información de distintas formas.

La capacidad de navegación se mueve desde un nivel bajo con poca libertad donde los movimientos del visitante son predecibles, a un nivel alto con mucha libertad

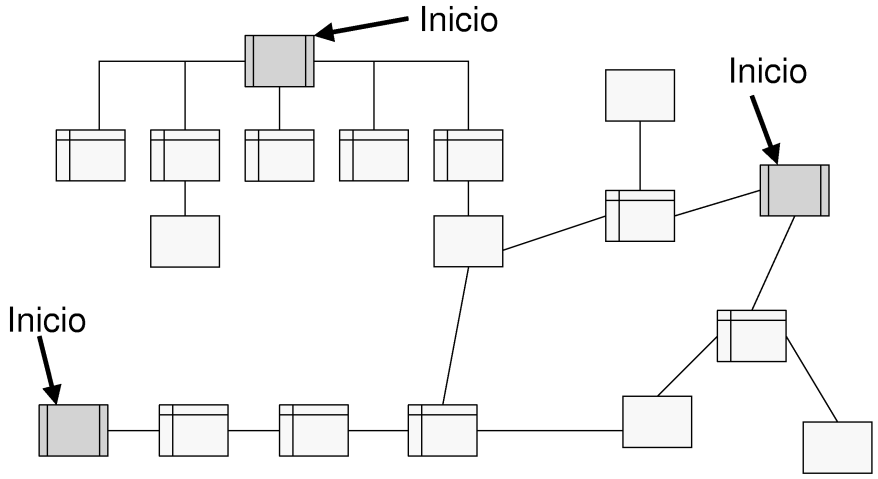


Figura 5.16: Estructura mixta

donde la navegación es impredecible, el visitante se puede perder fácilmente o llegar a páginas a las que no nos interese que llegue (riesgo alto de navegación confusa).

5.4.7. Cómo no perderse en la estructura

Un visitante de un sitio web se puede perder fácilmente en las páginas que lo forman. Por tanto, es recomendable emplear algún mecanismo que indique al usuario dónde está, de dónde ha venido y a dónde puede ir a continuación.

Existen distintos esquemas que proporcionan esta información durante la navegación. Dos de los más conocidos son el "rastro de las migas de pan"[4] (*breadcrumb trail*) y la numeración de los pasos.

En el esquema del "rastro de las migas de pan", se muestra una lista con enlaces de vuelta a cada una de las páginas por las que el usuario ha pasado hasta llegar a la actual. Es decir, se muestra un "rastro" para que el usuario pueda volver hacia atrás hacia el punto de entrada en el sitio web. Por ejemplo, en la Figura 5.18 se muestra el siguiente rastro:

`Inicio > Internet y ordenadores > Formatos > HTML > HTML Dinámico`

Una variante de este esquema consiste en no tener en cuenta las páginas por donde ha pasado el usuario, sino mostrar el rastro de las páginas por el nivel actual en la jerarquía de navegación que conducen a la página principal. Así se consigue mostrar el contexto de la página actual respecto al conjunto de páginas del sitio web. Cada

[4] Por el cuento de "Hansel y Gretel" de los hermanos Grimm.

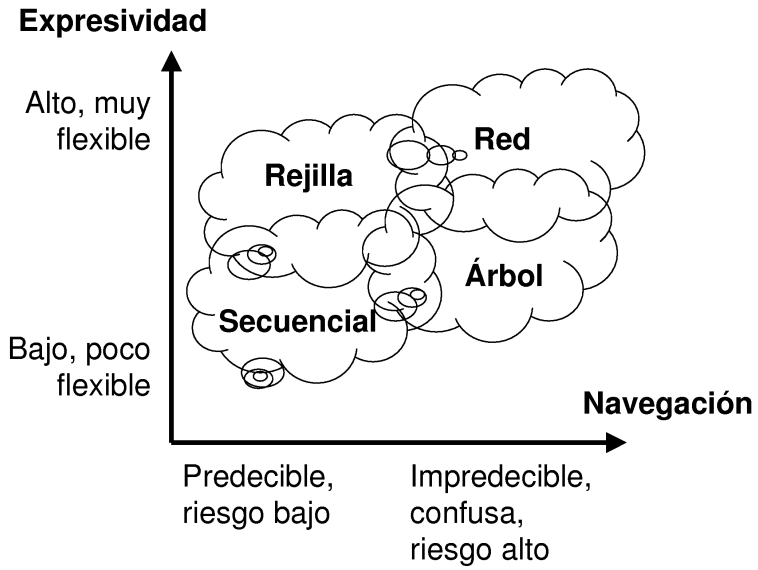


Figura 5.17: Comparación de las cuatro estructuras lógicas o de navegación básicas

nivel de la lista es una enlace a la página principal de dicho nivel. Este esquema es el adecuado para aquellos sitios web donde la información se estructure de una forma jerárquica.

Por otro lado, en el esquema de numeración de los pasos, se muestra una lista con todos los pasos necesarios para completar un proceso (por ejemplo, comprar un producto o registrarse en un servicio de correo electrónico). Además, se muestra en todo momento el paso en el que se encuentra el usuario. De este modo, el usuario sabe cuanto queda para terminar el proceso. Esta característica contrasta con los procesos en los que no se indica nada y el usuario no sabe cuantas veces tiene que pulsar el botón `Siguiente` antes de terminar. Por ejemplo, en la Figura 5.19 se puede ver el primer paso (de cuatro) para obtener una cuenta de correo electrónico en `mail.com`. En este paso el usuario tiene que elegir qué dirección de correo electrónico desea (nombre de usuario y dominio). En la parte superior de la página se muestra que el proceso se compone de cuatro pasos y que en la actualidad se encuentra en el primer paso. En la Figura 5.20 se muestra el segundo paso: el usuario tiene que introducir sus datos personales. En el tercer paso, que se muestra en la Figura 5.21, el usuario selecciona las listas de distribución de correo electrónico a las que se desea subscribir. Por último, la Figura 5.22 muestra el último paso que es la selección del tipo de servicio de correo electrónico (gratuito o de pago).

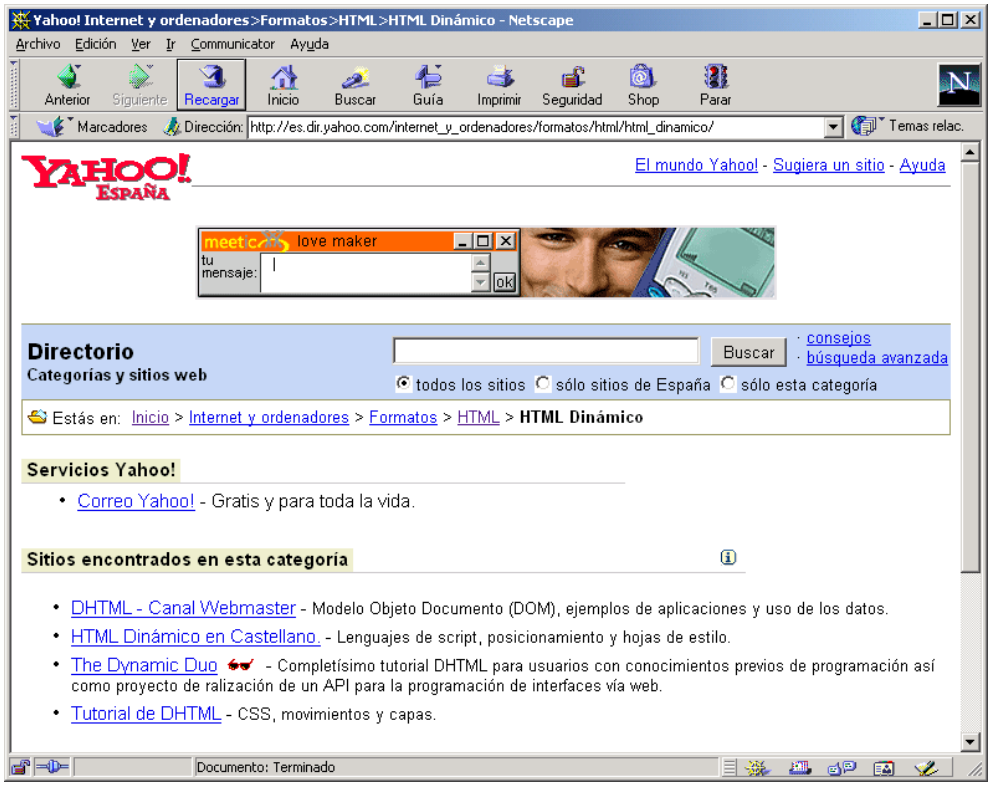


Figura 5.18: Ejemplo de "rastro de las migas de pan"

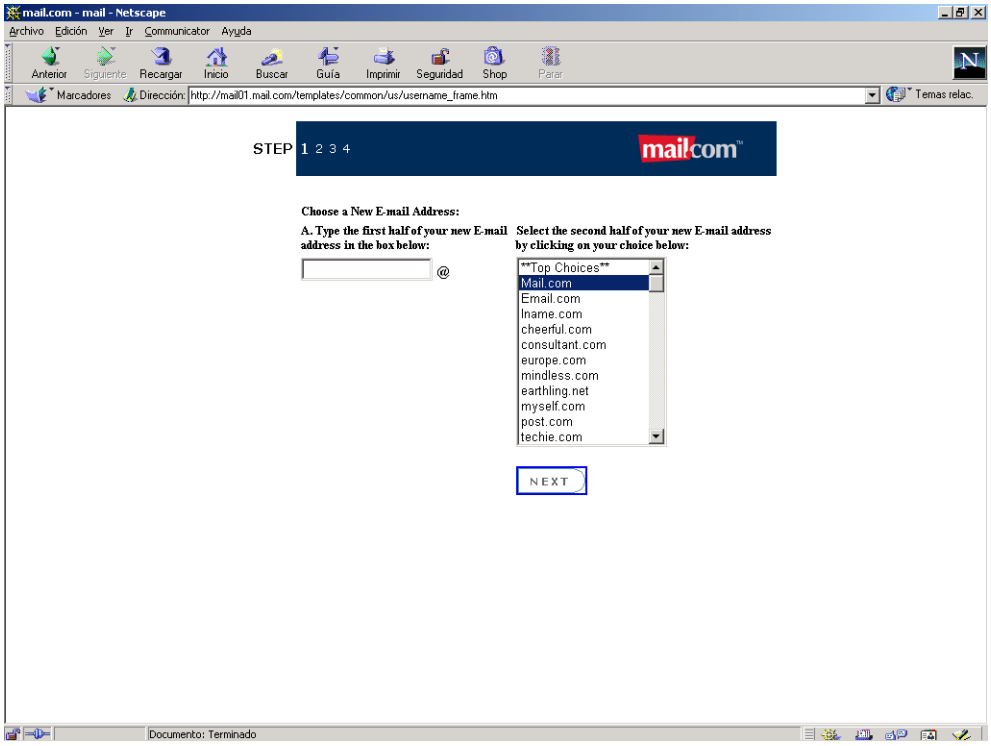


Figura 5.19: Ejemplo de esquema de numeración de los pasos: paso 1

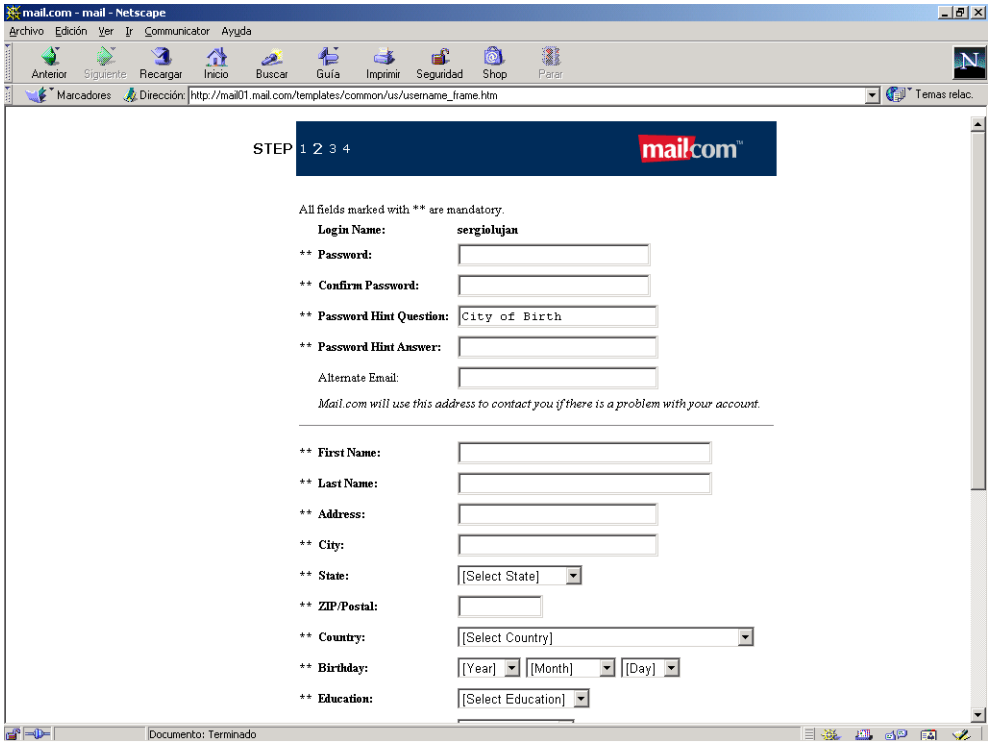


Figura 5.20: Ejemplo de esquema de numeración de los pasos: paso 2

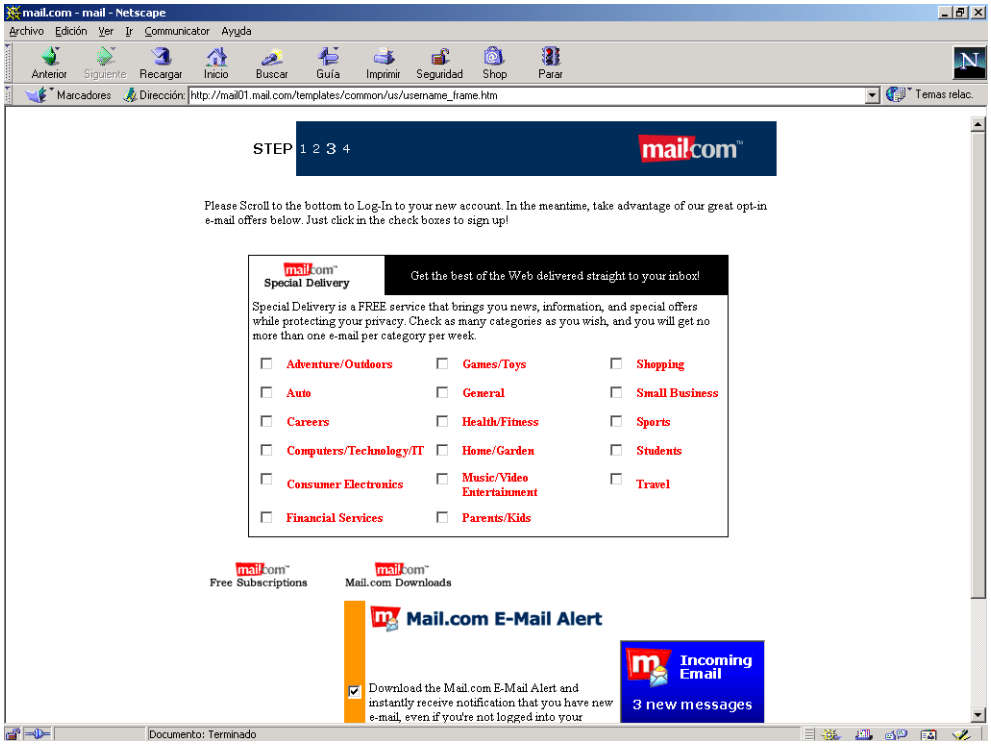


Figura 5.21: Ejemplo de esquema de numeración de los pasos: paso 3

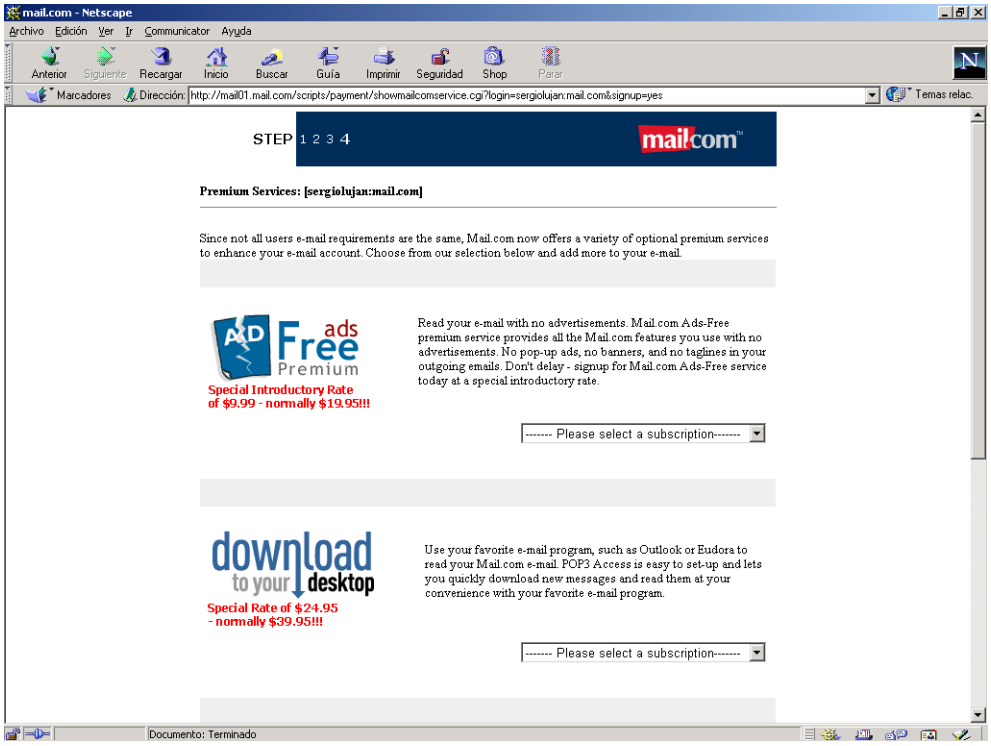


Figura 5.22: Ejemplo de esquema de numeración de los pasos: paso 4

5.5. Guía de estilo

A continuación se incluyen algunos consejos que pueden ayudar a obtener una estructura lógica correcta:

- Regla de las tres pulsaciones ("tres clicks"): la media de pulsaciones para acceder a cualquier información útil del sitio web debería de ser tres pulsaciones. Los usuarios buscan rapidez y efectividad.
- Hay que evitar los "callejones sin salida": páginas que no poseen enlaces para continuar la navegación.
- Para evitar el problema anterior, es conveniente que todas las páginas posean un enlace a la página principal y a la página anterior.
- Otra solución es crear una barra de navegación o menú del sitio web con las categorías principales en que se ha organizado la información.
- Los visitantes deberían saber en todo momento dónde se encuentran y cómo volver a la página principal.
- La opción de búsqueda, los índices y el mapa del sitio web facilitan a los visitantes la localización de la información que buscan. Existen herramientas que automáticamente gestionan estos recursos.
- Debería de estar claramente indicada en todo momento la información de contacto, tanto de la empresa como de la persona encargada del mantenimiento del sitio web.
- La memoria a corto plazo (*short term memory*) del ser humano tiene un límite (tanto de tiempo como de espacio). Miller estableció en 1956[5] la capacidad de esta memoria en 7 ± 2 elementos (*chunks*). Por ello, hay que evitar las estructuras muy profundas, para que el usuario no pierda la orientación (no sepa de dónde viene y por dónde ha pasado).
- Todas las páginas deben de poseer un elemento (un logotipo, el color de fondo, etc.) que las identifique como pertenecientes a un sitio web. De este modo, el usuario podrá saber fácilmente cuando ha abandonado el sitio web.

[5] George A. Miller. The magical number seven, plus of minus two: Some limits on our capacity for processing information. *Psychological Review*, 63, páginas 81-97, marzo 1956.

Capítulo 6

HTML

HTML es un lenguaje de marcas (etiquetas) que se emplea para dar formato a los documentos que se quieren publicar en la WWW. Los navegadores son capaces de interpretar las etiquetas y mostrar los documentos con el formato deseado. En este capítulo se presentan los conceptos básicos y avanzados (enlaces, tablas, marcos, etc.) de HTML. Además, se realiza un estudio especial de los formularios, ya que son una pieza clave de las aplicaciones web.

Índice General

6.1. Introducción

Las páginas web o páginas **HTML** son unos ficheros escritos en el lenguaje **HTML**. El desarrollo de estas páginas abarca un amplio grupo de tecnologías, desde las páginas más sencillas que sólo usan el lenguaje **HTML** hasta las más complejas que usan **DHTML**, **CSS**, *JavaScript*, *applets* realizados en *Java* o componentes *ActiveX*.

El lenguaje **HTML** se basa en **SGML**, un sistema mucho más completo y complicado de procesamiento de documentos que indica como organizar y marcar (etiquetar) un documento. **HTML** define e interpreta las etiquetas de acuerdo a **SGML**.

Las páginas **HTML** se pueden diseñar usando texto con distintos tipos de letras o colores, imágenes, listas de elementos, tablas, etc. Su modo de empleo es muy sencillo: se basa en el uso de etiquetas que indican que elementos contiene cada página, el formato que hay que aplicar a cada uno de ellos y como se tienen que distribuir por la página.

6.2. Evolución de HTML

El nacimiento de **HTML** va ligado al de la **WWW**. Los orígenes de ambos se sitúan en 1991, en los trabajos que llevaba a cabo Tim Berners-Lee y sus compañeros en el **CERN** en Suiza. Uno de los primeros artículos en los que muestran sus ideas es "World-Wide Web: The Information Universe"[1]. En este artículo detallan un sistema que permita realizar el sueño de "interconectar todo el conocimiento de la humanidad y facilitar su acceso a todo el mundo gracias al empleo de los ordenadores". Para lograrlo, hacen uso de tecnologías como el hipertexto o la hipermedia, tecnologías que ya existían desde hacía varios años (en contra de lo que la gente cree, estas tecnologías no fueron inventadas por ellos).

Entre las distintas partes que componen el sistema que proponen, se incluye "una sintaxis en el estilo de SGML" para proporcionar formato a los documentos. A partir de ahí nace **HTML** como un lenguaje para el intercambio de documentos científicos y técnicos. **HTML** evita la complejidad de **SGML** al definir un pequeño conjunto de etiquetas que simplifican la estructura de los documentos y las reglas no son tan estrictas como en **SGML**.

En octubre de 1994, Tim Berners-Lee, funda el **W3C** en el *Massachusetts Institute of Technology, Laboratory for Computer Science [MIT/LCS]* en colaboración con el **CERN**, y con el apoyo de *Defense Advanced Research Projects Agency* (**DARPA**) y de la Comisión Europea. En abril de 1995, el INRIA[2] se convierte en el primer *host* europeo de **W3C**.

[1] Tim Berners-Lee, Robert Cailliau, Jean-François Groff, Bernd Pollermann. World-Wide Web: The Information Universe. *Electronic Networking: Research, Applications and Policy*, Vol. 1, No. 2, Meckler, Westport CT, primavera 1992.

[2] *Institut National de Recherche en Informatique et Automatique.*

El objetivo principal del **W3C** es encabezar el desarrollo de la **WWW**, mediante la elaboración de protocolos que aseguren su estandarización. Hoy en día, **W3C** lidera el desarrollo de distintas tecnologías, como **HTML**, **HTTP**, *Extensible Markup Language* (**XML**), *Portable Network Graphics* (**PNG**), etc. En el Cuadro 6.1 mostramos las distintas versiones de **HTML** que se han estandarizado[3] desde 1995.

Fecha	Versión
Noviembre 1995	HTML 2.0
Enero 1997	HTML 3.2
Diciembre 1997	HTML 4.0
Diciembre 1999	HTML 4.01
Enero 2000	XHTML 1.0
Agosto 2002	XHTML 1.0 Second Edition
Agosto 2002	XHTML 2.0 (Working Draft)

Cuadro 6.1: Versiones de HTML

En enero de 2000 aparece *Extensible HyperText Markup Language* (**XHTML**) 1.0, el futuro sustituto de **HTML**. Como dice el propio estándar, se trata de "una reformulación de HTML en XML 1.0". **XHTML** es el lenguaje **HTML** escrito según las normas que impone **XML**. Por tanto, es una aplicación concreta de **XML** y no deben confundirse entre sí. Las principales diferencias entre **HTML** y **XHTML** 1.0 son:

- Las etiquetas y atributos tienen que escribirse en minúsculas.
- Los valores de los atributos tienen que ir entre comillas.
- No se admiten atributos sin valor.
- Todas las etiquetas tienen que aparecer por parejas (inicio y fin) o como etiquetas vacías.

6.3. Clasificación de las páginas

Según como se generan las páginas web en el servidor, se clasifican en:

[3] Algunas versiones, como **HTML**+ o **HTML** 3.0 nunca llegaron a estándar.

- **Estáticas**. Poseen un contenido fijo, todos los usuarios que las consultan reciben la misma información. El usuario recibe en su navegador la página del servidor sin un procesamiento previo[4].

- **Dinámicas o activas en el servidor**: poseen un contenido variable, distintos usuarios al consultar la misma página pueden recibir distintos contenidos. El usuario recibe en su navegador la página después de haber sido procesada en el servidor. Para lograrlo se emplean lenguajes de programación. Ejemplo: páginas generadas por un **CGI**, páginas **ASP**, etc.

Por otro lado, según como se visualizan las páginas en el cliente, se clasifican en:

- **Estáticas**. Cuando no poseen ningún tipo de código de *script*, *applets* o *plug-ins*. Ejemplo: sólo código **HTML**.

- **Dinámicas o activas en el cliente**. Cuando se interpreta o ejecuta código en el equipo del usuario. Para lograrlo se emplean lenguajes de programación y objetos integrados. Ejemplo: páginas con *JavaScript*, **DHTML**, *applets*, etc.

Las características anteriores se pueden combinar como se quieran: una página puede ser estática en el servidor y en el cliente, estática en el servidor pero dinámica en el cliente, dinámica en el servidor y estática en el cliente y, por último, dinámica en el servidor y dinámica en el cliente.

Si nos planteamos crear páginas dinámicas en el servidor o en el cliente, surgen una serie de ventajas y desventajas. Las ventajas principales de las páginas activas en el cliente son:

- Se descarga de trabajo al servidor, ya que la ejecución del código se realiza de forma distribuida en cada cliente.

- Se reduce el ancho de banda necesario, ya que se evitan continuos traspasos de información del servidor al cliente y viceversa.

- Ofrecen respuestas inmediatas al usuario.

Sin embargo, existen algunas razones que nos llevan a crear páginas activas en el servidor:

- Cuando la información sobre la que se realiza el procesamiento es muy amplia, su envío al cliente puede suponer un consumo de ancho de banda grande.

[4]Las páginas estáticas también pueden mostrar datos procedentes de una base de datos mediante la técnica *snap shot*. La información de la base de datos se convierte en **HTML** de forma manual, automáticamente cuando ocurre un suceso o a una fecha y hora dadas (por ejemplo, todos los días a las tres de la mañana). Esta técnica es adecuada para catálogos, listas de precios, directorios telefónicos, etc., que no se modifican muy a menudo.

- Puede existir información restringida sobre la que se realice el procesamiento y que no interese que pueda llegar íntegra al cliente.
- Las páginas activas en el cliente se basan en tecnologías dependientes del navegador y del sistema operativo del usuario. Presuponer que el usuario dispone de los requisitos necesarios para que funcionen las páginas es un error.
- Las páginas activas en el cliente pueden ser poco seguras.

Entonces, ¿cuál de las cuatro combinaciones posibles elegir? La mejor opción es la última: páginas dinámicas tanto en el servidor como en el cliente, ya que podremos aprovechar las ventajas de las dos opciones y eliminar sus desventajas. Simplemente, habrá que distribuir la lógica de nuestra aplicación entre el servidor y el cliente, de forma que se obtenga la solución más óptima a nuestro problema.

En este capítulo nos vamos a centrar en el lenguaje **HTML**, que nos permite realizar páginas estáticas en el cliente. En el Capítulo 9 veremos *JavaScript*, que nos permitirá realizar páginas dinámicas en el cliente. Ambas tecnologías no influyen en la parte del servidor. El desarrollo de páginas activas en el servidor queda fuera de los objetivos de este libro.

6.4. Qué necesito para usar HTML

No es necesario un servidor web, un proveedor web o tener una conexión a Internet para empezar a escribir documentos **HTML**. Los documentos **HTML** tienen un formato de texto plano (*American Standard Code for Information Interchange* (**ASCII**)), por lo que todo lo que se necesita es un editor (como el Bloc de notas de Microsoft Windows) para crear las páginas y un navegador (como Microsoft Internet Explorer) para verlas. Podemos crear, vincular y probar documentos **HTML** completos en nuestro ordenador, aunque no esté conectado a ninguna red.

Para facilitar la creación de páginas **HTML**, han aparecido gran cantidad de programas. Básicamente, se pueden dividir en dos grupos: los editores de **HTML** y los programas de diseño **HTML**.

La mayoría de editores que ayudan a escribir **HTML** son simples editores de texto con algunos botones que insertan en un documento las etiquetas más comunes. Otros, suelen incluir la característica *syntax highlight*: significa que el editor es capaz de comprender el lenguaje en el que se programa, y colorea las palabras diferenciándolas según sean etiquetas, atributos, comentarios, etc. Otros más avanzados ofrecen la opción de completar las etiquetas o muestran una ventana de ayuda con los atributos que posee cada etiqueta.

Por otro lado, los programas de diseño muestran la página **HTML** de forma gráfica y en tiempo real: es posible desplazar los distintos elementos que la componen (tablas, imágenes, texto), modificar sus propiedades (tamaño, color, tipo de letra) y crear efectos avanzados. Son programas del tipo **WYSIWYG**, ya que se ve en todo

momento la página tal como se visualizará en un navegador. Un inconveniente de estos programas es que generan mucho código **HTML**: si en el futuro se desea modificar la página directamente a través del código **HTML**, es prácticamente imposible. Entre los mejores programas de esta clase destacan Adobe Golive, Claris Home Page, Macromedia DreamWeaver y Microsoft FrontPage.

Conviene aclarar desde un principio que lo único que da formato a una página web es una etiqueta **HTML**. Por ejemplo, si editamos con cuidado un archivo de texto con objeto de tener párrafos y columnas de cifras formateadas, pero no se incluye ninguna etiqueta, al visualizarlo en **HTML** todo el texto fluirá en un solo párrafo y se perderá todo el formato que le hayamos aplicado.

La extensión de un archivo **HTML** suele ser `.html` o `.htm`[5]. Se deben emplear nombres cortos y sencillos. Hay que evitar el uso de espacios o de caracteres especiales en el nombre del archivo y también controlar el uso de mayúsculas y minúsculas puesto que en Internet existen multitud de sistemas operativos y alguno puede ser que no acepte los mismos nombres de archivo que acepta el nuestro. Por ejemplo, hay sistemas operativos en los que las mayúsculas y minúsculas se distinguen (Unix) y otros donde no (Microsoft Windows[6]).

6.5. Conceptos básicos de HTML

El lenguaje **HTML** consta de una serie de etiquetas o marcas (*tags*). La mayoría de las etiquetas aparecen por parejas (códigos pareados), siendo una de inicio (apertura) y otra de fin[7] (cierre): delimitan la parte del documento **HTML** que se ve afectada por su acción. Pero también hay etiquetas que aparecen de forma individual, como `<IMG>` para insertar una imagen. Este último tipo de etiquetas tiene efecto en el lugar en el que se incluyen o desde el lugar en que aparecen hasta el final de la página.

Todas las etiquetas comienzan con el símbolo < (menor que) y terminan con el símbolo > (mayor que). Entre estos dos símbolos aparece el nombre de la etiqueta. Por ejemplo, `<HR>` es una etiqueta válida, pero `<HR` o `<HR<` no lo son.

En el lenguaje **HTML** no se distinguen minúsculas/mayúsculas. Por tanto, las cadenas `<HTML>`, `<html>` y `<Html>` representan la misma etiqueta. De todos modos, es una buena práctica ponerlas siempre en mayúsculas, sobre todo en aras de una mayor

[5] Esta extensión existe debido a que en DOS y Microsoft Windows 3.x los ficheros sólo pueden tener una extensión de tres caracteres.

[6] Aunque los nombres de los archivos pueden tener mayúsculas y minúsculas, para acceder a un archivo no se tienen en cuenta. Por ello, en un mismo directorio no pueden existir dos archivos que sólo se diferencian porque algunos caracteres aparecen en mayúsculas en uno y en minúsculas en el otro.

[7] Los navegadores actuales son muy flexibles: si falta alguna etiqueta de fin no producen un error y muestran la página lo mejor posible. De todas formas, es recomendable ajustarse siempre a la sintaxis y no cometer errores, ya que así se logrará la máxima compatibilidad.

legibilidad y claridad de los documentos **HTML**[8].

Las etiquetas de fin tienen el mismo nombre que las de inicio, pero van precedidas del símbolo / (barra inclinada). Por ejemplo, la etiqueta de cierre correspondiente a `<HTML>` es `</HTML>`.

Una etiqueta puede poseer varios atributos a los que hay que asignar valor. Algunos de estos atributos son obligatorios, mientras que otros suelen ser opcionales. Por otro lado, algunos aceptan un valor de cualquier tipo, otros cualquier valor de un tipo concreto (alfanumérico, numérico, etc.) y, por último, algunos necesitan un valor concreto de un conjunto de valores. Los atributos se escriben dentro de la etiqueta y separados por espacios en blanco. Para asignar un valor a un atributo se emplea el signo igual (=). El valor que se asigna a un atributo tiene que ir encerrado entre comillas. Por ejemplo, la etiqueta `<IMG>`, que no tiene una etiqueta de cierre, tiene varios atributos, entre ellos `SRC` que espera cualquier carácter valido en una **URL**, `WIDTH` y `HEIGHT` que esperan un valor numérico y el atributo `ALT` que espera cualquier cadena de caracteres. Un ejemplo de uso de esta etiqueta es:

```
<IMG SRC="fichero.gif" WIDTH="10" HEIGHT="10" ALT="Algo">
```

Los atributos sólo se ponen en la etiqueta de inicio: la etiqueta de fin nunca lleva atributos.

Cuando un usuario solicita una página **HTML** a un servidor web, este envía la página tal cual. En el momento en que el explorador recibe la página, interpreta las etiquetas que la página contiene, mostrando al usuario el resultado final (donde no aparecen ya las etiquetas, sino el resultado de su evaluación).

6.5.1. Estructura de una página

La estructura básica de una página se divide en cabecera (`<HEAD> ... </HEAD>`) y cuerpo (`<BODY> ... </BODY>`). El esqueleto básico de una página es:

Ejemplo 6.1

```
1 <!DOCTYPE HTML PUBLIC ".//W3C//DTD HTML 4.0//EN">
2 <HTML>
3 <HEAD>
4 Cabecera de la página
5 </HEAD>
6 <BODY>
7 Cuerpo de la página
8 </BODY>
9 </HTML>
```

El sentido de cada una de las líneas es:

[8] Sin embargo, si se quieren escribir páginas que sean compatibles con **XHTML**, tanto las etiquetas como los atributos de las etiquetas se tienen que escribir en minúsculas.

- Línea 1: Permite indicar la versión **HTML** que se va a utilizar para escribir la página. De este modo, el navegador puede optimizar la interpretación de la página al conocer de antemano la versión empleada. Normalmente no se incluye.
- Línea 2: Junto a la línea 9, indican el comienzo y fin de la página. La etiqueta `<HTML>` es obligatoria.
- Línea 3: Junto a la línea 5, indican el comienzo y fin de la cabecera.
- Línea 4: Lo que escribamos en la cabecera no se verá en la página. Se suele usar para indicar el título de la página con `<TITLE> ... </TITLE>`, incluir código que se ejecuta en el cliente con `<SCRIPT> ... </SCRIPT>`, definir un estilo con `<STYLE> ... </STYLE>` o incluir información sobre el contenido de la página con la etiqueta `<META>`.
- Línea 6: Junto a la línea 8, indican el comienzo y fin del cuerpo.
- Línea 7: Lo que escribamos en el cuerpo se verá en la página. La información contenida en esta sección se puede visualizar con apariencias muy diversas: sólo hay que aplicar distintos formatos a las secciones que la componen.

Cuando se escriban las etiquetas de fin hay que llevar mucho cuidado en el orden. Por ejemplo, en el esqueleto anterior, la etiqueta `<HTML>` aparece antes que `<BODY>`; la última etiqueta en aparecer es la primera que se tiene que cerrar. Por tanto, el orden correcto es primero `</BODY>` y luego `</HTML>`.

El siguiente código crea una página **HTML** sencilla, que podemos ver en la Figura 6.1. El título indicado sólo aparece en la barra de título de la ventana activa del navegador. El título se indica usando los código pareados `<TITLE> ... </TITLE>`, y se especifica usando un texto plano, sin códigos **HTML** de formato (no se puede poner en negrita o cursiva, por ejemplo).

Ejemplo 6.2

```
1  <HTML>
2  <HEAD>
3  <TITLE>Esto es una página HTML</TITLE>
4  </HEAD>
5  <BODY>
6  Esto es el cuerpo de una página HTML
7  <BR>
8  Esto tiene que aparecer en una línea nueva
9  <BR><BR>
10 Esto tiene que aparecer separado por una línea en blanco
11 </BODY>
12 </HTML>
```

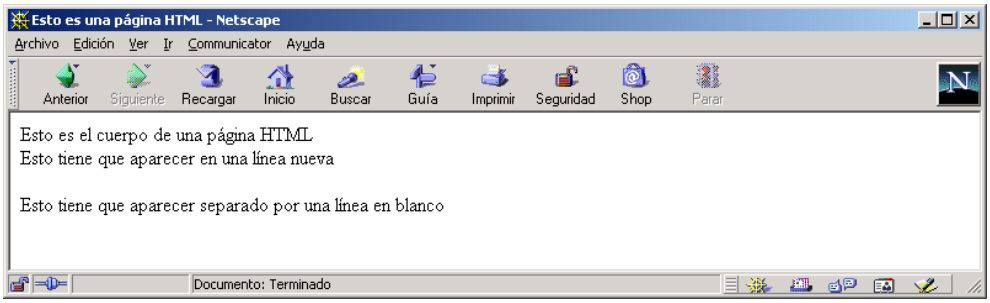


Figura 6.1: Primera página HTML

Es muy aconsejable poner título a todos los documentos y, además, se deberá procurar que éste sea lo más descriptivo posible, puesto que si algún usuario decide incluir nuestra página en su lista de enlaces (*bookmarks*), será el título lo que quede almacenado en dicha lista (junto con la **URL**). En el caso de que no se haya especificado un título, lo que se almacena en la lista de enlaces es la dirección de la página (**URL**).

En cualquier parte de una página **HTML**, si queremos incluir un comentario deberemos emplear la etiqueta `<!-- Comentario -->`. Los comentarios no se procesan y no producen una salida visible en la página.

Los saltos de línea que incluyamos en una página son irrelevantes (el navegador no los tiene en cuenta). Para producir un salto de línea se emplea la etiqueta `<BR>`. Los espacios en blanco también son irrelevantes: si separamos dos palabras por varios espacios en blanco, sólo se tendrá en cuenta uno de ellos. Si queremos incluir varios espacios en blanco, debemos de emplear el código de escape ` `. Por ejemplo, para incluir tres espacios en blanco se tiene que escribir `   `.

6.5.2. Caracteres especiales y secuencias de escape

Algunos sistemas informáticos trabajan con 7 bits en vez de 8. En esos casos, si se desea trabajar con caracteres **ASCII** de la parte superior de la tabla (128-255), es necesario codificarlos de algún modo. En los documentos **HTML** se pueden codificar de dos formas, mediante referencias decimales o referencias a entidades.

Las referencias decimales (también llamadas secuencias de escape) usan el formato `&#nnn;` donde `nnn` es el código **ASCII** en decimal del carácter. Por ejemplo, `&#193;` representa el carácter Á.

Las referencias a entidades usan el formato `&aaaa;` donde `aaaa` es una cadena de texto que representa el carácter. Por ejemplo, `&Aacute;` representa el carácter Á o `&ntilde;` es el carácter ñ.

Además, existen algunos caracteres de la parte inferior de la tabla de caracteres ASCII que poseen un significado especial en **HTML**, por lo que es necesario codificarlos. En el Cuadro 6.2 figuran los caracteres especiales con su secuencia de escape.

Carácter	Decimal	Entidad
"	&#34;	&quot;
&	&#38;	&amp;
<	&#60;	&lt;
>	&#62;	&gt;

Cuadro 6.2: Caracteres con un significado especial en HTML

También existen otros caracteres que son difíciles de conseguir desde el teclado, ya que no están representados. En el Cuadro 6.3 figuran algunos de los más significativos.

Carácter	Decimal	Entidad
§	&#167;	&sect;
©	&#169;	&copy;
®	&#174;	&reg;
¶	&#182;	&para;

Cuadro 6.3: Caracteres especiales

6.6. Metadatos

La etiqueta `<META>` permite indicar metadatos[9] de una forma muy simple. Permite incorporar información sobre el contenido de una página. Esta etiqueta sólo puede aparecer en la sección `<HEAD>`. Normalmente, esta etiqueta la emplean los motores de búsqueda, los navegadores y las herramientas de diseño de páginas web, pero no la emplean directamente los usuarios que visualizan una página.

La etiqueta `<META>` presenta dos variantes. La sintaxis de estas dos variantes es:

- `<META HTTP-EQUIV="propiedad" CONTENT="contenido">`. El atributo `HTTP-EQUIV` se emplea cuando la página web se recupera mediante **HTTP**. Los servidores web pueden usar el nombre de la propiedad para crear una cabecera según el estándar **RFC** 822[10] en la cabecera de la respuesta **HTTP** (algunas propiedades de la cabecera no se pueden fijar de esta forma). Por ejemplo, la

[9] Datos sobre los datos. Los metadatos, por ejemplo, permiten indicar cómo, cuándo y quién ha recogido una información y como le ha dado formato.

[10] *Standard for ARPA Internet Text Messages.*

siguiente etiqueta <META>:
<META HTTP-EQUIV="Expires" CONTENT="Tue, 20 Aug 1996 14:25:27 GMT">
se convierte en la siguiente cabecera **HTTP**:
Expires: Tue, 20 Aug 1996 14:25:27 GMT
que puede ser usada por la cache del navegador o por los servidores proxy para saber hasta cuando se puede emplear la copia de una página existente en la caché.

La etiqueta <META> también se puede emplear con la cabecera Refresh para refrescar una página cada cierto período de tiempo. Con esta etiqueta se puede recargar automáticamente una página cuyo contenido cambia con mucha frecuencia. Por ejemplo, la siguiente página se refresca cada 5 segundos:

Ejemplo 6.3

```
<HTML>
<HEAD>
<TITLE>Ejemplo de uso de la etiqueta META</TITLE>
<META HTTP-EQUIV="Refresh" CONTENT="5">
</HEAD>
<BODY>
Esta página se refresca cada 5 segundos
</BODY>
</HTML>
```

Si se indica una **URL** en el atributo CONTENT, se sustiuirá el documento actual por el indicado en la **URL** una vez haya transcurrido el número de segundos especificados. Esto es útil para redireccionar automáticamente al usuario desde una página web a otra (por ejemplo, cuando una página web ha cambiado de lugar). El siguiente ejemplo redirecciona automáticamente al usuario después de 3 segundos:

Ejemplo 6.4

```
<HTML>
<HEAD>
<TITLE>Ejemplo de uso de la etiqueta META</TITLE>
<META HTTP-EQUIV="Refresh" CONTENT="5;URL=http://www.nuevaweb.com">
</HEAD>
<BODY>
Esta página ha cambiado, la nueva dirección es
<A HREF="http://www.nuevaweb.com">http://www.nuevaweb.com</A>
</BODY>
</HTML>
```

- `<META NAME="propiedad" CONTENT="contenido">`. Se emplea para incluir propiedades del documento, tales como autor, fecha de caducidad, una lista de palabras clave, etc. El atributo `NAME` indica el nombre de la propiedad mientras que el atributo `CONTENT` especifica su valor. Por ejemplo, las siguientes etiquetas `<META>` indican el autor, una descripción, una lista de palabras clave y la fecha de creación de una página:

  Ejemplo 6.5

```
<META NAME="Author" CONTENT="Sergio Luján Mora">
<META NAME="Rights" CONTENT="Sergio Luján Mora">
<META NAME="Description" CONTENT="Una página de prueba">
<META NAME="Keywords" CONTENT="curso, html, diseño, web">
<META NAME="Date" CONTENT="Monday, 1 January, 2001">
```

  Esta variante de la etiqueta se emplea para indicar a los robots buscadores si la página se puede indexar y si se puede seguir buscando a partir de los enlaces que contiene la página. En el siguiente ejemplo, un robot buscador no debería ni indexar la página ni analizar la página para buscar más enlaces y seguir buscando:
  `<META NAME="ROBOTS" CONTENT="NOINDEX, NOFOLLOW">`

A título de ejemplo, la página principal de la web de la Universidad de Alicante contiene los siguientes metadatos:

Ejemplo 6.6

```
<META HTTP-EQUIV="pragma" CONTENT="no-cache">
<META NAME="Author" CONTENT="Universidad de Alicante">
<META NAME="Copyright" CONTENT="© Universidad de Alicante">
<META NAME="Description" CONTENT="Web que recoge toda la ...">
<META NAME="keywords" CONTENT="Universidad de Alicante, Alicante ...">
<META NAME="Formatter" CONTENT="Mozilla/4.5 [es](Win98; I)[Netscape]">
<META NAME="Generator" CONTENT="Mozilla/4.5 [es](Win98; I)[Netscape]">
<META NAME="robots" CONTENT="index, follow">
<META HTTP-EQUIV="Content-Language" CONTENT="ES">
<META HTTP-EQUIV="Content-Script-Type" CONTENT="JavaScript">
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=iso-8859">
```

6.7. Etiquetas HTML

Existen cientos de etiquetas, cada una con su conjunto de atributos. Además, no existe un estándar que acepten todos los navegadores: los dos principales navegadores, Netscape Navigator y Microsoft Internet Explorer, presentan diferencias entre ellos,

aceptando cada uno etiquetas que no acepta el otro. Existe un intento de estandarización por parte de **W3C**, que periódicamente publica borradores, recomendaciones y estándares definitivos. La última recomendación es **XHTML** 1.0[11] (combinación de **HTML** y **XML**), que sustituye a **HTML** 4.01.

En este libro sólo vamos a describir un subconjunto de las etiquetas reconocidas por Netscape Navigator 4.0 y superiores, ya que son las que aceptan la mayor parte de los navegadores actuales. Estas etiquetas las podemos clasificar en las siguientes categorías (en esta clasificación una etiqueta sólo aparece en una categoría, pero realmente hay etiquetas que se pueden clasificar en varias categorías):

- Etiquetas que definen la estructura del documento: `<HTML>`, `<HEAD>` y `<BODY>`.
- Etiquetas que pueden ir en la cabecera: `<TITLE>`, `<BASE>`, `<META>`, `<STYLE>` y `<LINK>`.
- Etiquetas que definen bloques de texto: `<ADDRESS>`, `<BLOCKQUOTE>`, `<DIV>`, `<H1> ... <H6>`, `<P>`, `<PRE>` y `<XMP>`.
- Etiquetas de listas: `<DIR>`, `<DL>`, `<DT>`, `<DD>`, `<MENU>`, `<OL>`, `<UL>` y `<LI>`.
- Etiquetas de características del texto: `<B>`, `<BASEFONT>`, `<BIG>`, `<BLINK>`, `<CITE>`, `<CODE>`, `<EM>`, `<FONT>`, `<I>`, `<KBD>`, `<PLAINTEXT>`, `<SMALL>`, `<S>`, `<STRIKE>`, `<STRONG>`, `<SUB>`, `<SUP>`, `<TT>`, `<U>` y `<VAR>`.
- Etiquetas de anclas y enlaces: `<A>`.
- Etiquetas de imágenes y mapas de imágenes: `<IMG>`, `<AREA>` y `<MAP>`.
- Etiquetas de tablas: `<TABLE>`, `<CAPTION>`, `<TR>`, `<TD>` y `<TH>`.
- Etiquetas de formularios: `<FORM>`, `<INPUT>`, `<SELECT>`, `<OPTION>`, `<TEXTAREA>`, `<KEYGEN>` y `<ISINDEX>`.
- Etiquetas de marcos: `<FRAME>`,`<FRAMESET>` y `<NOFRAMES>`.
- Etiquetas de situación de contenidos: `<LAYER>`, `<ILAYER>` y `<NOLAYER>`.
- Etiquetas de script: `<SCRIPT>`, `<NOSCRIPT>` y `<SERVER>`.
- Etiquetas de *applets* y *plug-ins*: `<APPLET>`, `<PARAM>`, `<EMBED>`, `<NOEMBED>` y `<OBJECT>`.
- Etiquetas de ajuste del texto: `<BR>`, `<CENTER>`, `<HR>`, `<MULTICOL>`, `<NOBR>`, `<SPACER>`, `<SPAN>` y `<WBR>`.

[11] Ya se encuentra en desarrollo **XHTML** 2.0.

Sólo vamos a comentar las etiquetas más importantes (las más empleadas) con los atributos más comunes. No es una explicación exhaustiva, pero suficiente para un primer contacto con **HTML**. En el Apéndice A se puede consultar una relación con todas las etiquetas que acepta Netscape Communicator 4.0.

6.8. Formato del texto

En esta sección vamos a repasar las etiquetas más importantes que permiten asignar formato al texto: los encabezados de sección, los estilos lógicos y físicos, la etiqueta `<FONT> ... </FONT>` y por último, como alinear el texto.

6.8.1. Encabezados de secciones

Existen seis niveles de encabezados, numerados del 1 al 6, y según tamaños decrecientes: el nivel 1 es la etiqueta `<H1> ... </H1>` (*heading*), la más prominente, y el nivel 6 es la etiqueta `<H6> ... </H6>`, la menos prominente. Los encabezados se suelen mostrar con tipos de letra más grandes, en negrita o más enfatizados que el texto normal. Pero los niveles 5 y 6 aparecen normalmente con un tamaño inferior al del texto normal. El tamaño de cada encabezado lo selecciona el navegador, por lo que la apariencia final de una página puede variar significativamente de uno a otro. Por ello, raramente se emplean en la actualidad.

Cuando se visualizan, los encabezados comienzan en una línea nueva y se suele dejar un espacio en blanco extra antes de ellos. Además, también provocan un salto de línea automáticamente. La sintaxis de esta etiqueta es:

Ejemplo 6.7

```
1 <Hn ALIGN="LEFT" | "CENTER" | "RIGHT" | "JUSTIFY"> ... </Hn>
```

donde `n` es un número del 1 al 6 y el atributo `ALIGN` especifica el alineamiento horizontal del encabezado[12]. Por ejemplo, en la Figura 6.2 vemos como se visualiza el siguiente código en un navegador.

Ejemplo 6.8

```
1 <HTML>
2 <BODY>
3 Texto normal
4 <H1>Encabezado nivel 1</H1>
5 <H2>Encabezado nivel 2</H2>
6 <H3>Encabezado nivel 3</H3>
7 <H4>Encabezado nivel 4</H4>
8 <H5>Encabezado nivel 5</H5>
```

[12] En la documentación de Netscape Communicator no figura el valor `JUSTIFY`.

```
<H6>Encabezado nivel 6</H6>
Texto normal
</BODY>
</HTML>
```

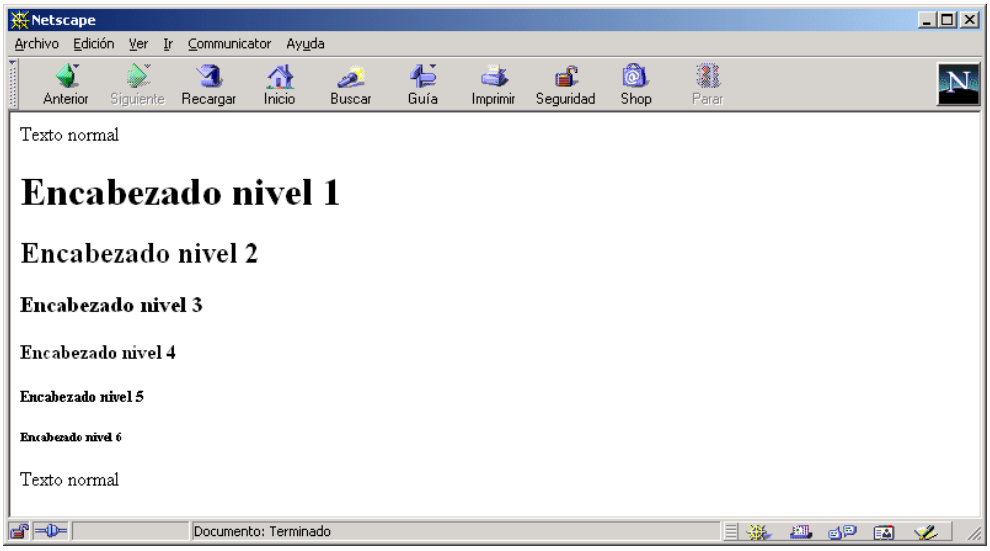


Figura 6.2: Ejemplo de encabezados

6.8.2. Formatos de caracteres

Las siguientes etiquetas se pueden emplear en línea (*inline*), lo que significa que no interrumpen el fluir del texto (no producen saltos de línea ni nada similar). Por tanto, se pueden aplicar a caracteres individuales.

Los formatos (también llamados estilos) se dividen en lógicos y físicos. Los formatos lógicos dependen del navegador (cada uno lo puede interpretar de distinta forma), mientras que los formatos físicos siempre se representan de la misma forma.

El siguiente código **HTML** muestra las etiquetas de los formatos más comunes. En la Figura 6.3 vemos como se visualiza en un navegador.

Ejemplo 6.9

```
<HTML>
<BODY>
Formatos físicos:<BR>
```

```
<B>Texto en negrita: &lt;B&gt;</B><BR>
<I>Texto en cursiva: &lt;I&gt;</I><BR>
<U>Texto subrayado: &lt;U&gt;</U><BR>
<TT>Texto en teletipo (fuente fija): &lt;TT&gt;</TT><BR>
<STRIKE>Texto tachado: &lt;STRIKE&gt;</STRIKE><BR>
<S>También funciona &lt;S&gt;</S><BR>
<BR>
Formatos lógicos:<BR>
<CITE>Cita: &lt;CITE&gt;</CITE><BR>
<CODE>Código: &lt;CODE&gt;</CODE><BR>
<DFN>Definición: &lt;DFN&gt;</DFN><BR>
<EM>Enfatizado: &lt;EM&gt;</EM><BR>
<KBD>Texto tecleado: &lt;KBD&gt;</KBD><BR>
<STRONG>Texto grueso: &lt;STRONG&gt;</STRONG><BR>
<VAR>Texto variable: &lt;VAR&gt;</VAR><BR>
</BODY>
</HTML>
```

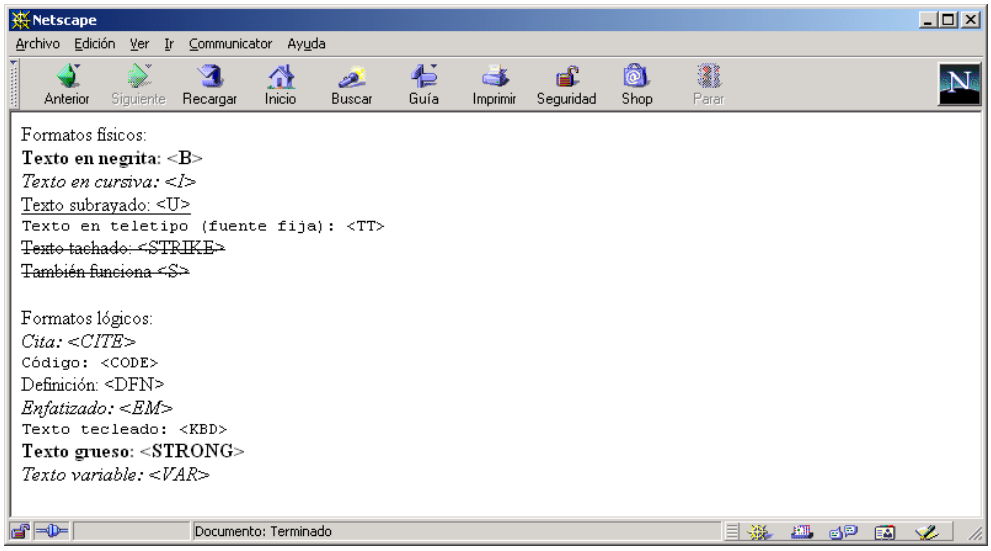


Figura 6.3: Formatos físicos y lógicos

Estas etiquetas se pueden combinar entre sí para crear nuevos formatos. Por ejemplo, mediante las etiquetas `<B> ... </B>` y `<I> ... </I>` se puede escribir un texto en negrita y cursiva simultáneamente. Únicamente hay que llevar cuidado y no intercalar unos códigos con otros, ya que se pueden obtener resultados no esperados. Por

ejemplo, el siguiente código presenta varios errores de solapamiento; en la Figura 6.4 se pueden apreciar los errores al visualizar la páginas en un navegador.

Ejemplo 6.10

```
<HTML>
<BODY>
<B>Esto está en negrita.
<I>Esto está en negrita y cursiva.</B>
Esto está en cursiva.</I>
<BR><BR>
<I>Esto está en cursiva.
<B>Esto está en negrita y cursiva.</I>
Esto está en negrita.</B>
<BR><BR>
<I>Esto está en cursiva.
<B>Esto está en negrita y cursiva.
<U>Esto está en negrita, cursiva y subrayado.</I>
Esto está en negrita y subrayado.</B>
Esto está en subrayado.</U>
</BODY>
</HTML>
```

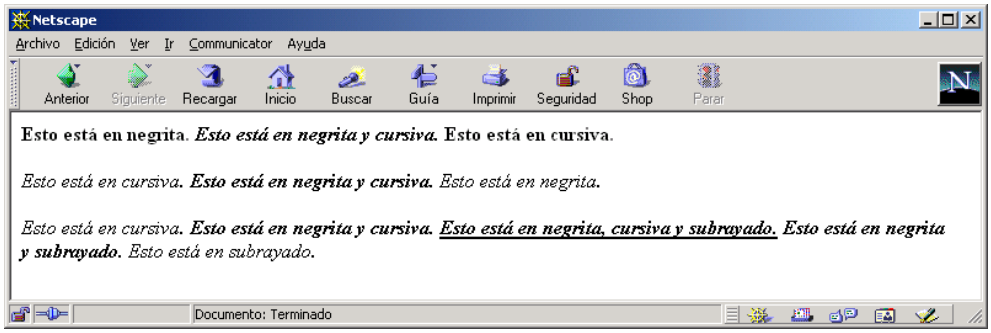


Figura 6.4: Resultados inesperados al solapar etiquetas

6.8.3. La etiqueta <FONT>

La etiqueta `<FONT> ... </FONT>` permite modificar el tipo de letra, el tamaño y el color del texto . Todo el texto entre las etiquetas `<FONT> ... </FONT>` se muestra según los valores especificados en los atributos de la etiqueta.

Para modificar el tipo de letra se emplea el atributo `FACE`. Se le puede indicar una lista de tipos de letra separados por comas. El navegador comprueba si el primer tipo de letra está disponible, si no es así prueba con el segundo y así sucesivamente. Si ninguno de los tipos de letra está disponible, no se produce ningún cambio en el tipo de letra.

Los tipos de letra pueden ser específicos o genéricos. Entre los primeros, los más empleados son `Arial`, `Courier`, `Helvetica`, `Tahoma`, `Times` y `Verdana`. Los tipos de letra genéricos son `serif`, `sans-serif`, `cursive`, `monospace` y `fantasy`. El siguiente código muestra el uso de distintos tipos de letra. En la Figura 6.5 podemos observar como se visualiza en Netscape Communicator 4.78 y en la Figura 6.6 como se muestra en Microsoft Internet Explorer 6.0. Se pueden apreciar ligeras diferencias entre ambos navegadores; además, algunos tipos de letra, como `Helvetica`, no se visualizan correctamente debido a que en el ordenador en el que se visualiza la página no están disponibles esos tipos de letra.

Ejemplo 6.11

```
1  <HTML>
2  <HEAD>
3  <TITLE>Distintos tipos de letra</TITLE>
4  </HEAD>
5  <BODY>
6  Tipos de letra específicos:<BR>
7  <FONT FACE="Arial">Texto en Arial</FONT><BR>
8  <FONT FACE="Helvetica">Texto en Helvetica</FONT><BR>
9  <FONT FACE="Tahoma">Texto en Tahoma</FONT><BR>
10 <BR>
11 Tipos de letra genéricos:<BR>
12 <FONT FACE="serif">Texto en serif</FONT><BR>
13 <FONT FACE="sans-serif">Texto en sans-serif</FONT><BR>
14 <FONT FACE="cursive">Texto en cursive</FONT><BR>
15 <FONT FACE="monospace">Texto en monospace</FONT><BR>
16 <FONT FACE="fantasy">Texto en fantasy</FONT><BR>
17 </BODY>
18 </HTML>
```

Para modificar el tamaño del texto se emplea el atributo `SIZE`. Este atributo define el tamaño de forma relativa, en un intervalo del 1 (letra más pequeña) a 7 (letra más grande). El tamaño base por defecto es 3[13]. También se puede indicar un valor relativo al tamaño base si se emplean los signos + o -. Por ejemplo, +2 significa un incremento en dos unidades respecto al tamaño base. En la Figura 6.7 vemos como se visualiza el siguiente código que muestra el uso del atributo `SIZE`.

Ejemplo 6.12

[13] Se puede cambiar con la etiqueta `<BASEFONT SIZE="numero">`.

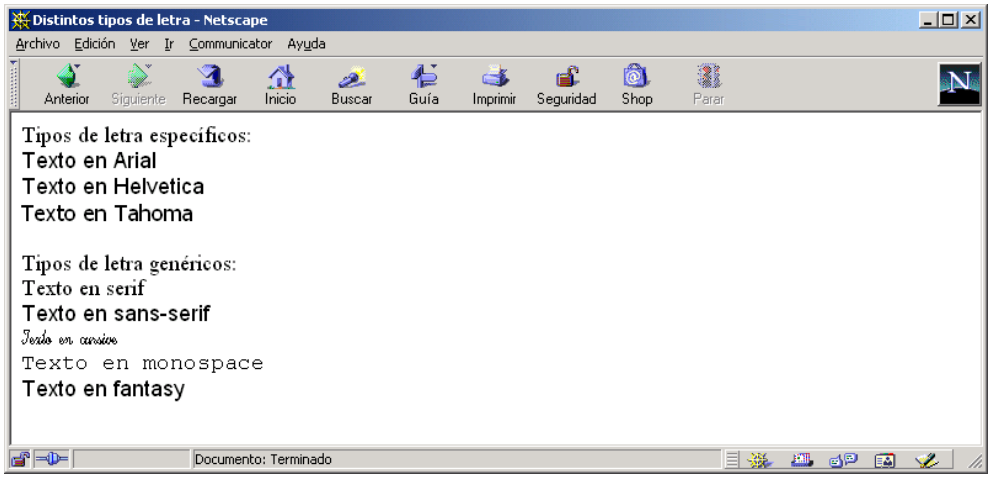


Figura 6.5: Distintos tipos de letra con la etiqueta <FONT> en Netscape Communicator

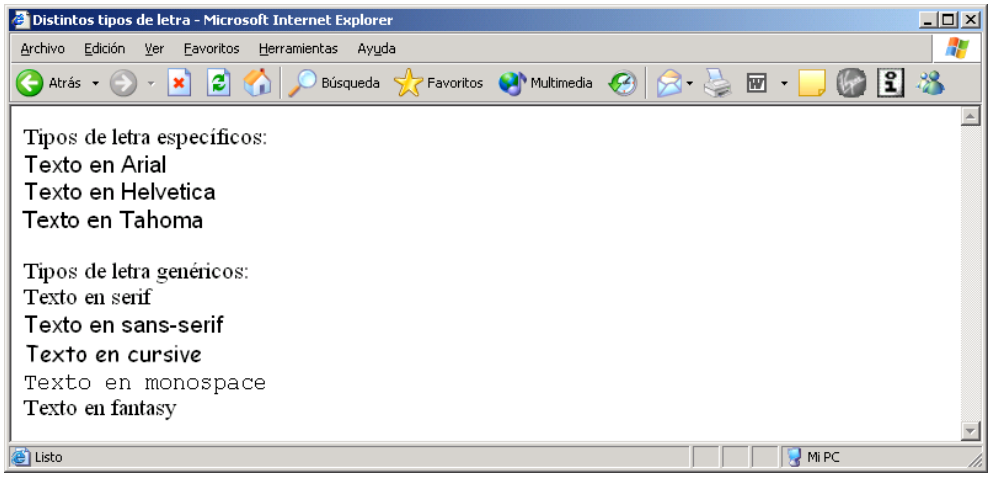


Figura 6.6: Distintos tipos de letra con la etiqueta <FONT> en Microsoft Internet Explorer

```
1  <HTML>
2  <BODY>
3  Distintos tamaños de letra:<BR>
4  <FONT SIZE="1">Texto en 1</FONT><BR>
5  <FONT SIZE="2">Texto en 2</FONT><BR>
6  <FONT SIZE="3">Texto en 3</FONT><BR>
7  <FONT SIZE="4">Texto en 4</FONT><BR>
8  <FONT SIZE="5">Texto en 5</FONT><BR>
9  <FONT SIZE="6">Texto en 6</FONT><BR>
10 <FONT SIZE="7">Texto en 7</FONT><BR>
11 </BODY>
12 </HTML>
```

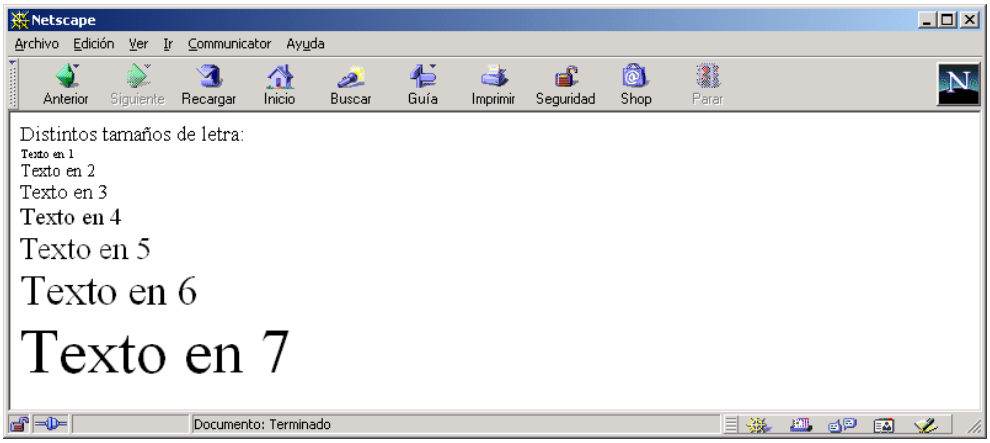


Figura 6.7: Distintos tamaños de letra con la etiqueta <FONT>

6.8.4. Alineamiento del texto

La etiqueta `<P> ... </P>`[14] (*paragraph*) se emplea para marcar un párrafo de texto. Un párrafo comienza en una línea nueva y el navegador suele dejar un espacio en blanco extra antes del párrafo.

Para alinear el contenido de un párrafo se emplea el atributo `ALIGN`. El contenido del párrafo puede alinearse a la izquierda (`LEFT`), a la derecha (`RIGHT`), centrado (`CENTER`) o justificado (`JUSTIFY`). El siguiente código **HTML** muestra el mismo

[14] En **HTML** 4.01 la etiqueta de cierre `</P>` es opcional, pero a partir de **XHTML** 1.0 es obligatoria.

párrafo alineado de cuatro formas distintas. En la Figura 6.8 podemos ver el código visualizado en un navegador.

Ejemplo 6.13

```
<HTML>
<BODY>
<P ALIGN="LEFT">
Los enlaces o hiperenlaces permiten relacionar distintas páginas
entre sí (hipertexto). Esta característica da la posibilidad de
organizar la información en distintas páginas HTML enlazadas, de
forma que el usuario pueda seleccionar la que más le interese en
cada momento.
</P>
<P ALIGN="RIGHT">
Los enlaces o hiperenlaces permiten relacionar distintas páginas
entre sí (hipertexto). Esta característica da la posibilidad de
organizar la información en distintas páginas HTML enlazadas, de
forma que el usuario pueda seleccionar la que más le interese en
cada momento.
</P>
<P ALIGN="CENTER">
Los enlaces o hiperenlaces permiten relacionar distintas páginas
entre sí (hipertexto). Esta característica da la posibilidad de
organizar la información en distintas páginas HTML enlazadas, de
forma que el usuario pueda seleccionar la que más le interese en
cada momento.
</P>
<P ALIGN="JUSTIFY">
Los enlaces o hiperenlaces permiten relacionar distintas páginas
entre sí (hipertexto). Esta característica da la posibilidad de
organizar la información en distintas páginas HTML enlazadas, de
forma que el usuario pueda seleccionar la que más le interese en
cada momento.
</P>
</BODY>
</HTML>
```

También se puede emplear la etiqueta `<CENTER> ... </CENTER>` para centrar el texto. Esta etiqueta no sólo permite centrar el texto, sino cualquier otro elemento que contenga la página, como puede ser una imagen o una tabla.

Los bloques de texto se especifican usando la etiqueta `<BLOCKQUOTE> ... </BLOCKQUOTE>`. Un bloque de texto aparece sangrado hacia la derecha. Se suele emplear para marcar citas textuales o definiciones de especial relevancia. Los bloques de texto se pueden anidar para producir un mayor sangrado. El siguiente código muestra el empleo de esta etiqueta. En la Figura 6.9 se puede ver como se muestra en un

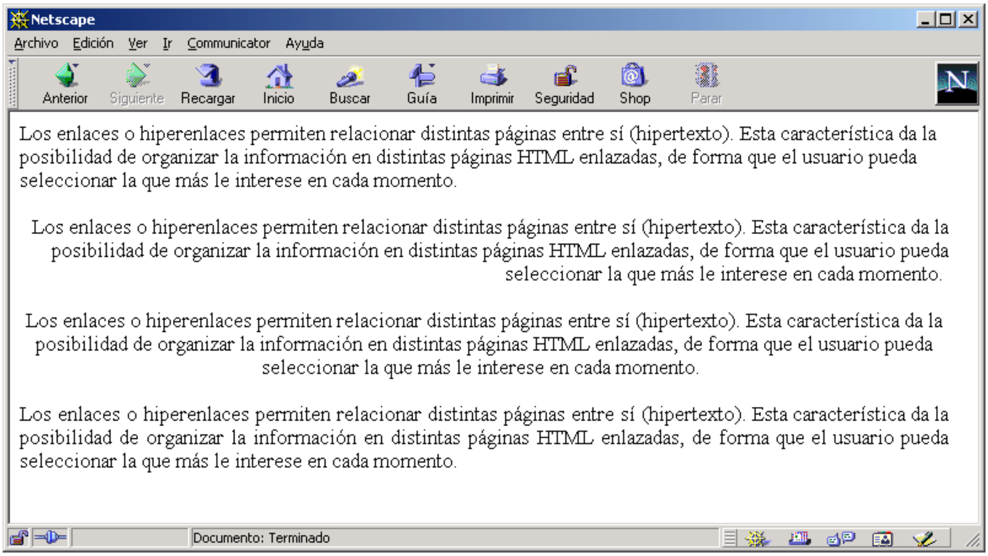


Figura 6.8: Alineamiento de párrafos: izquierda, derecha, centrado y justificado

navegador web.

Ejemplo 6.14

```
<HTML>
<BODY>
Este texto no tiene sangría.
<BLOCKQUOTE>
1. Un nivel de sangría.
<BLOCKQUOTE>
2. Dos niveles de sangría.
<BLOCKQUOTE>
3. Tres niveles de sangría.
</BLOCKQUOTE>
</BLOCKQUOTE>
Volvemos al nivel 1.
</BLOCKQUOTE>
Volvemos a texto
sin sangría.
</BODY>
</HTML>
```

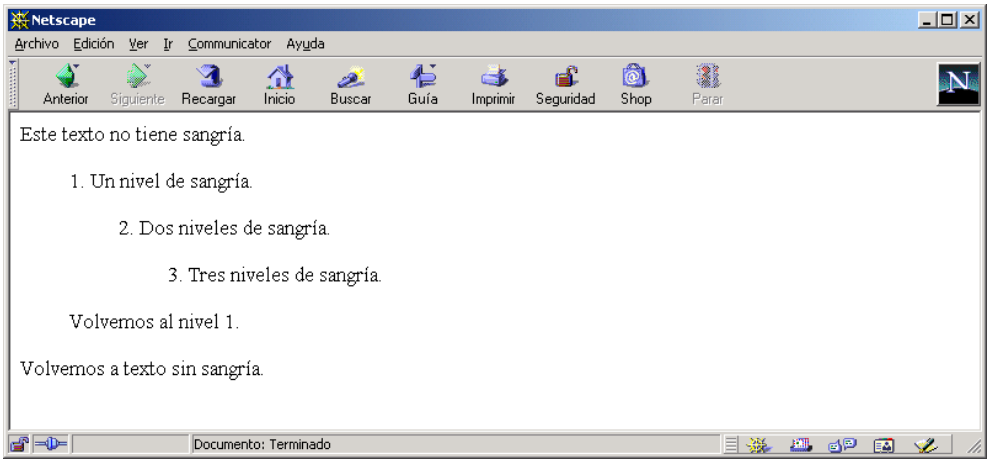


Figura 6.9: Bloques de texto con distinta sangría

6.8.5. Líneas horizontales

Si se desea dibujar una línea horizontal como separador de bloques de texto, se puede emplear la etiqueta `<HR>` (*horizontal rule*). El uso de este elemento gráfico permite organizar el texto, separando las distintas secciones que lo componen.

Esta etiqueta posee los siguientes atributos que permiten controlar la apariencia de la línea horizontal dibujada:

- `WIDTH="anchura"`. Anchura de la línea horizontal. Se puede especificar usando un valor absoluto (número de pixels) o un valor relativo (porcentaje respecto a la anchura de la página). El valor por defecto es la anchura de la página.
- `ALIGN="LEFT" | "RIGHT" | "CENTER"`. Alineamiento de la línea horizontal respecto a la página. Por defecto, aparece centrada.
- `SIZE="grosor"`. Grosor de la línea horizontal, expresado en pixels. El valor por defecto es 2.
- `NOSHADE`. Por defecto las líneas horizontales aparecen sobreadas. Este atributo permite eliminar el sobreado, lo que produce una línea sólida.

El siguiente código muestra el empleo de esta etiqueta. En la Figura 6.10 se puede ver como se muestra en un navegador web.

Ejemplo 6.15

```
1 <HTML>
2 <BODY>
```

```
3  Una línea normal
4  <HR>
5  Una línea sin sombra
6  <HR NOSHADE>
7  Una línea con una anchura del 50%
8  <HR WIDTH="50%">
9  Una línea con una anchura del 50% alineada a la izquierda
10 <HR WIDTH="50%" ALIGN="LEFT">
11 Una línea con una anchura del 50% alineada a la derecha
12 <HR WIDTH="50%" ALIGN="RIGHT">
13 Una línea con una anchura de 50 pixels
14 <HR WIDTH="50">
15 Una línea con un grosor de 15 pixels
16 <HR SIZE="15">
17 Una línea con un grosor de 15 pixels sin sombra
18 <HR SIZE="15" NOSHADE>
19 Una línea normal
20 <HR>
21 </BODY>
22 </HTML>
```

6.9. Listas

Las listas permiten organizar la información de una manera lógica, lo que facilita su legibilidad. Existen cinco tipos de listas en **HTML**: listas de definición, listas ordenadas, listas no ordenadas, listas de directorio y listas de menú. Como las dos últimas listas están obsoletas, ya que se visualizan como las listas no ordenadas, no las vamos a tratar.

Las listas se pueden anidar entre sí, incluso si son de distinto tipo. En el caso de anidar listas no numeradas, cada nivel de anidamiento tendrá un tipo de símbolo distinto.

6.9.1. Listas de definición

Una lista de definición se emplea para mostrar términos con sus correspondientes definiciones, como si se tratase de un glosario o diccionario.

Una lista de definición se crea con la etiqueta `<DL> ... </DL>` (*definition list*). Contiene una serie de términos, que se definen con la etiqueta `<DT> ... </DT>`[15] (*definition term*), y cada término posee una o más definiciones, que se indican cada

[15] En **HTML** 4.01 la etiqueta de cierre `</DT>` es opcional, pero a partir de **XHTML** 1.0 es obligatoria.

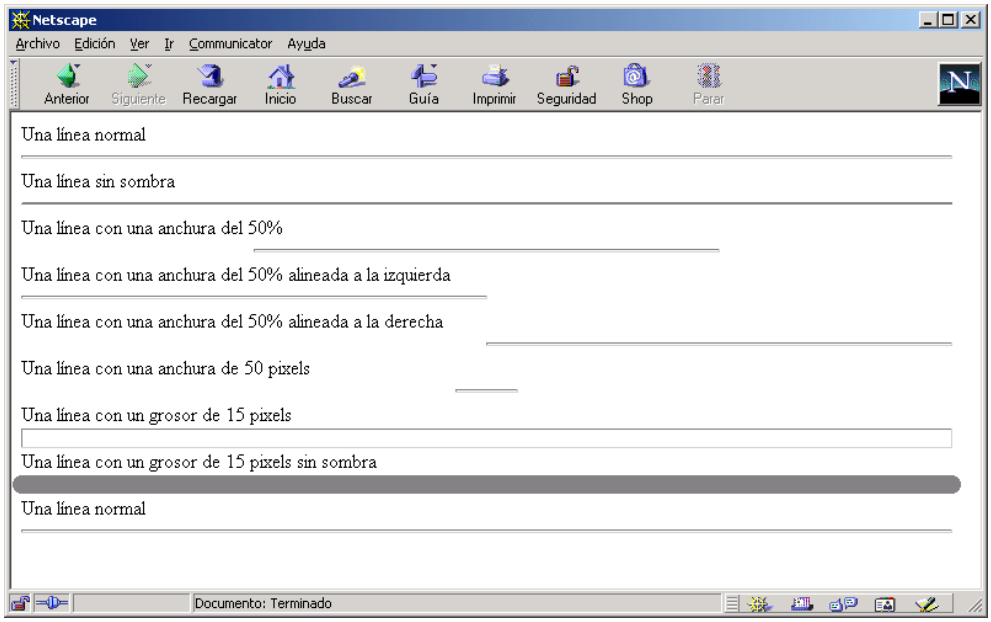


Figura 6.10: Distintos tipos de líneas

una con la etiqueta `<DD> ... </DD>`[16] (*definition description*). Cuando se visualiza una lista de definición, las definiciones de cada término aparecen con una sangría hacia la derecha.

El siguiente ejemplo muestra como se usan estas etiquetas. Aunque las líneas que contienen las etiquetas `<DD>` aparecen con unos espacios en blanco al principio, estos no influyen para nada en su visualización. En la Figura 6.11 vemos el resultado que produce este ejemplo.

Ejemplo 6.16

```
1  <HTML>
2  <BODY>
3  <DL>
4  <DT>BANCO</DT>
5    <DD>Lugar donde se deposita dinero</DD>
6    <DD>Sitio donde se sienta la gente</DD>
7  <DT>GATO</DT>
8    <DD>Animal de cuatro patas con pelo</DD>
9    <DD>Herramienta para levantar un vehículo</DD>
10 <DT>ORDENADOR</DT>
11   <DD>Aparato electrónico que realiza cálculos</DD>
12 </DL>
13 </BODY>
14 </HTML>
```

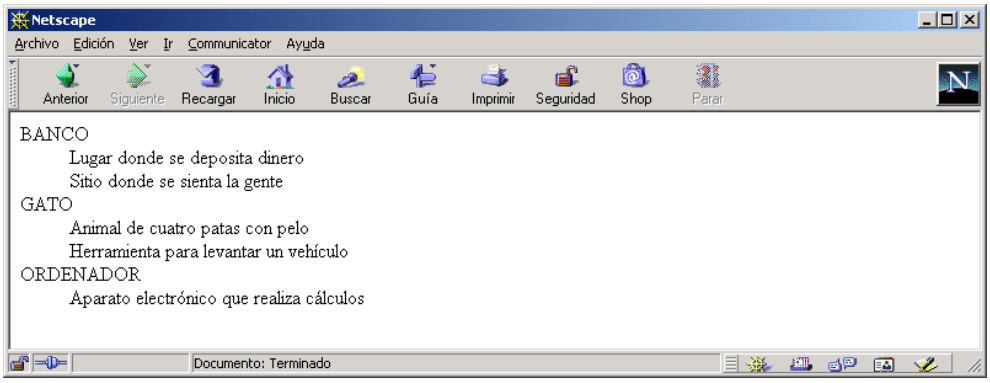


Figura 6.11: Listas de definición

[16] En **HTML** 4.01 la etiqueta de cierre `</DD>` es opcional, pero a partir de **XHTML** 1.0 es obligatoria.

6.9.2. Listas ordenadas

En las listas ordenadas o numeradas, cada elemento aparece numerado. La etiqueta `<OL> ... <OL>` (*ordered list*) define una lista de este tipo. Cada elemento se define con la etiqueta `<LI> ... </LI>`[17] (*list item*). Esta etiqueta posee dos atributos: `START` y `TYPE`.

El atributo `START` indica el valor desde el que empieza la numeración; tiene que ser un valor positivo.

El atributo `TYPE` indica el tipo de numeración de los elementos de la lista. Los posibles valores de este atributos son:

- `A`: indica una numeración con letras mayúsculas. Ejemplo: `A, B, C`.
- `a`: indica una numeración con letras minúsculas. Ejemplo: `a, b, c`.
- `I`: indica una numeración con números romanos en mayúsculas. Ejemplo: `I, II, III`.
- `i`: indica una numeración con números romanos en minúsculas. Ejemplo: `i, ii, iii`.
- `1`: indica una numeración con números. Ejemplo: `1, 2, 3`.

El siguiente ejemplo muestra el uso de esta etiqueta. Además, también se puede ver como se pueden anidar listas (incluir una lista dentro de otra lista). En la Figura 6.12 vemos como se muestra esta página.

Ejemplo 6.17

```
1  <HTML>
2  <BODY>
3  Lista normal, con anidamiento:
4  <OL>
5  <LI>Elemento 1</LI>
6  <LI>Elemento 2</LI>
7  <LI>Lista anidada:
8      <OL>
9      <LI>Elemento 1</LI>
10     <LI>Elemento 2</LI>
11     </OL>
12 </LI>
13 </OL>
14 Lista numerada con letras en mayúsculas:
15 <OL TYPE="A">
16 <LI>Elemento 1</LI>
```

[17] En **HTML** 4.01 la etiqueta de cierre `</LI>` es opcional, pero a partir de **XHTML** 1.0 es obligatoria.

```
17 <LI>Elemento 2</LI>
18 </OL>
19 Lista numerada con números romanos:
20 <OL TYPE="i">
21 <LI>Elemento 1</LI>
22 <LI>Elemento 2</LI>
23 </OL>
24 </BODY>
25 </HTML>
```

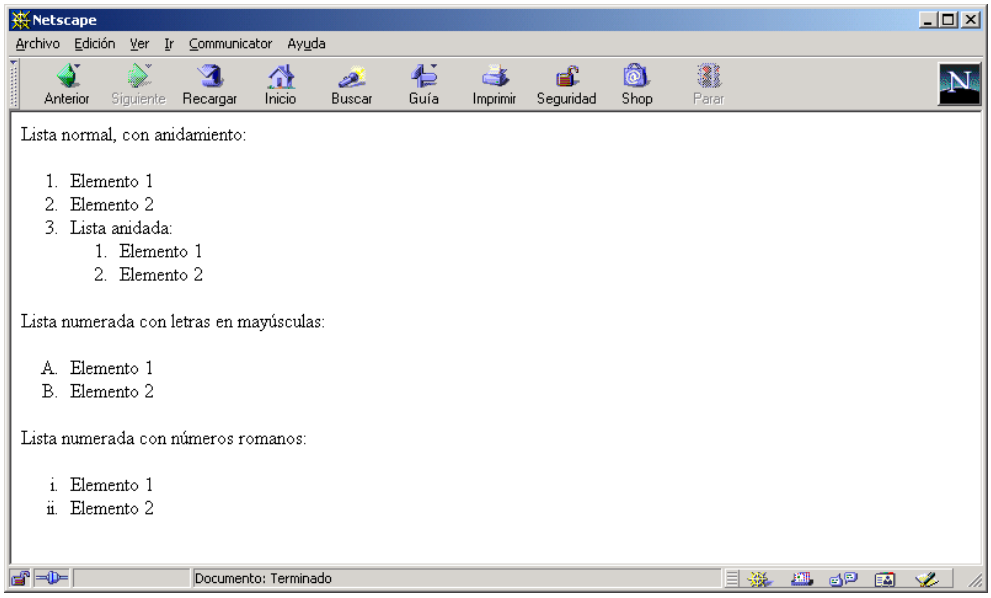


Figura 6.12: Listas ordenadas

6.9.3. Listas no ordenadas

En las listas no ordenadas, los elementos aparecen marcados mediante unos pequeños elementos gráficos, llamados en inglés *bullet*. La etiqueta `<UL> ... </UL>` (*unordered list*) define una lista no ordenada. Cada elemento se define con la etiqueta `<LI> ... </LI>`[18] (*list item*).

[18] En **HTML** 4.01 la etiqueta de cierre `</LI>` es opcional, pero a partir de **XHTML** 1.0 es obligatoria.

Esta etiqueta posee el atributo `TYPE`, que permite cambiar el elemento gráfico empleado para marcar los elementos. Los posibles valores que puede tomar este atributo son:

- `CIRCLE`: un disco con el centro vacío.
- `DISC`: un disco sólido.
- `SQUARE`: un cuadrado.

El siguiente código muestra como se emplea esta etiqueta. Además, en la última lista hay un anidamiento de varias listas. En la Figura 6.13 se puede ver que cuando se anidan varias listas, el elemento gráfico *bullet* cambia automáticamente, según cual sea el nivel de anidamiento.

Ejemplo 6.18

```
<HTML>
<BODY>
<UL TYPE="CIRCLE">
<LI>Elemento 1</LI>
<LI>Elemento 2</LI>
</UL>
<UL TYPE="DISC">
<LI>Elemento 1</LI>
<LI>Elemento 2</LI>
</UL>
<UL TYPE="SQUARE">
<LI>Elemento 1</LI>
<LI>Elemento 2</LI>
</UL>
<UL>
<LI>Elemento a:
    <UL>
    <LI>Elemento b:
        <UL>
        <LI>Elemento c:</LI>
        </UL>
    </LI>
    </UL>
</LI>
</UL>
</BODY>
</HTML>
```

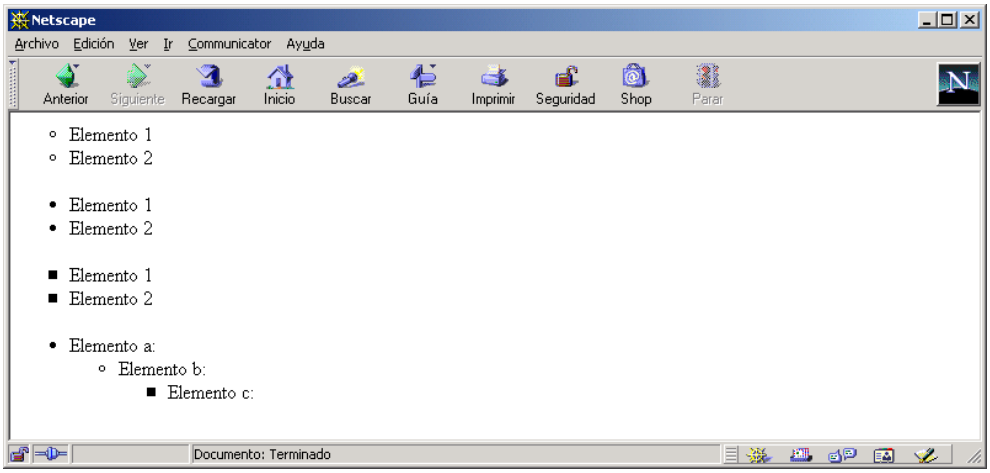


Figura 6.13: Listas no ordenadas

6.10. Colores

En **HTML** se puede cambiar el color de distintos elementos, como el color de fondo de una página, el color del texto, el color de una tabla, etc. Existen dos formas de especificar los colores:

- Mediante las componentes *Red Green Blue* (**RGB**). Mediante esta codificación, se especifica la intensidad de los colores básicos rojo, verde y azul que permiten obtener por combinación cualquier otro color. Cada componente puede variar entre 0 y 255, por lo que se tienen 16 777 216 (256 x 256 x 256) colores. Las componentes se tienen que expresar en hexadecimal y al principio se tiene que poner el signo `#`; por tanto, el formato general es `#RRGGBB`. Por ejemplo, el color negro se representa mediante `#000000`, el color blanco como `#FFFFFF` y un rojo brillante como `#FF0000`.
- Mediante los nombres de los colores. Existen una serie de colores a los que se les ha asignado un nombre en inglés, como *green*, *olive* o *white*.

En el Apéndice B se incluye más información sobre el uso de colores en **HTML** y como pasar las componentes **RGB** de decimal a hexadecimal.

6.10.1. Color de fondo de una página

El color de fondo de una página se puede cambiar mediante el atributo `BGCOLOR` de la etiqueta `<BODY> ... </BODY>`.

6.10.2. Color del texto

La etiqueta `<FONT> ... </FONT>` posee el atributo `COLOR` que permite indicar el color del texto. También se puede cambiar el color del texto para toda una página, el color de los enlaces, el color de los enlaces visitados y el color de los enlaces al activarse. Para ello se emplean los atributos `TEXT`, `LINK`, `VLINK` y `ALINK` de la etiqueta `<BODY> ... </BODY>`.

En el siguiente ejemplo, se cambia el color del texto y de los enlaces para que todos tengan el mismo color y no se puedan diferenciar unos de otros.

Ejemplo 6.19

```
1 <HTML>
2 <BODY TEXT="black" LINK="black" VLINK="black" ALINK="black">
3 <A HREF="http://www.ua.es">Universidad de Alicante</A>
4 <BR>
5 <A HREF="http://www.eps.ua.es">Escuela Politécnica Superior</A>
6 </BODY>
7 </HTML>
```

6.11. Enlaces

Los enlaces o hiperenlaces permiten relacionar distintas páginas entre sí (hipertexto). Esta característica da la posibilidad de organizar la información en distintas páginas **HTML** enlazadas, de forma que el usuario pueda seleccionar la que más le interese en cada momento.

Un hiperenlace puede hacer referencia a un punto determinado de la página que lo contiene, a otra página **HTML** o a un punto determinado de otra página **HTML**. En los dos últimos casos, la página destino puede residir en el mismo servidor que la página que contiene el hiperenlace o en otro distinto.

La etiqueta que utiliza **HTML** para definir un hiperenlace es `<A> ... </A>`[19]. Todo aquello que aparezca en una página **HTML** entre dichas etiquetas se considera un hiperenlace (será el objeto sensible que al ser pulsado producirá el salto al destino del enlace). Normalmente suele utilizarse texto e imágenes como hiperenlaces.

6.11.1. Enlace a un punto del mismo documento

Un enlace de este tipo consta de dos partes: una referencia y un destino. El destino se identifica por un nombre. La referencia hará alusión al nombre del destino. En una página se pueden incluir varias referencias a un mismo destino, pero no se pueden crear varios destinos con el mismo nombre. La forma de definir este enlace es:

[19] La `A` de esta etiqueta viene de la palabra inglesa *anchor*, que significa "ancla".

- Referencia: `<A HREF="#nombre">objeto del enlace</A>`.
- Destino: `<A NAME="nombre">objeto del destino</A>`.

Como puede observarse, en el caso de la referencia el nombre al que hace alusión va precedido del símbolo almohadilla (#), mientras que en el destino no.

En el siguiente código de ejemplo, tenemos un enlace sobre la misma página. En la Figura 6.14 vemos que la palabra `enlace` es un enlace; al pulsar sobre ella saltamos al destino que podemos ver en la Figura 6.15.

Ejemplo 6.20

```
<HTML>
<HEAD>
<TITLE>Esto es una página HTML</TITLE>
</HEAD>
<BODY>
Aquí tenemos un <A HREF="#destino">enlace</A> sobre la misma
página. Hay que dejar muchas líneas en blanco para que se
pueda comprobar el efecto.
<BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR>
<BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR>
<A NAME="destino">Esto</A> es el destino del enlace que aparece
al principio de la página.
</BODY>
</HTML>
```

6.11.2. Enlace a otro documento

Para incluir un enlace en un documento a otro documento, simplemente hay que incluir en el documento origen una referencia al documento destino. En este último no hace falta indicar que es destino de un enlace. La forma de definir este enlace es:

- Referencia: `<A HREF="pagina.html">objeto del enlace</A>`.

Muy importante: cuando indiquemos el nombre del documento destino (`pagina.html`), hay que tener mucho cuidado con las mayúsculas y minúsculas. En el siguiente ejemplo tenemos dos páginas con colores de fondo distintos; en cada una de ellas figura un enlace a la otra.

- Página `blanco.html` (Figura 6.16):

Ejemplo 6.21

```
<HTML>
<HEAD>
```

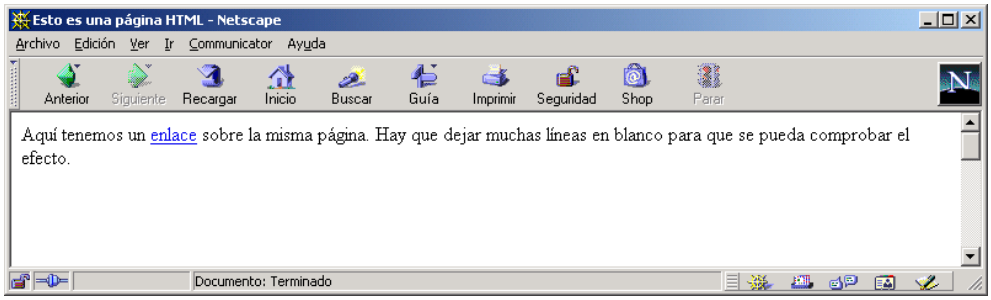


Figura 6.14: Enlace a un destino interno

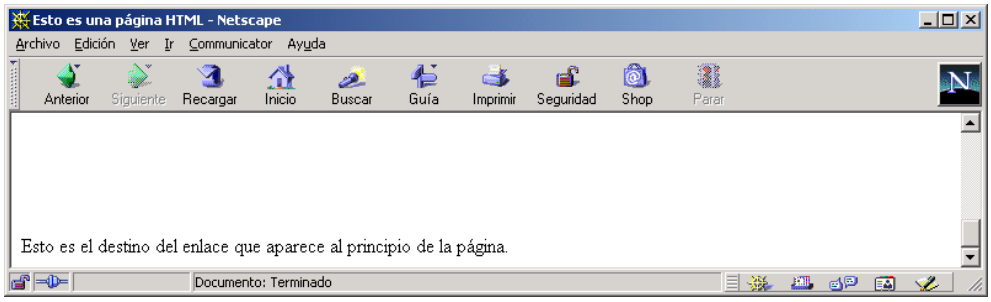


Figura 6.15: Destino del enlace interno

```
<TITLE>Esto es una página HTML</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<B>Aquí tenemos un <A HREF="azul.html">enlace</A> a otra
página que tiene el fondo azul.</B>
</BODY>
</HTML>
```

- Página `azul.html` (Figura 6.17):

Ejemplo 6.22

```
<HTML>
<HEAD>
<TITLE>Esto es una página HTML</TITLE>
</HEAD>
<BODY BGCOLOR="#BBBBFF">
<B>Aquí tenemos un <A HREF="blanco.html">enlace</A> a otra
página que tiene el fondo blanco.</B>
</BODY>
</HTML>
```

Figura 6.16: Página con enlace a otra página

6.11.3. Enlace a un punto de otro documento

Este tipo de enlace es una combinación de los dos anteriores. La forma de definir este enlace es:

- Referencia: `<A HREF="pagina.html#nombre">objeto del enlace</A>`.
- Destino: `<A NAME="nombre">objeto del destino</A>`.

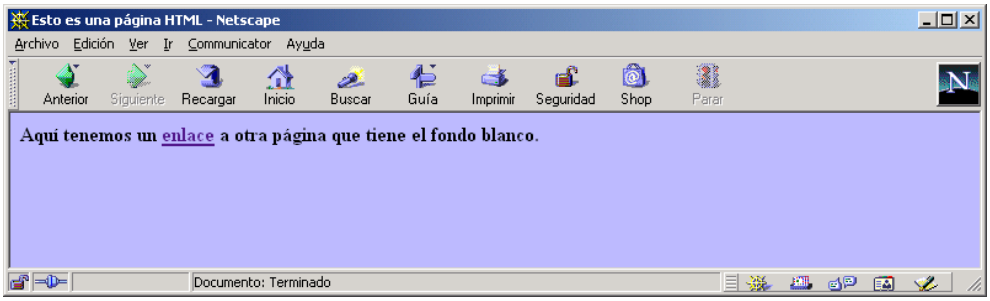


Figura 6.17: Página con enlace a otra página

En el siguiente ejemplo tenemos una página con dos enlaces a la misma página, pero a distintos destinos dentro de esa página.

- Página `unapagina.html` (Figura 6.18):

Ejemplo 6.23

```
<HTML>
<HEAD>
<TITLE>Esto es una página HTML</TITLE>
</HEAD>
<BODY>
<B>Aquí tenemos un <A HREF="otrapagina.html#destino1">enlace</A>
a un destino de otra página.
<BR><BR>
Aquí tenemos otro <A HREF="otrapagina.html#destino2">enlace</A>
a otro destino de la misma página que antes.
</B>
</BODY>
</HTML>
```

- Página `otrapagina.html` (Figura 6.19 y Figura 6.20):

Ejemplo 6.24

```
<HTML>
<HEAD>
<TITLE>Esto es una página HTML</TITLE>
</HEAD>
<BODY>
<B>Aquí tenemos un <A NAME="destino1">destino</A>, el número 1.
<BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR>
<BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR><BR>
```

```
Aquí tenemos otro <A NAME="destino2">destino</A>, el número 2.
</B>
</BODY>
</HTML>
```

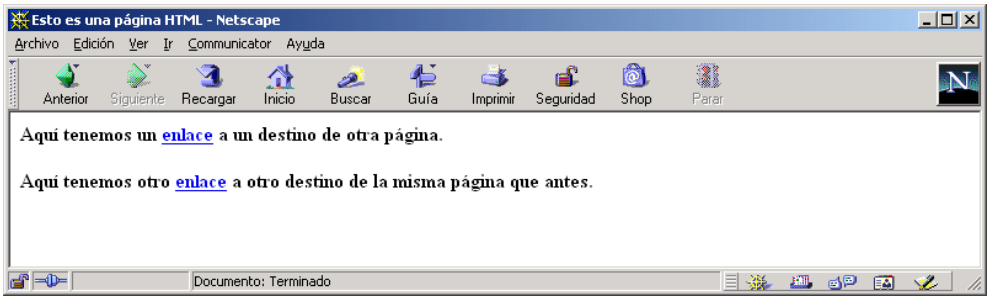


Figura 6.18: Página con dos enlaces a otra página

6.11.4. Envío de un correo electrónico

Mediante el protocolo `mailto:` se puede enviar un correo electrónico cuando el usuario pulse sobre un enlace. Realmente no se envía un correo electrónico automáticamente: se abre el programa de correo electrónico que tenga configurado el usuario por defecto y se crea un nuevo mensaje, pero no se envía hasta que el usuario no lo confirme.

La sintaxis de este protocolo es:
`mailto:dir1[,dir2,...][?opcion1[&opcion2&...]]`

donde `dir1, dir2, ...` son direcciones de correo electrónico y `opcion1, opcion2, ...` es una lista de opciones "opcional". Las principales opciones que se pueden emplear son:

- `cc` (*carbon copy*): para indicar el envío de copias del mensaje a otros destinatarios.
- `bcc` (*blind carbon copy*): para indicar el envío de copias del mensaje a otros destinatarios ocultos.
- `subject`: el título del mensaje.
- `body`: el texto del mensaje.

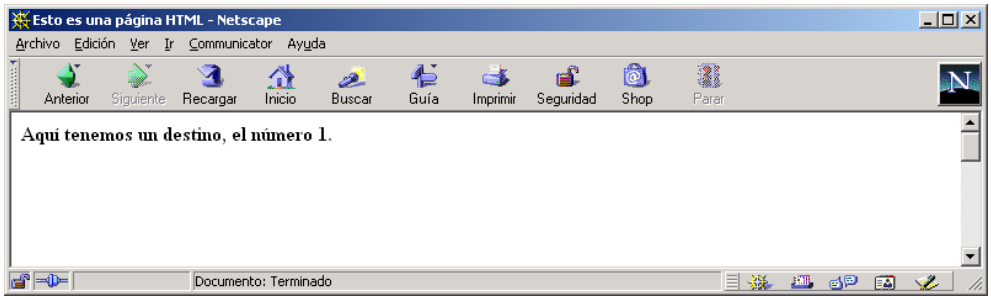


Figura 6.19: Página destino de los enlaces

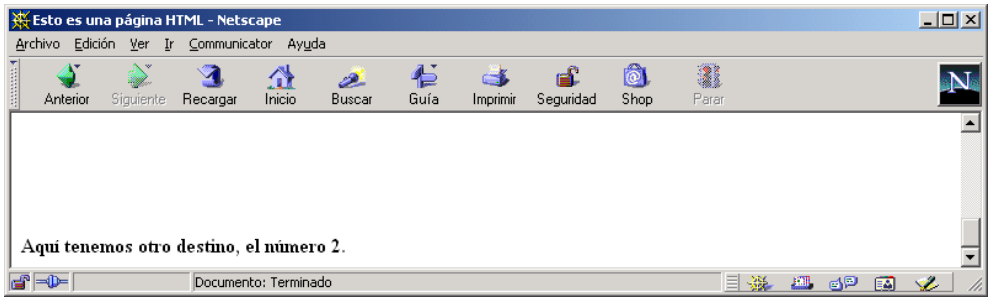


Figura 6.20: Página destino de los enlaces

Estas opciones se pueden combinar en un único enlace. Sólo hay que tener en cuenta que se tienen que separar con el símbolo "&" y que los valores de las opciones se tienen que escribir empleando "codificación URL" (*URL encoding*). En esta codificación, una serie de caracteres especiales, como por ejemplo "&", "%", "$" o "ñ", se codifican usando el símbolo "%" seguido de dos dígitos que expresan, en hexadecimal, su código **ASCII**. Por ejemplo, la cadena "&%$ñ" se codificaría como "%26%25%24%F1".

El siguiente ejemplo contiene seis enlaces con distintas opciones que envían un correo electrónico al ser pulsado cualquiera de ellos. En la Figura 6.21 se puede observar como se visualiza esta página en un navegador: en la barra de estado del navegador se puede apreciar la **URL** del último enlace; en la Figura 6.22 se muestra la ventana de Netscape Messenger que se abre automáticamente al pulsar sobre el último enlace.

Ejemplo 6.25

```
<HTML>
<HEAD>
<TITLE>Ejemplo de envío de correo electrónico</TITLE>
</HEAD>
<BODY>
<A HREF="mailto:a@b.c">Simple</A>
<BR>
<A HREF="mailto:a@b.c,d@e.f">A dos personas</A>
<BR>
<A HREF="mailto:a@b.c?cc=d@e.f">Con copia</A>
<BR>
<A HREF="mailto:a@b.c?bcc=d@e.f">Con copia oculta</A>
<BR>
<A HREF="mailto:a@b.c?subject=El%20tema%20del%20mensaje">Con título</A>
<BR>
<A HREF="mailto:a@b.c?cc=d@e.f&subject=Tema&body=Cuerpo%20mensaje">
Combinando varias opciones</A>
</BODY>
</HTML>
```

6.12. Tablas

Hasta que aparecieron las tablas en el lenguaje **HTML**, la única posibilidad de tabular cosas en una página consistía en usar texto con preformato (`<PRE> ... </PRE>`) y colocar manualmente los espacios hasta que las columnas estuviesen perfectamente alineadas. Este proceso, además de ser realmente tedioso, no ofrece resultados con la vistosidad deseada.

Las tablas se construyen siguiendo una serie de reglas sencillas:

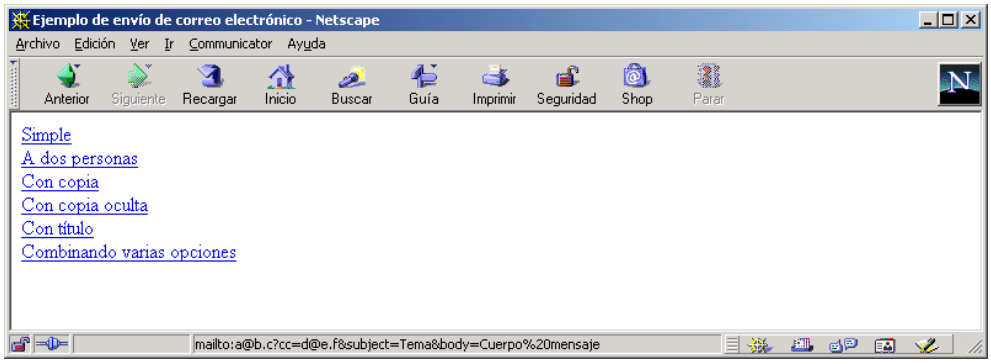


Figura 6.21: Envío de un correo electrónico mediante un enlace

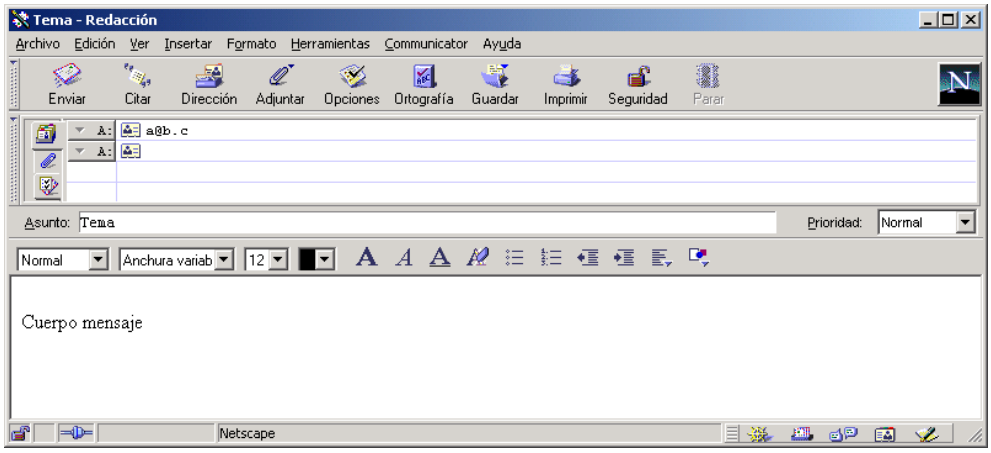


Figura 6.22: Mensaje visualizado en Netscape Messenger

- Las tablas en **HTML** se definen mediante los códigos pareados `<TABLE> ... </TABLE>`.
- Una tabla se compone de celdillas de datos. Una celdilla se define usando los códigos `<TD> ... </TD>`.
- Las celdillas se agrupan en filas, que se definen con los códigos `<TR> ... </TR>`.
- Pueden existir celdillas que se emplean como encabezados de filas o columnas, y se definen con los códigos `<TH> ... </TH>`. Este tipo de celdas suele aparecer diferenciada de las otras, normalmente en negrita.
- Las celdillas pueden contener cualquier elemento **HTML**: texto, imágenes, enlaces e incluso otras tablas anidadas.

El siguiente ejemplo muestra una tabla con bordes formada por nueve celdas, de las que tres forman parte del encabezado. La tabla se ha dividido en tres filas y tres columna. En la Figura 6.23 vemos como se ve esta página en una navegador.

Ejemplo 6.26

```
<HTML>
<BODY>
<TABLE BORDER="1">
<TR>
  <TH>Cabecera 1</TH>
  <TH>Cabecera 2</TH>
  <TH>Cabecera 3</TH>
</TR>
<TR>
  <TD>Elemento (1, 1)</TD>
  <TD>Elemento (1, 2)</TD>
  <TD>Elemento (1, 3)</TD>
</TR>
<TR>
  <TD>Elemento (2, 1)</TD>
  <TD>Elemento (2, 2)</TD>
  <TD>Elemento (2, 3)</TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

Los atributos más importantes de la etiqueta `<TABLE>` son:

- `BGCOLOR="color"`. Color de fondo de la tabla.

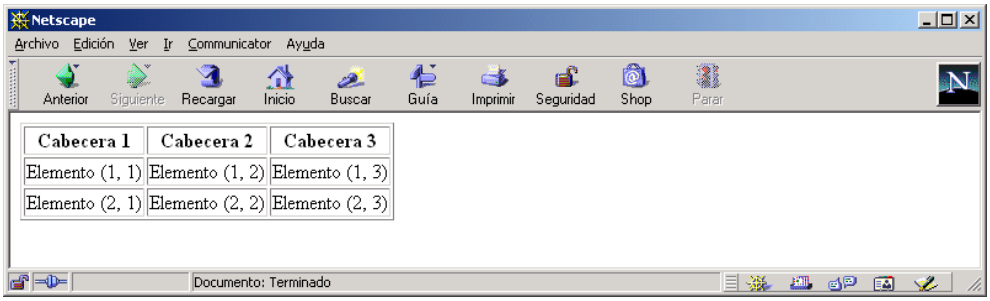


Figura 6.23: Tabla sencilla

- `BORDER="n"`. Indica el grosor del borde exterior de la tabla y de las líneas delimitadoras interiores. Un valor 0 produce que no tenga borde.
- `CELLPADDING="n"`. Determina el espacio que debe existir entre los bordes de cada celdilla y el contenido de la celda.
- `CELLSPACING="n"`. Especifica la cantidad de espacio que debe existir entre celdas contiguas.
- `HEIGHT="n"` | `"n%"`. Alto de la tabla. Se puede indicar en pixels o en tanto por ciento en relación a la altura total disponible.
- `WIDTH="n"` | `"n%"`. Ancho de la tabla. Se puede indicar en pixels o en tanto por ciento en relación a la anchura total disponible.

Los atributos más importantes de las etiquetas `<TR>`, `<TH>` y `<TD>` son:

- `ALIGN="LEFT"` | `"CENTER"` | `"RIGHT"`. Alineamiento horizontal del contenido de una fila (`<TR>`) o de una celda (`<TH>` y `<TD>`).
- `BGCOLOR="color"`. Color de fondo de una celda.
- `VALIGN="BASELINE"` | `"BOTTOM"` | `"MIDDLE"` | `"TOP"`. Alineamiento vertical del contenido de una una fila (`<TR>`) o de una celda (`<TH>` y `<TD>`).

6.12.1. Fusión de filas y columnas

En ocasiones interesa que las tablas no tengan el mismo número de filas y de columnas, sino que se fusionen las celdas de algunas filas y/o columnas. Para ello se emplean los atributos `COLSPAN` y `ROWSPAN` de las etiquetas `<TD> ... </TD>` y `<TD> ... </TD>`.

Si se quieren fusionar varias celdas de columnas contiguas, se indica mediante `COLSPAN="n"`, donde `n` es el número de columnas que ocupa una determinada celda. Si se quieren fusionar varias celdas de filas contiguas, se indica mediante `ROWSPAN="n"`, donde `n` es el número de filas que ocupa una determinada celda.

El siguiente ejemplo muestra una tabla con varias celdas fusionadas. En la Figura 6.24 vemos como se ve esta página en una navegador.

Ejemplo 6.27

```
<HTML>
<BODY>
<BR><BR>
<CENTER>
<TABLE BORDER="1">
<TR>
  <TD>Una celda normal</TD>
  <TD>Una celda normal</TD>
  <TD>Una celda normal</TD>
</TR>
<TR>
  <TD COLSPAN="2">Una celda fusionada con otra celda
  mediante COLSPAN</TD>
  <TD>Una celda normal</TD>
</TR>
<TR>
  <TD COLSPAN="3">Una celda fusionada con otras dos celdas
  mediante COLSPAN</TD>
</TR>
<TR>
  <TD ROWSPAN="2">Una celda fusionada con otra celda
  mediante ROWSPAN</TD>
  <TD>Una celda normal</TD>
  <TD>Una celda normal</TD>
</TR>
<TR>
  <TD>Una celda normal</TD>
  <TD>Una celda normal</TD>
</TR>
<TR>
  <TD ROWSPAN="3">Una celda fusionada con otras dos celdas
  mediante ROWSPAN</TD>
  <TD>Una celda normal</TD>
  <TD>Una celda normal</TD>
</TR>
<TR>
  <TD>Una celda normal</TD>
  <TD>Una celda normal</TD>
```

```
39 </TR>
40 <TR>
41   <TD>Una celda normal</TD>
42   <TD>Una celda normal</TD>
43 </TR>
44 </TABLE>
45 </CENTER>
46 </BODY>
47 </HTML>
```

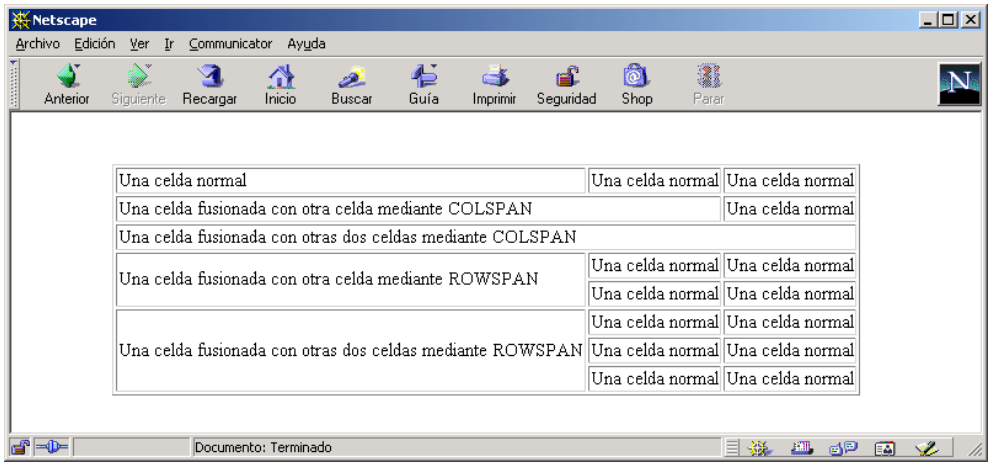


Figura 6.24: Tabla fusión de varias celdas

6.12.2. Tablas invisibles

Se conoce como tablas invisibles a aquellas que no poseen borde (`BORDER="0"`). Las tablas invisibles son muy útiles para distribuir los distintos elementos en una página **HTML**.

Por ejemplo, mediante tablas invisibles se puede mostrar el texto con márgenes a la izquierda y a la derecha, mostrar texto a varias columnas, dividir una imagen en diferentes ficheros y que se muestre como si no estuviese dividida, etc.

6.12.3. Alineamiento del contenido de una tabla

El contenido de una tabla se puede alinear de forma horizontal o vertical. El alineamiento horizontal se indica con el atributo `ALIGN`, que puede tomar los valores

LEFT (a la izquierda), CENTER (centrado) y RIGHT (a la derecha). El alineamiento vertical se indica con el atributo VALIGN, que puede tomar los valores BOTTOM (abajo del todo), BASELINE (en la línea base), MIDDLE (en el medio) y TOP (arriba del todo).

Estos dos atributos se pueden aplicar tanto a la etiqueta <TR> ... </TR> como a la etiqueta <TD> ... </TD>. Si se aplica a ambas etiquetas a la vez, el alineamiento que se emplea es el indicado por la etiqueta <TD> ... </TD>.

El siguiente ejemplo muestra una tabla con distinto alineamiento horizontal y vertical. En la Figura 6.25 vemos como se muestra esta página en una navegador.

Ejemplo 6.28

```
<HTML>
<BODY>
<BR><BR>
<CENTER>
<TABLE BORDER="1">
<TR ALIGN="CENTER">
  <TD>Varias<BR>líneas<BR>de texto</TD>
  <TD>Alineamiento CENTER</TD>
  <TD>Alineamiento CENTER</TD>
  <TD>Alineamiento CENTER</TD>
</TR>
<TR ALIGN="RIGHT" VALIGN="TOP">
  <TD>Varias<BR>líneas<BR>de texto</TD>
  <TD>Alineamiento RIGHT y TOP</TD>
  <TD>Alineamiento RIGHT y TOP</TD>
  <TD>Alineamiento RIGHT y TOP</TD>
</TR>
<TR>
  <TD>Varias<BR>líneas<BR>de texto</TD>
  <TD ALIGN="LEFT">Alineamiento LEFT</TD>
  <TD ALIGN="CENTER">Alineamiento CENTER</TD>
  <TD ALIGN="RIGHT">Alineamiento RIGHT</TD>
</TR>
<TR>
  <TD>Varias<BR>líneas<BR>de texto</TD>
  <TD VALIGN="TOP">Alineamiento TOP</TD>
  <TD VALIGN="MIDDLE">Alineamiento MIDDLE</TD>
  <TD VALIGN="BOTTOM">Alineamiento BOTTOM</TD>
</TR>
<TR VALIGN="MIDDLE">
  <TD>Varias<BR>líneas<BR>de texto</TD>
  <TD VALIGN="TOP">Alineamiento TOP</TD>
  <TD VALIGN="MIDDLE">Alineamiento MIDDLE</TD>
  <TD VALIGN="BOTTOM">Alineamiento BOTTOM</TD>
</TR>
</TABLE>
```

```
37 </CENTER>
38 </BODY>
39 </HTML>
```

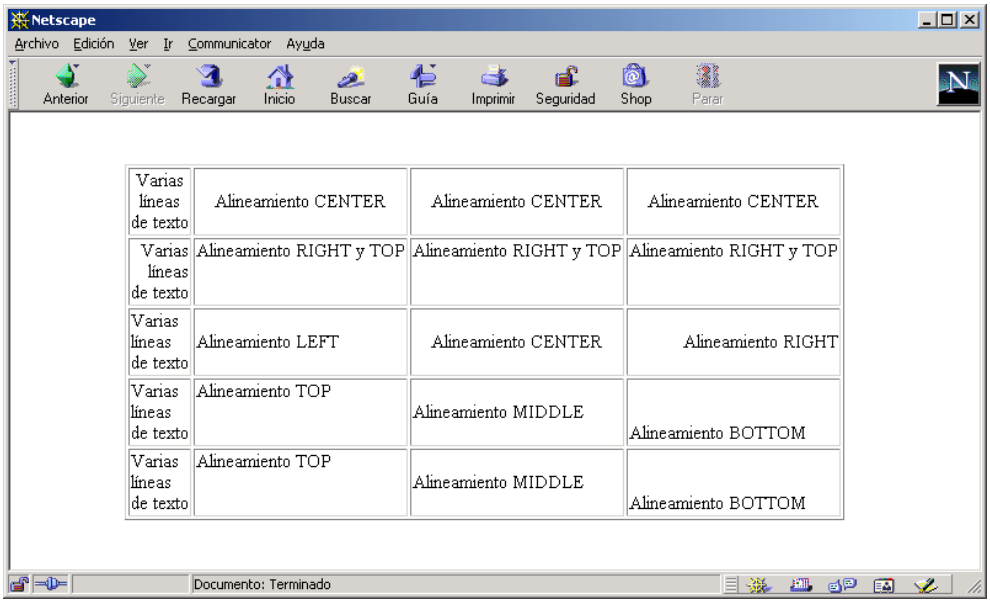


Figura 6.25: Alineamiento del contenido de una tabla

6.12.4. Distancia entre celdas

La distancia entre el borde de una celda y su contenido, y la distancia entre los bordes de dos celdas contiguas se puede configurar. Desgraciadamente, no se puede configurar únicamente para un conjunto de celdas de una tabla, sino que afecta a todas las celdas de una tabla.

El atributo `CELLPADDING="n"` de la etiqueta `<TABLE> ... </TABLE>` modifica la distancia, en pixels, existente entre el borde de una celda y su contenido. El valor por defecto es 1.

El atributo `CELLSPACING="n"` de la etiqueta `<TABLE> ... </TABLE>` modifica la distancia, en pixels, existente entre los bordes de dos celdas contiguas. El valor por defecto es 2.

El siguiente ejemplo muestra cuatro tablas con distintas distancias entre el borde y su contenido y entre celdas contiguas. En la Figura 6.26 vemos como se visualiza esta página en una navegador.

Ejemplo 6.29

```
<HTML>
<BODY>
<BR><BR>
<CENTER>
<TABLE BORDER="1" CELLPADDING="5">
<TR>
  <TD>Alineamiento CENTER</TD>
  <TD>Alineamiento CENTER</TD>
  <TD>Alineamiento CENTER</TD>
</TR>
<TR>
  <TD>Alineamiento RIGHT y TOP</TD>
  <TD>Alineamiento RIGHT y TOP</TD>
  <TD>Alineamiento RIGHT y TOP</TD>
</TR>
</TABLE>
<BR>
<TABLE BORDER="1" CELLPADDING="15">
<TR>
  <TD>Alineamiento CENTER</TD>
  <TD>Alineamiento CENTER</TD>
  <TD>Alineamiento CENTER</TD>
</TR>
<TR>
  <TD>Alineamiento RIGHT y TOP</TD>
  <TD>Alineamiento RIGHT y TOP</TD>
  <TD>Alineamiento RIGHT y TOP</TD>
</TR>
</TABLE>
<BR>
<TABLE BORDER="1" CELLSPACING="5">
<TR>
  <TD>Alineamiento CENTER</TD>
  <TD>Alineamiento CENTER</TD>
  <TD>Alineamiento CENTER</TD>
</TR>
<TR>
  <TD>Alineamiento RIGHT y TOP</TD>
  <TD>Alineamiento RIGHT y TOP</TD>
  <TD>Alineamiento RIGHT y TOP</TD>
</TR>
```

```
</TABLE>
<BR>
<TABLE BORDER="1" CELLSPACING="15">
<TR>
  <TD>Alineamiento CENTER</TD>
  <TD>Alineamiento CENTER</TD>
  <TD>Alineamiento CENTER</TD>
</TR>
<TR>
  <TD>Alineamiento RIGHT y TOP</TD>
  <TD>Alineamiento RIGHT y TOP</TD>
  <TD>Alineamiento RIGHT y TOP</TD>
</TR>
</TABLE>
</CENTER>
</BODY>
</HTML>
```

6.12.5. Tablas como marcos

Las tablas son muy útiles para crear marcos alrededor del texto o cualquier otro elemento de una página **HTML**. Se pueden conseguir efectos muy elegantes y a su vez sencillos de realizar mediante diversas tablas anidadas. Por ejemplo, el siguiente código, cuyo resultado se muestra en la Figura 6.27, muestra un texto sobre un fondo rojo rodeado de un marco amarillo en una página cuyo fondo es de color naranja. El tamaño de las celdas de la tabla exterior se ha modificado mediante el atributo `CELLPADDING="10"` para obtener un marco más ancho.

Ejemplo 6.30

```
<HTML>
<BODY BGCOLOR="orange">
<BR>
<CENTER>
<TABLE BORDER="0" BGCOLOR="yellow" CELLPADDING="10">
<TR><TD>
  <TABLE BORDER="0" BGCOLOR="red">
  <TR><TD>
  <FONT SIZE="5">HTML útil y práctico</FONT>
  </TD></TR>
  </TABLE>
</TD></TR>
</TABLE>
</CENTER>
```

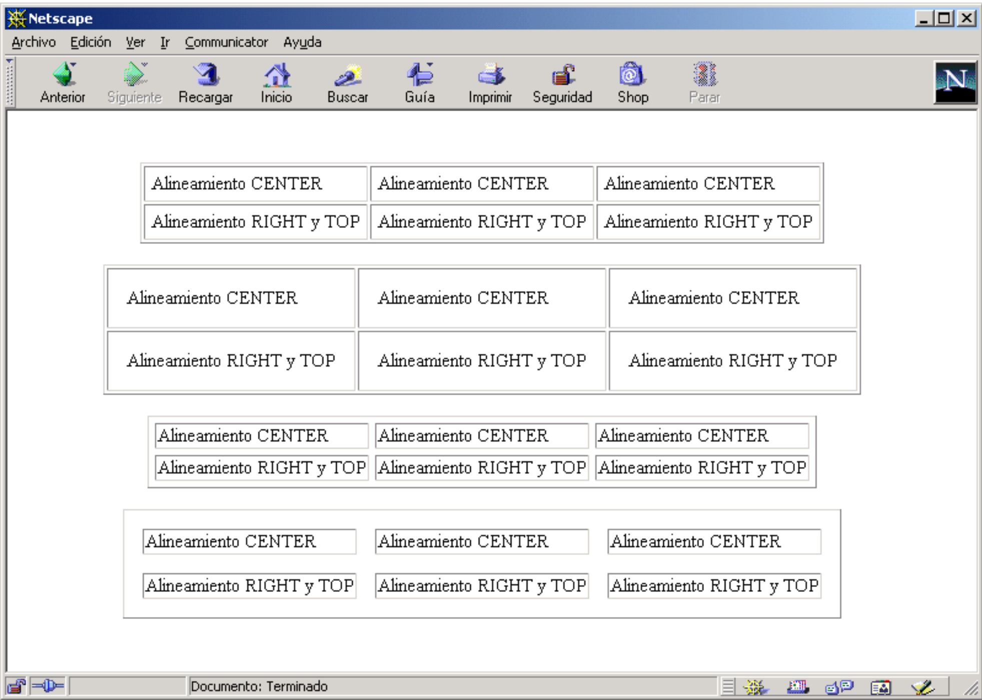


Figura 6.26: Distintas distancias entre celdas

```
15 </BODY>
16 </HTML>
```

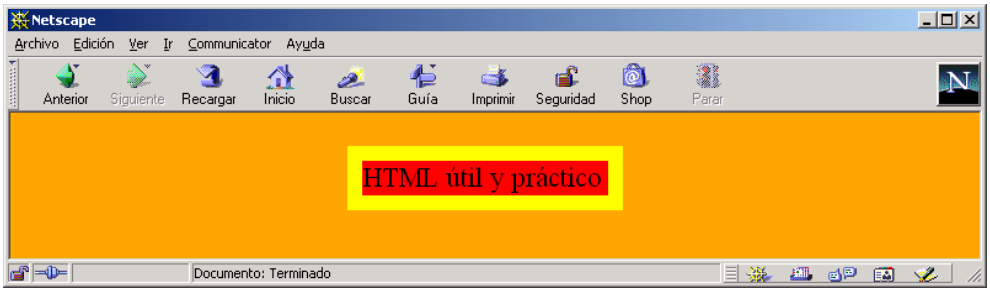


Figura 6.27: Tablas como marcos

6.13. Imágenes

Al inicio de la Web, el empleo de imágenes en los documentos **HTML** era muy escaso[20]. Sin embargo, hoy es todo lo contrario y es muy difícil encontrar páginas que no empleen una gran cantidad de imágenes.

Los dos formatos de imagen que admiten la mayoría de los navegadores son *Graphics Interchange Format* (**GIF**) y **JPEG**, pero ya se empieza a usar cada vez más el nuevo formato **PNG**. Las características básicas de los tres formatos se han resumido en el Cuadro 6.4. Colores indica el máximo número de colores que permite el formato gráfico; transparencia indica si soporta transparencias (zonas de una imagen que muestran lo que está situado detrás de ella); animación indica si soporta animaciones; compresión indica si emplea un formato de compresión sin pérdidas o con pérdidas; dibujo indica si es un formato gráfico adecuado para almacenar imágenes sencillas, como dibujos, logotipos, textos, etc.; fotografía indica si es un formato gráfico adecuado para almacenar fotografías; por último, gamma indica si soporta el empleo de la corrección gamma cuando se visualiza la imagen.

[20] Las razones pueden ser varias: los ordenadores no tenían la suficiente potencia para manejar varias imágenes a la vez, el ancho de banda en las comunicaciones era menor que el actual o a nadie se le había ocurrido la idea de hacer un uso "intensivo y extensivo" de las imágenes.

Característica	GIF	JPEG	PNG
Colores	256 (8 bits)	16 777 216 (24 bits)	48 bits
Transparencia	Sí	No	Sí (alfa)
Animación	Sí	No	No
Compresión	Sin pérdidas	Con pérdidas	Sin pérdidas
Dibujo	Sí	No	Sí
Fotografía	No	Sí	Sí
Gamma	No	No	Sí

Cuadro 6.4: Diferencias entre GIF, JPEG y PNG

6.13.1. Archivos GIF

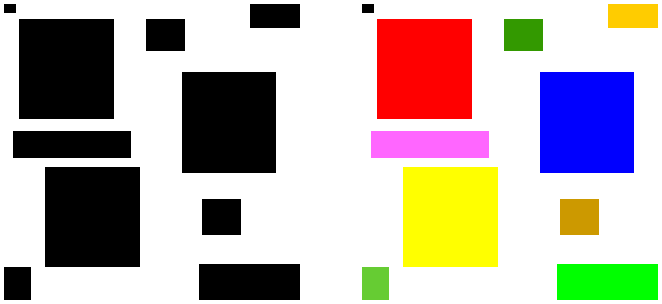
Este formato gráfico es uno de los más populares en Internet. Emplea un esquema de compresión sin pérdidas[21] para reducir su tamaño, y la paleta de color se limita a 8 bits, lo que permite un máximo de 256 colores. Además, permite definir un color como transparente y la forma de descarga en el navegador se puede definir como normal o entrelazada. En el modo entrelazado, las líneas que forman la imagen no se almacenan secuencialmente, sino entrelazadas por grupos, como por ejemplo, grupo 1 (líneas 1, 5, 9, ...), grupo 2 (líneas 2, 6, 10, ...), etc. De este modo, conforme se descarga la imagen se van visualizando las líneas entrelazadas, lo que proporciona una idea global de la imagen final. Por último, existen distintas versiones de este formato gráfico, siendo las más empleadas GIF87A y GIF89A.

El esquema de compresión empleado es *Lempel Ziv Welch* (**LZW**). Este sistema funciona mucho mejor en imágenes con zonas de color homogéneo, ya que es menos eficaz en imágenes complejas con muchos colores y texturas complejas. Cuantos menos colores tiene una imagen, mejor funciona el esquema de compresión empleado por **GIF**. Por ejemplo, en la Figura 6.28 se muestra la misma imagen en blanco/negro y en color: la primera ocupa 1 276 bytes y la segunda 1 478 bytes.

El formato **GIF** también permite definir un color como transparente. De esta forma, las zonas de la imagen que posean ese color serán transparentes y se podrá ver lo que esté detrás de la imagen. Normalmente se asigna la transparencia al color de fondo de la imagen. La propiedad de transparencia no es selectiva, lo que significa que si se asigna la transparencia a un color, todos los puntos de la imagen que contengan ese color se convertirán en transparentes, lo que puede ocasionar problemas en algunos casos.

Otros problemas que pueden surgir con las transparencias ocurren cuando en la imagen se ha aplicado un suavizado (*antialiasing*). El suavizado modifica los bordes insertando colores intermedios en los límites de las formas como pueden ser las letras

[21] Compresión de la información en la que todos los datos iniciales se almacenan, por lo que la calidad de las imágenes no se ve afectada al recuperar las imágenes una vez comprimidas.



(a) (b)

Figura 6.28: La misma imagen GIF en blanco/negro y en color

o las líneas. Tiene como fin mejorar la apariencia de las imágenes, ya que suaviza los "dientes de sierra". Cuando se intenta usar una imagen con suavizado y fondo transparente sobre un color de fondo distinto al empleado durante el suavizado ocurren problemas, ya que el suavizado aparece como un halo alrededor de los límites de las formas de la imagen. Por ejemplo, en la Figura 6.29 aparece una imagen con suavizado sobre un fondo blanco. Cuando la misma imagen se coloca sobre un fondo de otro color, aparecen distintas zonas que corresponden al suavizado, tal como se puede apreciar en la Figura 6.30.

Figura 6.29: GIF transparente sobre fondo blanco

Figura 6.30: GIF transparente sobre fondo no blanco

Los anuncios que se insertan en las páginas web, llamados pancartas o banderolas (*banners*) suelen emplear el formato **GIF**, ya que permite realizar animaciones. The Interactive Advertising Bureau, una organización que agrupa a anunciantes en

Internet, posee una serie de guías[22] sobre el tamaño de los banners. A continuación se incluyen los tamaños que han sido estandarizados y en la Figura 6.31 se pueden ver algunos de ellos:

- 468 x 60 (*Full Banner*)
- 234 x 60 (*Half Banner*)
- 88 x 31 (*Micro Bar*)
- 120 x 90 (*Button 1*)
- 120 x 60 (*Button 2*)
- 160 x 600 (*Wide Skyscraper*)
- 120 x 600 (*Skyscraper*)
- 125 x 125 (*Square Button*)
- 120 x 240 (*Vertical Banner*)
- 180 x 150 (*Rectangle*)
- 300 x 250 (*Medium Rectangle*)
- 250 x 250 (*Square Pop-up*)
- 240 x 400 (*Vertical Rectangle*)
- 336 x 280 (*Large Rectangle*)

6.13.2. Archivos JPEG

El formato gráfico **JPEG** es uno de los más empleados en Internet debido a que permite comprimir enormemente las imágenes fotográficas. A diferencia de **GIF**, las imágenes **JPEG** son imágenes a todo color (24 bits por punto de la imagen). Además, si se emplea **JPEG** progresivo (mecanismo similar al entrelazado en el formato **GIF**) se pueden descargar progresivamente las imágenes, lo que facilita una previsualización rápida de las imágenes.

El esquema de compresión de **JPEG** es una sofisticada técnica matemática denominada transformación discreta de cosenos. Mediante esta técnica se pueden elegir distintos grados de compresión y de calidad: cuanto más se comprime una imagen, menor es su calidad y vice versa. **JPEG** emplea compresión con pérdidas: los datos

[22] `http://www.iab.net/iab_banner_standards/bannersizes.html`.

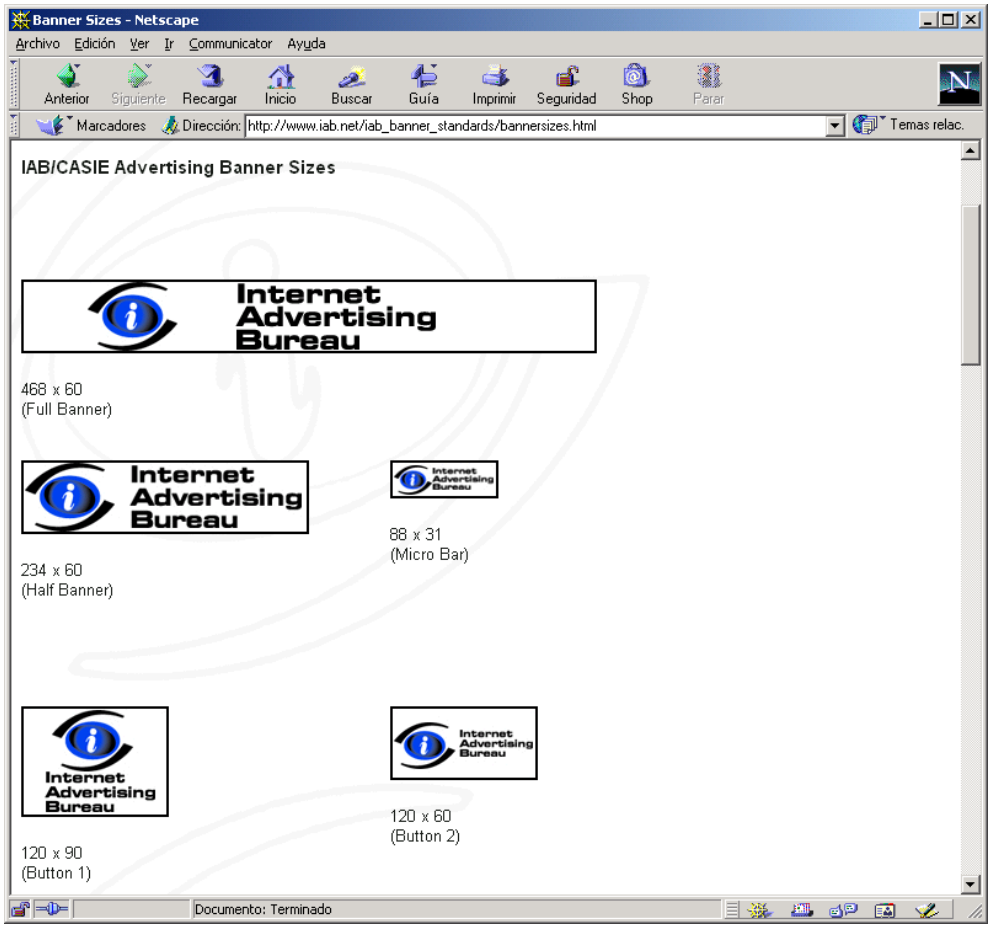


Figura 6.31: Distintos tamaños de pancartas (banners)

no necesarios se descartan a medida que se comprime cada vez más la imagen[23]. Si se selecciona correctamente el grado de compresión, la reducción del tamaño del fichero que contiene la imagen compensa con creces la inapreciable pérdida de calidad.

A medida que se aumenta el nivel de compresión, aparecen varios problemas en las imágenes **JPEG**: el ruido, la distorsión y la textura cuadriculada. Por ejemplo, en la Figura 6.32 se puede ver a la izquierda una imagen en su formato original (sin pérdidas) y a la derecha la palabra "en" ampliada para apreciar la calidad de la imagen. Esta imagen de 100 x 300 ocupa 31 078 bytes en formato *Bit-map* (**BMP**). En la Figura 6.33 y Figura 6.34 se muestra la misma imagen comprimida con dos niveles de compresión distintos: en la Figura 6.33 el nivel de compresión es 15 sobre 100 (10 281 bytes) y en la Figura 6.34 75 sobre 100 (3 643 bytes). En las imágenes de la izquierda prácticamente no se aprecian diferencias, pero cuando se amplia la imagen se puede observar la aparición de ruido, los bordes borrosos y la textura cuadriculada.

JPEG es un formato apto para imágenes fotográficas o ilustraciones con apariencia de fotografía: imágenes complejas, con suaves transiciones de tono y color y sin bordes muy marcados. Para las imágenes con formas geométricas, bordes marcados y transiciones de color acentuadas, como pueden ser los esquemas, logotipos o dibujos, es mejor no emplear este formato gráfico.

6.13.3. Archivos PNG

El formato gráfico **PNG** ha sido desarrollado por **W3C** como una alternativa de carácter público al formato **GIF** tradicional[24]. Este formato gráfico se ha creado específicamente para Internet y otras redes de ordenadores. Sus características más importantes son:

- Soporte de imágenes basadas en paleta (1, 2, 4, 8-bit), como **GIF**.
- Soporte de escala de grises (1, 2, 4, 8, 16-bit).
- Soporte de color real (24, 48-bit), como **JPEG**.
- Transparencia binaria, como **GIF**.
- Transparencia alfa (256 o 65 536 niveles de transparencia parcial).
- Corrección gamma.
- Algoritmo de compresión sin pérdidas no patentado (público).

[23] Si se trabaja con **JPEG**, es recomendable conservar una copia de la imagen original en un formato que no produzca pérdidas, ya que una vez comprimida la imagen con **JPEG** es imposible recuperar los datos perdidos y, por tanto, la calidad original de la imagen.

[24] El formato **GIF** lo desarrolló Compuserve a partir de un esquema de compresión privado (**LZW**) propiedad de Unisys Corporation. Todo programador que emplee el formato **GIF** "debe pagar" los correspondientes derechos de propiedad a ambas compañías.

(a) (b)

Figura 6.32: Imagen en formato PNG y detalle

(a) (b)

Figura 6.33: Imagen en formato JPG (alta calidad) y detalle

(a) (b)

Figura 6.34: Imagen en formato JPG (baja calidad) y detalle

- Entrelazado 2D (*interlacing*).

El formato **PNG** no permite animaciones como **GIF**, pero se ha desarrollado un nuevo formato basado en **PNG** que sí que permite crear animaciones.

El esquema de compresión de **PNG** es muy superior al de **GIF**. Por ejemplo, las dos imágenes de la Figura 6.28 ocupan con este formato 248 bytes y 279 bytes respectivamente, lo que supone una reducción de aproximadamente el 80 % en el tamaño del fichero.

Aunque Microsoft Internet Explorer y Netscape Communicator en sus últimas versiones (a partir de 4.x) ya lo soportan parcialmente, aún no se encuentra muy extendido su uso. Además, algunas características no se implementan correctamente. Por ejemplo, en la Figura 6.35, 6.36 y 6.37 se puede ver la misma imagen **PNG** visualizada mediante Netscape Communicator 4.78 (donde peor se ve), Microsoft Internet Explorer 5.5 y Opera 6.0 (donde mejor se ve). Esta imagen posee transparencia alfa, pero esta característica no se visualiza correctamente en todos los navegadores:

- En Netscape Communicator se ve una barra horizontal con un degradado y las letras tienen un color sólido porque no se tienen en cuenta la transparencia. Además en las letras (como la 'e' y la 'g') se pueden observar dientes de sierra, ya que al no realizarse la transparencia no se puede hacer *antialiasing*.

- En Microsoft Internet Explorer se puede observar como el color de las letras no es sólido y la barra horizontal ha desaparecido: ahora hay tres barras más pequeñas y el degradado ha cambio. Esto se debe a que la transparencia tiene efecto con el color de fondo de la imagen.

- En Opera la transparencia se realiza con la imagen de fondo y no con el color de fondo. Se puede observar como la textura de la imagen de fondo se ve a través de las letras.

6.13.4. Etiqueta <IMG>

La etiqueta **HTML** que permite insertar una imagen en un documento es `<IMG>` (*image*). Una imagen se puede colocar en cualquier punto de un documento: en un enlace, en una tabla, etc. Los atributos más importantes de esta etiqueta son:

- `SRC="localización"`. Indica la localización y el nombre de la imagen que se quiere insertar.

- `BORDER="anchura"`. Anchura del borde que se crea alrededor de la imagen. Cuando se coloca una imagen en un enlace, automáticamente se crea un borde; si no se quiere que aparezca el borde, hay que asignar el valor 0 a este atributo.

- `WIDTH="anchura"`. Anchura de la imagen.

Figura 6.35: Imagen PNG con transparencias visualizada en Netscape Communicator 4.78

Figura 6.36: Imagen PNG con transparencias visualizada en Microsoft Internet Explorer 5.5

Figura 6.37: Imagen PNG con transparencias visualizada en Opera 6.0

- `HEIGHT="altura"`. Altura de la imagen.
- `ALT="texto"`. Texto alternativo que se muestra si el navegador no admite la etiqueta `<IMG>` o se ha interrumpido la carga de imágenes.[25]
- `ALIGN="LEFT" | "RIGHT" | "TOP" | "ABSMIDDLE" | "ABSBOTTOM" | "TEXTTOP" | "MIDDLE" | "BASELINE" | "BOTTOM"`. Especifica el alineamiento de la imagen con respecto al texto que la rodea.

Se recomienda indicar siempre la anchura y la altura de cada imagen con los atributos `WIDTH` y `HEIGHT`, ya que así la visualización de las páginas es más rápida[26].

El siguiente código muestra como se emplean las imágenes. La página está formada por una tabla con cuatro celdas. En cada celda se muestra la misma imagen con un alineamiento distinto. Además, la última imagen también posee un borde. En la Figura 6.38 se puede observar como se visualiza este código en un navegador.

Ejemplo 6.31

```
<HTML>
<BODY>
<TABLE BORDER="0">
<TR>
  <TD>
  <IMG SRC="foto.jpg" WIDTH="134" HEIGHT="191" ALIGN="TEXTTOP">
  El manitas de la casa
  </TD>
  <TD>
  <IMG SRC="foto.jpg" WIDTH="134" HEIGHT="191" ALIGN="MIDDLE">
  El manitas de la casa
  </TD>
</TR>
<TR>
  <TD>
  <IMG SRC="foto.jpg" WIDTH="134" HEIGHT="191" ALIGN="BOTTOM">
  El manitas de la casa
  </TD>
  <TD>
  <IMG SRC="foto.jpg" WIDTH="134" HEIGHT="191" BORDER="10">
  El manitas de la casa
  </TD>
</TR>
</TABLE>
```

[25] La gente con problemas de visión emplea un software especial que "lee" las páginas web; cuando encuentra una imagen, busca la etiqueta `ALT` y lee su contenido.

[26] El navegador conoce el tamaño de las imágenes antes de cargarlas, por lo que ya puede reservar el correspondiente espacio en el diseño de la página.

```
25 </BODY>
26 </HTML>
```

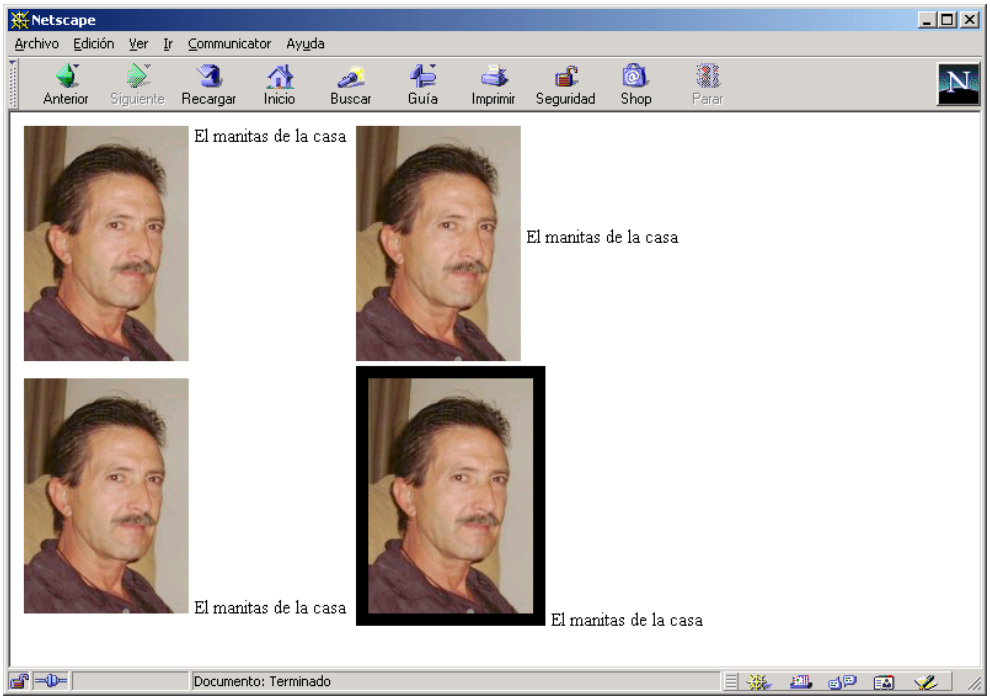


Figura 6.38: Imágenes con distinto alineamiento del texto

6.13.5. Imágenes como fondo de una página

La etiqueta `<BODY> ... </BODY>` posee el atributo `BACKGROUND` que permite mostrar una imagen como fondo de una página. Si la imagen tiene un tamaño menor que la ventana del navegador, la imagen se muestra en forma de "mosaico" hasta cubrir toda la superficie de la ventana.

6.14. Formularios

Los formularios son la herramienta que ofrece **HTML** para poder obtener información de un usuario que visita una página **HTML** y enviarla al servidor web para su procesamiento.

Un formulario contiene dos tipos de elementos básicos: campos de datos (cuadros de texto, listas de selección, casillas de verificación) y de control (botones).

Dentro de una página **HTML** se puede incluir más de un formulario, pero teniendo en cuenta que no pueden anidarse ni solaparse. El servidor web sólo podrá recibir la información introducida en uno de ellos (sólo se envía la información de uno de los formularios al servidor).

La estructura básica de un formulario es:

Ejemplo 6.32

```
1 <FORM NAME="nombre" ACTION="pagina.html" METHOD="metodo">
2 Controles del formulario
3 </FORM>
```

El sentido de cada una de las líneas es:

- Línea 1: La etiqueta `<FORM>` marca el inicio del formulario. El atributo `NAME` asigna un nombre al formulario (para poder hacer referencia a él posteriormente), `ACTION` indica la dirección (**URL**) de la página o programa que procesa los datos del formulario en el servidor y `METHOD` indica el método que se va a utilizar para enviar los datos del formulario al servidor (`GET` o `POST`[27]).
- Línea 2: En esta sección se incluyen los controles que posee el formulario.
- Línea 3: Fin del formulario.

6.14.1. Controles de un formulario

Un formulario puede contener los siguiente controles:

- Botones (para enviar información, borrar y otras acciones): `<INPUT TYPE="SUBMIT">`, `<INPUT TYPE="RESET">`, `<INPUT TYPE="BUTTON">`.
- Imágenes que actúan como botones (para enviar información): `<INPUT TYPE="IMAGE">`.
- Campos de verificación: `<INPUT TYPE="CHECKBOX">`.
- Campos excluyentes (botones de radio): `<INPUT TYPE="RADIO">`.
- Campos de texto: `<INPUT TYPE="TEXT">`.

[27] Normalmente se utiliza `POST`, que indica que los datos se envíen por la entrada estándar. Si se utiliza `GET`, los datos se envían unidos a la **URL** y en el servidor se pueden recuperar a través de la variable de entorno `QUERY_STRING`.

- Campos de contraseña[28] (*password*): `<INPUT TYPE="PASSWORD">`.
- Campos ocultos: `<INPUT TYPE="HIDDEN">`.
- Envío de ficheros: `<INPUT TYPE="FILE">`.
- Listas de selección: `<SELECT> ... </SELECT>, <OPTION>`.
- Áreas de texto (campos de texto multilínea): `<TEXTAREA> ... </TEXTAREA>`.

Para que los datos introducidos en un formulario se envíen al servidor, todo formulario tiene que tener un botón de tipo `TYPE="SUBMIT"`, que envía automáticamente los datos. Este botón se puede sustituir por un botón normal `TYPE="BUTTON"`, pero entonces el envío se tiene que realizar manualmente mediante código de *script*.

Conviene dar un nombre a los campos que coloquemos en un formulario, ya que al enviar la información, ésta se transmite como pares nombre-valor. Para ello, todas las etiquetas de los controles poseen el atributo `NAME` para asignar un nombre al control, que deberá ser un nombre único[29], es decir, ningún otro control tendrá que tener el mismo nombre (excepto en el caso de los botones de radio), y el atributo `VALUE` para asignar un valor (todas las etiquetas tienen este atributo excepto `<INPUT TYPE="IMAGE">`, `<SELECT> ... </SELECT>` y `<TEXTAREA> ... </TEXTAREA>`). En los botones, el atributo `VALUE` modifica el texto que muestra el botón (la etiqueta del botón).

El siguiente código genera una página **HTML** con un formulario que contiene un campo de texto, un campo de contraseña, dos campos excluyentes con el mismo nombre (y por tanto sólo se puede elegir una de las dos opciones), dos campos de verificación, una lista de selección, un área de texto y dos botones (para enviar información y borrar). En la Figura 6.39 se muestra el formulario tal como se ve en un navegador.

Ejemplo 6.33

```
1  <HTML>
2  <HEAD>
3  <TITLE>Esto es una página HTML con formularios</TITLE>
4  </HEAD>
5  <BODY>
6  Esto es el cuerpo de una página HTML. Esta página posee un formulario:
7  <HR>
8  <FORM NAME="miFormulario" ACTION="procesa.asp" METHOD="POST">
9  Cuadro de texto: <INPUT TYPE="TEXT" NAME="control1a" VALUE="Algo">
10 <BR>
```

[28] Un campo de contraseña es idéntico a un campo de texto, pero los caracteres se ocultan mediante asteriscos.

[29] En realidad, pueden existir varios controles (incluso de distinto tipo) con el mismo nombre, pero cuando se procesen los datos en el servidor no se podrá saber de que control procede cada dato.

```
11 Cuadro de texto contraseña: <INPUT TYPE="PASSWORD" NAME="control1b">
12 <BR><BR>
13 Botones de radio:
14 <INPUT TYPE="RADIO" NAME="control2" VALUE="1">Opción 1
15 <INPUT TYPE="RADIO" NAME="control2" VALUE="2">Opción 2
16 <BR><BR>
17 Casilla de verificación:
18 <INPUT TYPE="CHECKBOX" NAME="control3a" VALUE="1">Opción 1
19 <INPUT TYPE="CHECKBOX" NAME="control3b" VALUE="2">Opción 2
20 <BR><BR>
21 Lista de selección:
22 <SELECT NAME="control4">
23   <OPTION VALUE="ali">Alicante</OPTION>
24   <OPTION VALUE="val">Valencia</OPTION>
25   <OPTION VALUE="cas">Castellón</OPTION>
26 </SELECT>
27 <BR><BR>
28 Area de texto: <TEXTAREA NAME="control5"></TEXTAREA>
29 <BR><BR>
30 <INPUT TYPE="SUBMIT" VALUE="Enviar">
31 <INPUT TYPE="RESET" VALUE="Borrar">
32 </FORM>
33 <HR>
34 </BODY>
35 </HTML>
```

6.14.2. Campos de verificación

Los campos de verificación (`<INPUT TYPE="CHECKBOX">`) poseen dos valores: activado y desactivado. Si al enviarse un formulario un campo de verificación está activo, se envía al servidor el valor indicado por `VALUE`. Distintos campos de verificación pueden tener el mismo nombre (`NAME`), aunque no es lo usual ya que "complica" la programación de la aplicación web en el servidor.

Si se desea que por defecto un campo de verificación aparezca activado, se tiene que incluir el atributo `CHECKED` en la etiqueta `<INPUT TYPE="CHECKBOX">`.

6.14.3. Campos excluyentes

Los campos excluyentes o botones de radio (`<INPUT TYPE="RADIO">`) tienen sentido cuando se emplean varios a la vez. Un grupo de botones de radio está formado por varios botones de radio que tienen todos el mismo nombre (`NAME`). En un grupo de botones de radio sólo un botón de radio puede estar seleccionado en un instante. Un formulario puede contener distintos grupos de botones de radio.

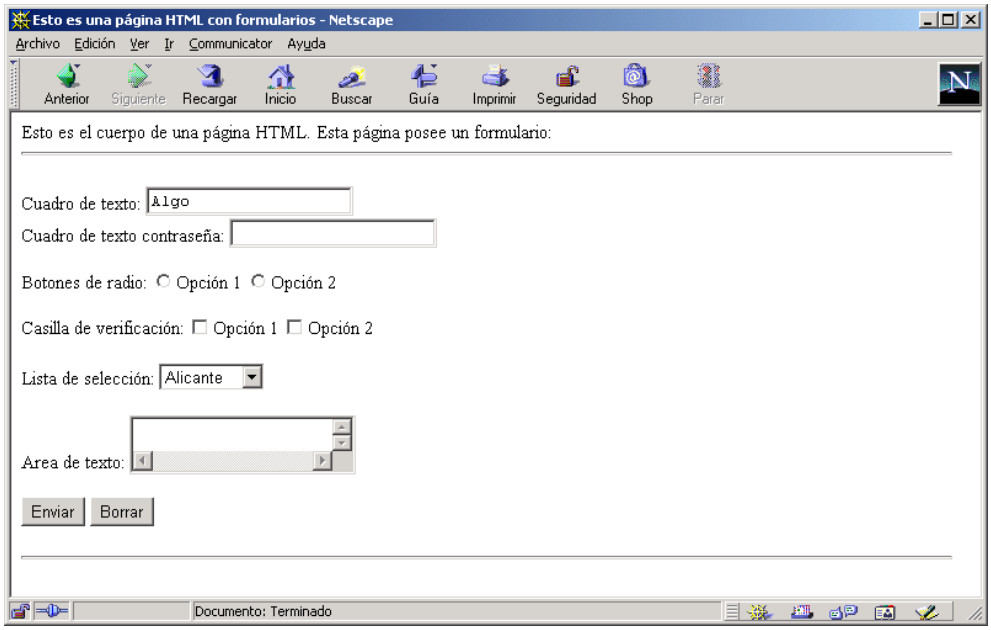


Figura 6.39: Formulario con distintos controles

Los botones de radio también poseen el atributo CHECKED que permite indicar un botón de radio seleccionado por defecto.

6.14.4. Campos de texto

En los campos de texto normal (<INPUT TYPE="TEXT">) y de contraseña (<INPUT TYPE="PASSWORD">) se puede escribir una cadena de caracteres. Se puede emplear el atributo SIZE para especificar el tamaño "visual" del cuadro de texto. Es decir, se puede indicar cuantos caracteres se pueden visualizar en un momento dado.

No confundir este atributo con MAXLENGTH, que especifica el número máximo de caracteres que se pueden introducir. Si no se especifica nada, se pueden introducir tantos caracteres como se desee.

Por último, para indicar un valor por defecto que tiene que contener el campo de texto cuando se visualice la página por primera vez se emplea el atributo VALUE.

6.14.5. Listas de selección

Las listas de selección se crean con la etiqueta <SELECT> ... </SELECT>. En las listas de selección se muestra una serie de opciones de las que el usuario puede elegir una. Si se añade el atributo MULTIPLE, el usuario puede elegir múltiples opciones[30]. El atributo SIZE permite indicar cuantas opciones de la lista se visualizan simultáneamente. Si el valor de este atributo es 1 (valor por defecto), se muestra una lista desplegable; si el valor de este atributo es mayor que 1, se muestra un cuadro de opciones con una barra de desplazamiento vertical.

Cada opción de una lista de selección se indica con la etiqueta <OPTION> ... </OPTION>[31]. Cada opción puede tener asociado un valor (VALUE), que es el valor que se enviará al servidor. Si se desea que una opción aparezca marcada por defecto se tiene que añadir a la opción el atributo SELECTED.

El siguiente ejemplo muestra tres listas: una lista normal, una lista normal que muestra tres opciones a la vez y posee una seleccionada por defecto y una lista múltiple. En la Figura 6.40 se puede observar como se visualiza esta página en un navegador.

Ejemplo 6.34

```
1 <HTML>
2 <BODY>
3 <FORM>
4 Lista de selección normal:
```

[30] Para seleccionar varias opciones se emplean la tecla **Control** para seleccionar de una en una y la tecla **Mays (⇑)** para seleccionar un conjunto contiguo de opciones (se marca la primera y la última).

[31] En **HTML** 4.01 la etiqueta de cierre </OPTION> es opcional, pero a partir de **XHTML** 1.0 es obligatoria.

```
<SELECT NAME="provincia">
  <OPTION VALUE="1">Alicante</OPTION>
  <OPTION VALUE="2">Valencia</OPTION>
  <OPTION VALUE="3">Castellón</OPTION>
</SELECT>
<BR><BR><BR><BR>
Lista de selección normal de tamaño 3:
<SELECT NAME="universidad" SIZE="3">
  <OPTION VALUE="uv">Universidad de Valencia</OPTION>
  <OPTION VALUE="uji">Universidad Jaime I</OPTION>
  <OPTION VALUE="ua" SELECTED>Universidad de Alicante</OPTION>
  <OPTION VALUE="upv">Universidad Politécnica de Valencia</OPTION>
  <OPTION VALUE="umh">Universidad Miguel Hernández</OPTION>
</SELECT>
<BR><BR><BR><BR>
Lista de selección múltiple:
<SELECT NAME="departamento" MULTIPLE>
  <OPTION VALUE="dlsi">D. de Lenguajes y Sistemas Informáticos</OPTION>
  <OPTION VALUE="damma">D. de Análisis M. y M. Aplicada</OPTION>
  <OPTION VALUE="dfists">D. de Física e Ingeniería de Sistemas</OPTION>
  <OPTION VALUE="dagr">D. de Análisis Geográfico Regional</OPTION>
  <OPTION VALUE="mmlab">Laboratorio Multimedia</OPTION>
</SELECT>
</FORM>
</BODY>
</HTML>
```

6.14.6. Áreas de texto

La etiqueta `<TEXTAREA> ... </TEXTAREA>` define un área de texto donde se pueden escribir varias líneas de texto. Esta etiqueta posee dos atributos que permiten modificar su tamaño. El atributo `COLS` indica el número de caracteres por línea que se pueden mostrar sin tener que realizar *scroll* horizontal. El atributo `ROWS` define el número de líneas que se pueden mostrar sin realizar *scroll* vertical.

Si se quiere que el área de texto muestre un texto por defecto, se puede incluir entre las etiquetas de inicio y fin. El siguiente ejemplo muestra dos áreas de texto de distinto tamaño, una de ellas con un texto por defecto. En la Figura 6.41 se puede ver esta página visualizada en un navegador. Se pueden observar las barras de desplazamiento vertical y horizontal.

Ejemplo 6.35

```
<HTML>
<BODY>
```

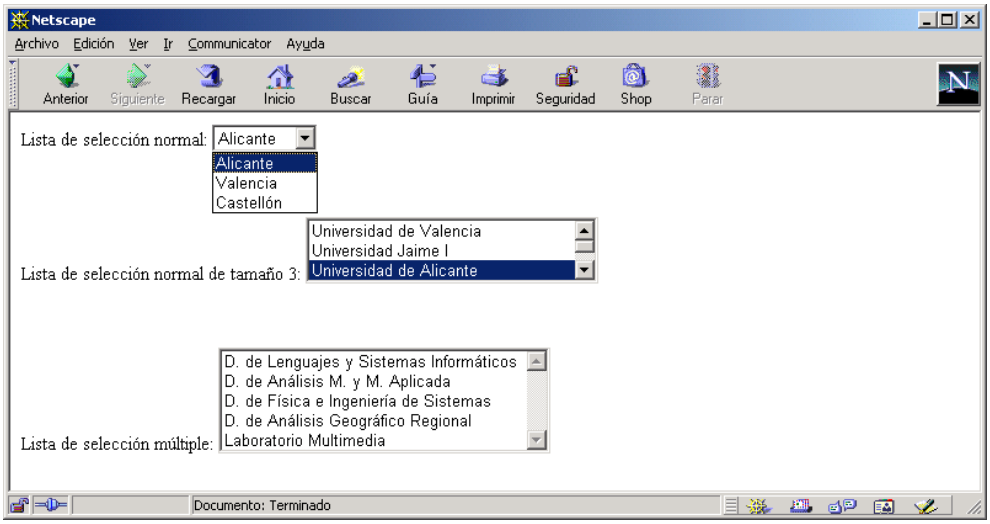


Figura 6.40: Distintas listas de selección

```
<FORM>
Área 1:
<TEXTAREA ROWS="2" COLS="40">Texto por defecto...</TEXTAREA>
<BR>
Área 2:
<TEXTAREA ROWS="4" COLS="20"></TEXTAREA>
</FORM>
</BODY>
</HTML>
```

6.14.7. Alineamiento de formularios

El alineamiento de los controles de un formulario es muy importante, ya que facilita su lectura y su utilización.

El siguiente ejemplo muestra un formulario sin alineamiento, donde cada control comienza en una posición horizontal diferente. En la Figura 6.42 se puede ver como se muestra esta página en un navegador. El formulario tiene una apariencia descuidada y su uso es difícil y lento, ya que los controles no están organizados.

Ejemplo 6.36

```
<HTML>
<BODY>
```

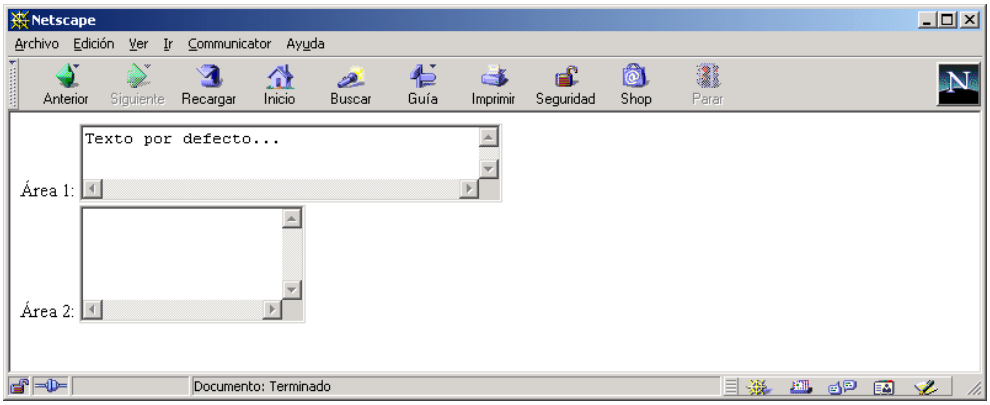


Figura 6.41: Áreas de texto de distinto tamaño

```
<FORM>
Nombre empresa: <INPUT TYPE="TEXT" NAME="nombreempresa" SIZE="40">
<BR>
Nombre: <INPUT TYPE="TEXT" NAME="nombre" SIZE="20">
<BR>
Apellidos: <INPUT TYPE="TEXT" NAME="apellidos" SIZE="30">
<BR>
Dirección: <INPUT TYPE="TEXT" NAME="direccion" SIZE="40">
<BR>
Población:
<SELECT NAME="poblacion">
  <OPTION VALUE="1">Alicante</OPTION>
  <OPTION VALUE="2">Castellón</OPTION>
  <OPTION VALUE="3">Valencia</OPTION>
</SELECT>
<BR>
Sexo:
<INPUT TYPE="RADIO" NAME="sexo" VALUE="H"> Hombre
<INPUT TYPE="RADIO" NAME="sexo" VALUE="M"> Mujer
<BR>
</FORM>
</BODY>
</HTML>
```

El siguiente ejemplo muestra el mismo formulario que antes, pero organizado gracias al empleo de una tabla con bordes invisibles. En la Figura 6.43 se puede observar como se visualiza en un navegador. Aunque la página ha variado poco, ahora el

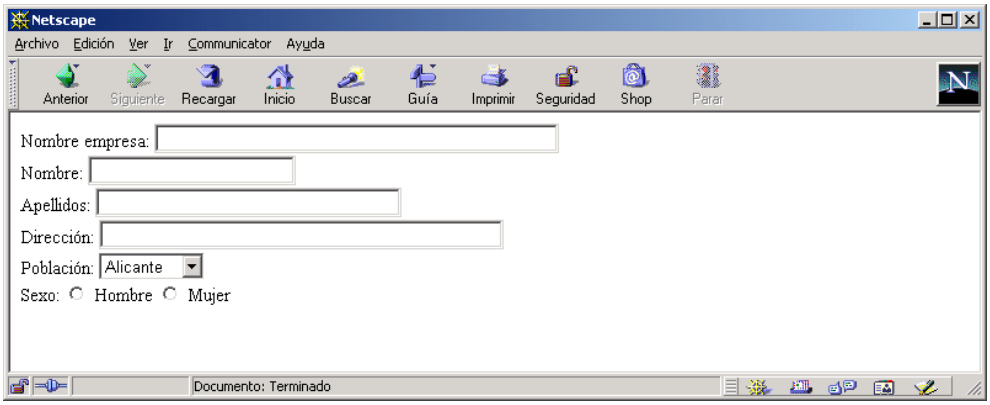


Figura 6.42: Formulario sin alineamiento de los controles

formulario tiene una apariencia más elegante y es más fácil de usar.

Ejemplo 6.37

```
<HTML>
<BODY>
<CENTER>
<FORM>
<TABLE BORDER="0">
<TR>
  <TD ALIGN="RIGHT">Nombre empresa:</TD>
  <TD><INPUT TYPE="TEXT" NAME="nombreempresa" SIZE="40"></TD>
</TR>
<TR>
  <TD ALIGN="RIGHT">Nombre:</TD>
  <TD><INPUT TYPE="TEXT" NAME="nombre" SIZE="20"></TD>
</TR>
<TR>
  <TD ALIGN="RIGHT">Apellidos:</TD>
  <TD><INPUT TYPE="TEXT" NAME="apellidos" SIZE="30"></TD>
</TR>
<TR>
  <TD ALIGN="RIGHT">Dirección:</TD>
  <TD><INPUT TYPE="TEXT" NAME="direccion" SIZE="40"></TD>
</TR>
<TR>
  <TD ALIGN="RIGHT">Población:</TD>
  <TD><SELECT NAME="poblacion">
    <OPTION VALUE="1">Alicante</OPTION>
```

```
26      <OPTION VALUE="2">Castellón</OPTION>
27      <OPTION VALUE="3">Valencia</OPTION>
28    </SELECT></TD>
29  </TR>
30  <TR>
31    <TD ALIGN="RIGHT">Sexo:</TD>
32    <TD><INPUT TYPE="RADIO" NAME="sexo" VALUE="H"> Hombre
33    <INPUT TYPE="RADIO" NAME="sexo" VALUE="M"> Mujer</TD>
34  </TR>
35  </TABLE>
36  </FORM>
37  </CENTER>
38  </BODY>
39  </HTML>
```

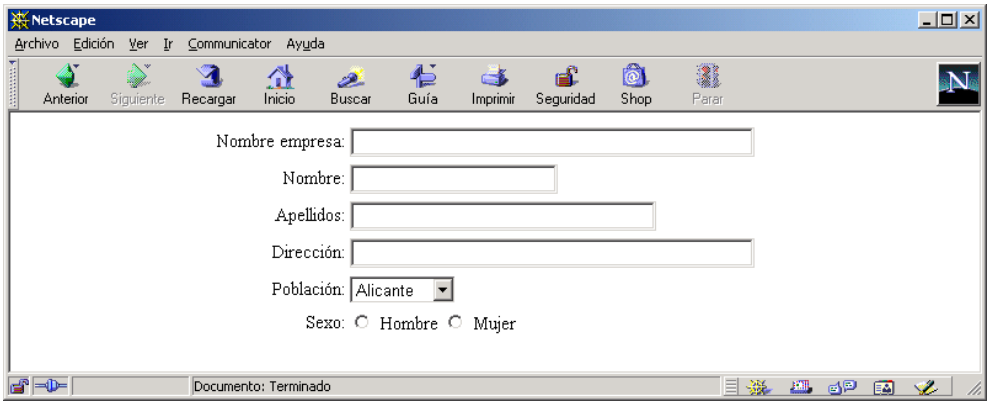


Figura 6.43: Formulario con alineamiento de los controles

6.15. Marcos

Un marco (*frame*) es una región de una ventana que actúa como si fuera una ventana ella misma. La ventana principal puede contener múltiples marcos, de forma que diferentes regiones de la ventana muestren diferentes contenidos. A su vez, un marco puede contener otros marcos y así sucesivamente.

Normalmente, los marcos se emplean para dividir la ventana del navegador en dos partes: en una de ellas, la más pequeña, se muestra un índice del sitio web (esta parte no varía); en la otra, la más grande, su contenido se modifica según los enlaces que se pulsen en el índice.

Para definir los marcos se emplean las tres etiquetas siguientes: <FRAMESET> ... </FRAMESET>, <FRAME> y <NOFRAMES> ... </NOFRAMES>.

La etiqueta <FRAMESET> ... </FRAMESET> define un conjunto de marcos que van a aparecer en la ventana. Esta etiqueta contiene una o más etiquetas <FRAME> que definen cada uno de los marcos. Los principales atributos de esta etiqueta son:

- BORDER="anchura". Anchura del borde de cada marco.
- FRAMEBORDER="YES" | "NO". Indica si el borde es en tres dimensiones o plano.
- COLS="listaAnchurasColumnas". Anchuras de cada uno de los marcos, separadas por comas. La anchura se puede indicar como número de pixels o como un porcentaje sobre el total disponible (en este caso, el número se tiene que acompañar del signo porcentaje). Por ejemplo, si se quiere dividir la ventana del navegador en dos columnas que ocupan el 30 y 70 por ciento:

Ejemplo 6.38

```
1 <FRAMESET COLS="30%,70%">
```

- ROWS="listaAlturasFilas". Alturas de cada uno de los marcos, separadas por comas. La altura se puede indicar como número de pixels o como un porcentaje sobre el total disponible (en este caso, el número se tiene que acompañar del signo porcentaje). Por ejemplo, si se quiere dividir la ventana del navegador en tres filas que ocupan el 10, 80 y 10 por ciento:

Ejemplo 6.39

```
1 <FRAMESET ROWS="10%,80%,10%">
```

La etiqueta <FRAMESET> ... </FRAMESET> define el número de filas (ROWS) o columnas (COLS) en que se va a dividir la ventana. Los dos atributos no se pueden emplear simultáneamente. Esta etiqueta se puede anidar, de forma que dentro de un <FRAMESET> se puede incluir otro <FRAMESET>. De este modo, se pueden combinar filas con columnas.

La etiqueta <FRAME> define un marco, que es una región de una ventana con contenido propio y que se puede navegar de forma independiente. Cada marco tiene su propia **URL** que define el contenido que se va a mostrar. Los atributos más importantes de esta etiqueta son:

- SRC="URL". Indica la **URL** del documento que se quiere mostrar en el marco.
- NAME="nombreMarco". Nombre del marco. Este nombre se emplea en la etiqueta <A> ... </A> para mostrar una página en un marco concreto.

- FRAMEBORDER="YES" | "NO". Igual que el atributo de la etiqueta <FRAMESET> ... </FRAMESET>.
- NORESIZE. Si aparece este atributo, no se puede modificar el tamaño del marco.
- SCROLLING="YES" | "NO" | "AUTO". Indica si pueden aparecer barras de desplazamiento cuando no quepa toda la página en el marco.

Por último, la etiqueta <NOFRAMES> ... </NOFRAMES> se emplea para mostrar contenido en los navegadores que no pueden mostrar marcos. Los navegadores que sí que pueden mostrar marcos ignoran todo el contenido de esta etiqueta.

Lo interesante de los marcos es que se puede variar el contenido de los mismos de forma independiente. Desde un marco se puede modificar el contenido de otro. Para ello, se tiene que emplear un enlace (<A> ... </A>) con el atributo TARGET="nombre". Este atributo debe de tomar el nombre del marco que se quiere modificar (el nombre se habrá indicado en la etiqueta <FRAME> con el atributo NAME).

En el siguiente ejemplo, la página frame1.html crea una venta con dos marcos (marcoIzq y marcoDer). En el marco izquierdo se muestra la página frame1a.html; en el marco derecho se muestran las páginas frame1b.html y frame1c.html según se seleccione en los enlaces que contiene marcoIzq. En la Figura 6.44 vemos el resultado de mostrar estas páginas en un navegador y en la Figura 6.45 se puede observar como cambia el marco de la derecha al pulsar el último enlace del marco de la izquierda.

- Página frame1.html:

Ejemplo 6.40

```
<HTML>
<HEAD>
<TITLE>Ejemplo de marcos</TITLE>
</HEAD>
<FRAMESET COLS="30%,70%" BORDER=10>
  <FRAME SRC="frame1a.html" NAME="marcoIzq">
  <FRAME SRC="frame1b.html" NAME="marcoDer">
  <NOFRAMES>
  Su navegador de Internet no permite mostrar marcos
  </NOFRAMES>
</FRAMESET>
</HTML>
```

- Página frame1a.html:

Ejemplo 6.41

```
<HTML>
<HEAD>
<TITLE>Ejemplo de marcos</TITLE>
```

```
</HEAD>
<BODY>
Esta página tiene que aparecer a la izquierda.
<BR><BR>
Si pincha <A HREF="frame1b.html" TARGET="marcoDer">aquí</A>,
a la derecha tiene que aparecer la página 1.
<BR><BR>
Si pincha <A HREF="frame1c.html" TARGET="marcoDer">aquí</A>,
a la derecha tiene que aparecer la página 2.
</BODY>
</HTML>
```

- Página `frame1b.html`:

Ejemplo 6.42

```
<HTML>
<HEAD>
<TITLE>Ejemplo de marcos</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
Esta es la página <B>1</B>.
<BR>
El color de fondo es blanco.
</BODY>
</HTML>
```

- Página `frame1c.html`:

Ejemplo 6.43

```
<HTML>
<HEAD>
<TITLE>Ejemplo de marcos</TITLE>
</HEAD>
<BODY BGCOLOR="#BBBBFF">
Esta es la página <B>2</B>.
<BR>
El color de fondo es azul.
</BODY>
</HTML>
```

6.15.1. Nombres de destinos especiales

En **HTML** se han definido unos nombres especiales que se pueden emplear en el atributo `TARGET` de cualquier enlace:

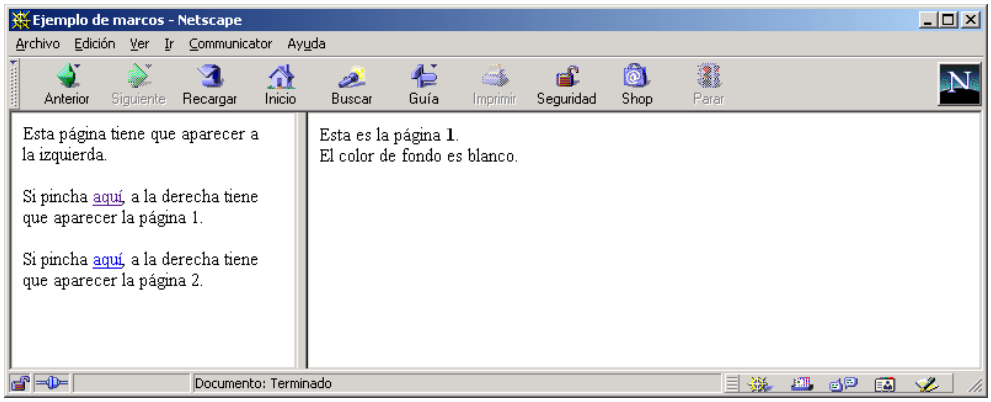


Figura 6.44: Página con dos marcos verticales

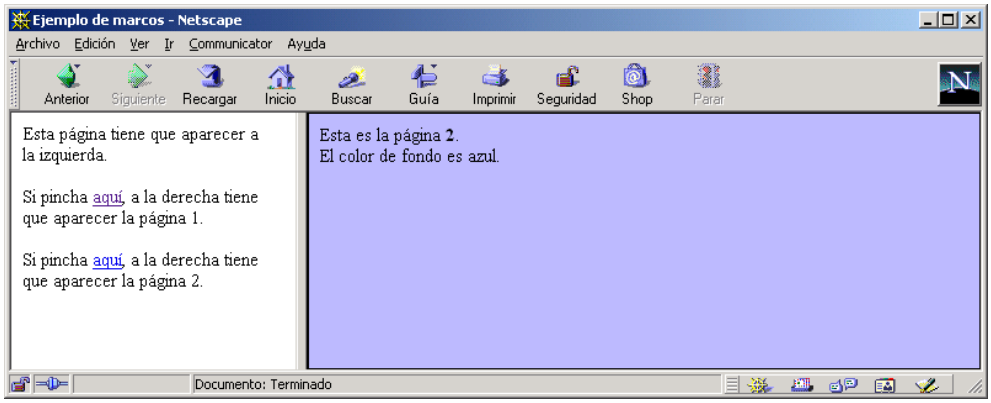


Figura 6.45: Página con dos marcos verticales

- `_blank`. Carga el nuevo documento en una nueva ventana sin nombre.
- `_self`. Carga el nuevo documento sobre la misma ventana (marco) en la que se ha pulsado el enlace.
- `_parent`. Carga el nuevo documento en la ventana (marco) padre de la ventana que muestra el documento actual.
- `_top`. Carga el nuevo documento en el nivel superior de la jerarquía de ventanas (marcos). De este modo, se pueden eliminar los marcos de una ventana.

6.15.2. Como evitar que cambie la dirección en el navegador al pulsar un enlace

Cuando se pulsa un enlace, el navegador envía una petición al servidor web solicitando la página que indica el enlace. Además, el navegador muestra en la barra de dirección la **URL** completa de la página nueva que se está solicitando. Sin embargo, por razones estéticas o de seguridad, a veces interesa que no cambie el contenido de la barra de dirección al pulsar sobre un enlace. Este efecto ocurre siempre que se trabaja con marcos. Pero, ¿y si no queremos emplear marcos en nuestras páginas? Tal como muestra el siguiente ejemplo, podemos crear una página con dos marcos, donde uno de ellos tiene un tamaño 0 y por tanto es invisible. A menos que el usuario consulte el código de la página, no se dará cuenta de que la página está dividida en dos marcos.

Ejemplo 6.44

```
<HTML>
<FRAMESET ROWS="0,*" BORDER="0">
  <FRAME>
  <FRAME SRC="pagina.html">
</FRAMESET>
</HTML>
```

6.15.3. El atributo TARGET en un formulario

La etiqueta `<FORM> ... </FORM>` empleada para crear formularios también posee el atributo `TARGET="nombre"`, que permite indicar el nombre del marco o de la ventana en el que se desea mostrar la página resultado del envío de un formulario.

Capítulo 7

Guía de estilo

En este capítulo se presentan una serie de consejos que si se aplican se evitará cometer los errores más comunes a la hora de crear un sitio web. Además, también se incluyen una serie de consejos sobre accesibilidad, tanto de cara al usuario como de cara al navegador.

Índice General

7.1. Guía de estilo

A continuación hemos incluido una serie de indicaciones que pueden ayudar a la hora de crear páginas web. Muchas de estas indicaciones se basan en la experiencia acumulada y en casos reales de gente que cometió los errores que se intentan evitar con estos consejos.

7.1.1. Organizar el código HTML

Si se crea una página web directamente con el código **HTML**, es recomendable hacerlo como si se estuviese programando. Es decir, organizar el código para que sea más fácil su lectura (sangría, separación de fragmentos, etc.), poner comentarios, etc. Todo ello supone un trabajo extra, pero facilita el mantenimiento futuro de las páginas.

7.1.2. Cuidado con los colores

Hay que llevar mucho cuidado con las combinaciones de colores que se emplean, ya que algunas cuestan mucho de leer.

Si tenemos páginas con mucho texto, es conveniente usar una combinación de colores de alto contraste, que facilite la lectura: un color oscuro sobre un fondo claro (negro o azul sobre blanco o un color crema) o al revés (un color claro para el texto sobre un fondo oscuro).

7.1.3. Cuidado con los colores por defecto

Cada navegador emplea unos colores por defecto para el color de fondo de una página, para el color del texto, para los enlaces, etc. Así, lo normal es que el color de fondo de una página sea el blanco, el color del texto el negro y el color de los enlaces el azul. Sin embargo, hay navegadores que no emplean estos colores. Por tanto, para evitar problemas, lo mejor es configurar los colores de una página siempre, aunque se empleen los colores que por defecto emplean la mayoría de los navegadores.

Muy importante: cuando se modifique uno de los colores por defecto (por ejemplo, el color de fondo de una página) es muy recomendable modificar el resto de colores. Imaginemos que puede ocurrir si sólo se cambia el color de fondo de una página a amarillo en un navegador en el que los colores por defecto del fondo de la página y del texto son azul oscuro y amarillo respectivamente: no se podrá leer ningún texto.

7.1.4. Cuidado con los tipos de letra

El uso de distintos tipos de letra puede originar bastantes problemas. Se suele recomendar un tipo de letra `sans-serif`[1] (como `Arial`, `Helvetica`, `Geneva`, `Tahoma` o `Verdana`) para la lectura de texto en pantalla, ya que los "serifs" no se muestran correctamente debido a que la resolución de la pantalla es limitada.

Por otro lado, normalmente se elige un tipo de letra concreto para un sitio web, pero por descuido u olvido, en algunas páginas aparece texto escrito en otro tipo de letra (por ejemplo, el tipo de letra por defecto del navegador). El visitante suele apreciar estos errores inmediatamente y suele causar una sensación de "dejadez" en la creación de las páginas web.

Por último, hay que intentar elegir tipos de letra "estándar". Si empleamos un tipo poco usual, seguramente la mayoría de la gente no lo tendrá instalado en sus ordenadores y las páginas no se visualizarán correctamente.

7.1.5. Cuidado con los valores absolutos

En muchas etiquetas (por ejemplo, `<HR>`, `<TABLE> ... </TABLE>`, y `<FRAMESET> ... </FRAMESET>`) de pueden especificar tamaños (anchura o altura) mediante valores absolutos (número de pixels) o valores relativos (porcentaje respecto al tamaño de la página o de la ventana). Puesto que distintos usuarios pueden tener distintos tamaños de ventana, es aconsejable usar siempre valores relativos. Si se emplean valores absolutos, cuando el usuario emplea una resolución de pantalla distinta a la que hemos empleado en la creación de la página, puede ser que para ver el contenido de la página haya que emplear la barra de desplazamiento (cuando la resolución del usuario sea inferior a la nuestra), o que aparezcan grandes zonas en blanco o vacías (cuando la resolución del usuario sea superior a la nuestra).

7.1.6. Cuidado con las barras de desplazamiento

Hay que intentar crear páginas web que quepan en una ventana, es decir, que no sea necesario emplear las barras de desplazamiento para ver todo el contenido de una página. Si el usuario tiene que emplear las barras de desplazamiento para leer una página acabará cansándose. Sobretodo, hay que evitar la aparición de la barra de desplazamiento horizontal, ya que es la que más dificulta la lectura de una página web.

[1] En español este tipo de letra se conoce como "sin gracias". Los "serifs" son pequeñas líneas decorativas en los extremos de los caracteres. Para textos impresos en papel los tipos de letra `serif` facilitan la lectura, ya que varios experimentos han demostrado que los "serifs" de los caracteres ayudan a fijar la vista en la línea que se está leyendo. Los tipos de letra `serif` más comunes son `Courier`, `Georgia`,`New Century Schoolbook`, `Palatino` y `Times Roman`.

7.1.7. Cuidado con las imágenes de fondo

Las imágenes de fondo mal utilizadas pueden originar varios problemas. Por un lado, si tienen muchos colores y detalles pueden ocultar otros detalles de la página e impedir la lectura correcta del texto.

Por otro lado, cuando la imagen tiene un tamaño menor que la ventana del navegador, la imagen se muestra en forma de "mosaico" hasta cubrir toda la ventana del navegador. Si usamos una imagen con un tamaño que cubre toda la ventana para una resolución de pantalla dada, cuando se emplee una resolución mayor la imagen se mostrará varias veces hasta cubrir toda la ventana y puede ser que se produzcan efectos indeseados si no hemos preparado la imagen para que se muestre en forma de mosaico.

7.1.8. Sacar partido al hipertexto

Las referencias cruzadas permiten abordar un mismo tema en distintos niveles de detalle. Se puede permitir al usuario pasar de largo información técnica o conceptos avanzados de un tema, estableciendo enlaces a textos más extensos por si desea ampliar información. En general, podemos disponer de una página corta, con una presentación de la información escueta, que lleve a páginas que contengan detalles acerca de los temas tratados.

7.1.9. Usar las capacidades multimedia

Como se suele decir, "una imagen vale más que mil palabras". Pero hay que tener cuidado: el uso de muchas imágenes puede confundir al usuario. Además, hay que tener en cuenta la velocidad de transferencia: las imágenes emplean muchos kilobytes, lo que aumenta el tiempo que tarda en cargarse una página. Por ello, hay que calcular el tiempo que tarda en cargarse una página según diferentes velocidades de conexión y ponerse un límite de unos 10 segundos como máximo. Por ejemplo, con un módem de 56 Kbps (bits por segundo) en 10 segundos se pueden recibir unos 70 KB (bytes) y con una línea *Asymmetric Digital Subscriber Line* (**ADSL**) de 128 Kbps unos 160 KB[2].

Un truco muy importante: una vez cargada una imagen, el navegador la almacena en la cache; cuando volvamos a emplear esa misma imagen en otra página, no necesitará descargarla otra vez. Para que nuestras páginas se carguen más rápidamente, es aconsejable combinar los mismos elementos gráficos (iconos, líneas, imágenes de fondo, etc.) en todas nuestras páginas.

También hay que tener en cuenta que si una página es muy compleja, con muchas tablas e imágenes, según la velocidad del ordenador el navegador puede tardar varios segundos en mostrar la página (incluso aunque haya recibido todos los datos que definen la página).

[2] Todo ello sin contar que se pueden emplear algoritmos de compresión en la transmisión.

7.1.10. Identidad corporativa

Hay que intentar conseguir una "identidad corporativa" en todas las páginas: emplear los mismos tipos de letra, colores, imágenes de fondo, iconos, etc., para dotar a las páginas de un estilo homogéneo. De esta forma, el usuario se sentirá "inmerso" en las páginas.

7.1.11. Permitir que los usuarios se comuniquen

En todas las páginas (o por lo menos en la inicial y en la principal) hay que incluir una dirección de contacto para que los usuarios se puedan comunicar. De este modo, se podrán recibir sugerencias, indicaciones de cómo mejorar las páginas o errores localizados.

7.1.12. Facilitar las búsquedas

Una adecuada clasificación y exposición de la información que contienen nuestras páginas facilitará la navegación de los usuarios. En general, la página principal debería presentar un índice de toda la información disponible en el sitio web. Otros mecanismos que facilitan la búsqueda de información y la navegación son el buscador y el mapa del sitio web.

7.1.13. Revisar las páginas periódicamente

No hay nada peor que encontrar enlaces rotos: enlaces que apuntan a páginas o recursos que no existen, porque se hayan borrado, cambiado de nombre, movido de sitio o porque falle la ruta de acceso. Una revisión periódica del sitio web puede ahorrar al usuario muchos problemas. Además, es importante indicar la fecha de la última modificación y las novedades añadidas. Esto permite, entre otras cosas, que el usuario se asegure de la seriedad del autor de las páginas y facilita la navegación a los usuarios asiduos.

Por otro lado, si una página aparece con un signo "en construcción" y su última revisión es del año anterior, el usuario pensará que es mejor buscar lo que quiere en otra parte.

7.1.14. Los enlaces

La elección del lugar apropiado para poner los enlaces es crucial para una correcta presentación del hipertexto. Por ejemplo, compárense los dos siguientes trozos de código **HTML** que contienen un enlace. Es evidente que la primera opción es mucho mejor que la segunda.

Ejemplo 7.1

```
1 La <A HREF="/concejalias/turismo">Concejalía de Turismo</A> se
2 encarga de gestionar el turismo rural y de playa ...
```

Ejemplo 7.2

```
1 La Concejalía de Turismo se encarga de gestionar el turismo rural
2 y de playa ... (<A HREF="/concejalias/turismo">haga click aquí
3 para ver más información acerca de la Concejalía de Turismo</A>).
```

En los primeros años de la web había que poner `haga click` porque la gente no estaba acostumbrada a los enlaces, pero eso ya es historia.

7.2. Accesibilidad

Aunque **W3C** se encarga de estandarizar la tecnología empleada en la Web, las compañías que crean los principales navegadores web emplean su propia versión del estándar. Esta se suele diferenciar en varios aspectos:

- Incorpora etiquetas o atributos que no están definidos en el estándar de **W3C**.
- No incorpora etiquetas o atributos que están definidos en el estándar de **W3C**.
- Modifica el funcionamiento establecido de etiquetas o atributos que están definidos en el estándar de **W3C**.

Si se crea un sitio web para publicar en Internet, se tienen que evitar los diseños que limitan la consulta de las páginas web a determinados navegadores. Sobre este tema, Tim Berners-Lee se pronunció a mediados de 1996:

> Anyone who slaps a 'this page is best viewed with Browser X' label on a Web page appears to be yearning for the bad old days, before the Web, when you had very little chance of reading a document written on another computer, another word processor, or another network.
>
> *Los que estampan 'Esta página se ve mejor (optimizada) para el navegador X' en una página web parecen añorar los viejos tiempos, antes de la Web, cuando se tenían muy pocas posibilidades de leer un documento escrito en otro ordenador, con otro procesador de textos, o en otra red.*
>
> Tim Berners-Lee en *Technology Review*, julio 1996

En el contexto de la creación de páginas web, la *accesibilidad* mide la facilidad con la que se puede acceder, leer y entender el contenido de un sitio web. La accesibilidad tiene dos dimensiones: por un lado, cómo de accesible es un sitio web de cara al usuario que accede a él, y por otro lado, cómo de accesible es de cara al navegador que interpreta las páginas.

7.2.1. Accesibilidad de cara al usuario

El siguiente comentario de Tim Berners-Lee, inventor de **WWW** y director de **W3C** es muy revelador:

> The power of the Web is in its universality. Access by everyone regardless of disability is an essential aspect.
>
> *El poder de la Web reside en su universalidad. Que cualquiera pueda acceder independientemente de minusvalías es un hecho esencial.*
>
> Tim Berners-Lee, `http://www.w3.org/WAI/`

Un diseño web inadecuado puede presentar barreras a la gente con minusvalías, en especial a la que posea discapacidades sensoriales o neurológicas. Por ello, el **W3C** ha creado el *Web Accessibility Initiative* (**WAI**), cuyo objetivo es aumentar la usabilidad de la Web de cara a la gente con minusvalías. El **WAI** se estructura en tres grupos que desarrollan propuestas en tres ámbitos:

- *Web Content Accessibility Guidelines Working Group*: guías para la creación de sitios web.
- *Authoring Tool Accessibility Guidelines Working Group*: guías para las herramientas de diseño web.
- *User Agent Accessibility Guidelines Working Group*: guías para los navegadores.

Por ejemplo, *Web Content Accessibility Guidelines* 2.0 del 22 de agosto de 2002 contiene los siguientes consejos principales (cada consejo se divide en subconsejos y dentro de cada uno existen tres niveles de cumplimiento según los consejos que se cumplan):

1. Perceptible. Asegura que todo el contenido se puede presentar de una forma (o formas) que pueda ser percibido por cualquier usuario, excepto aquellos aspectos del contenido que no se puedan expresar con palabras.
2. Operable. Garantiza que los elementos del interfaz pueden ser manejados por cualquier usuario.
3. Navegable. Facilita la navegación.
4. Entendible. Cuanto más fácil sea de entender el contenido y los controles, mejor.
5. Robusto. Emplea tecnologías web que sean lo más compatible posible.

Existen diversas herramientas que ayudan a mejorar la accesibilidad de una página. Una de las más famosas es Bobby[3], que verifica tanto las guías establecidas por **W3C** como una normativa referente a la accesibilidad creada por el gobierno de los Estados Unidos.

7.2.2. Accesibilidad de cara al navegador

Como el lenguaje **HTML** se encuentra en continua evolución y los navegadores aceptan diferentes etiquetas y atributos, la "degradación elegante" (*graceful degradation*) es la clave para lograr que una página sea accesible por la mayor parte de los navegadores.

Cuando un navegador encuentra una etiqueta que no entiende[4], tiene lugar la degradación. En este proceso, se puede perder parte del contenido de la página o se puede seguir mostrando todo el contenido pero con la pérdida de algunas de las características establecidas por el autor de la página. Por ejemplo, un navegador que no acepte el atributo `BGCOLOR` en una tabla, debe de seguir mostrando la tabla, aunque sin un color de fondo. En este último caso es cuando hablamos de degradación elegante.

[3] Existe una versión de demostración en `http://bobby.watchfire.com/bobby/html/en/index.jsp`.

[4] También puede ser que el usuario haya desactivado alguna opción, como puede ser la carga de imágenes o la ejecución de código de *script*.

Capítulo 8

Lenguajes de script

Los lenguajes de script permiten incluir "programación" en las páginas web. En este capítulo se explican las principales características y las tres formas que existen de incluir y ejecutar código en una página web.

Índice General

8.1. Introducción

Un lenguaje de *script* (o lenguaje de guiones) es similar a un lenguaje de macros o a un fichero por lotes (*batch*): una lista de comandos que se pueden ejecutar sin o con la participación del usuario. Se trata en definitiva de un lenguaje de programación, que suele emplearse dentro de un contexto (dentro de una aplicación) y que no permite programar aplicaciones independientes (no permite crear ficheros ejecutables independientes). Los lenguajes de *script* suelen ser interpretados, aunque algunos pueden poseer una fase de pseudocompilación con el fin de optimizar su ejecución.

Existen multitud de lenguajes de *script* que se pueden emplear en páginas web: *JavaScript*, *VBScript*, *Perl*, *Rexx*, etc. Sin embargo, algunos sólo se pueden emplear en navegadores muy concretos y poco extendidos. Los lenguajes de *script* más empleados en Internet son *JavaScript* y en menor medida *VBScript*.

8.2. Diferencias entre VBScript y JavaScript

La diferencia más obvia entre *VBScript* y *JavaScript* (MICROSOFT lo denomina *JScript*) se encuentra en su sintaxis. *JavaScript* y *VBScript* nacieron con un mismo fin: dotar de un lenguaje de *script* rápido y sencillo a las páginas web. Cada uno de estos lenguajes posee una serie de características que no posee el otro, pero ninguna de las diferencias existentes entre ambos permite descartar un lenguaje por el otro. Ambos poseen la misma potencia.

VBScript es un subconjunto de *Visual Basic* el lenguaje de programación "estrella" de MICROSOFT. Como *VBScript* se diseñó específicamente para trabajar con navegadores, no incluye características que se escapan del ámbito de los lenguajes de *script*, como el acceso a ficheros y la impresión. Se puede emplear en la mayoría de los productos de MICROSOFT: Microsoft Internet Explorer, Microsoft Office, **ASP**, etc.

JavaScript, por otro lado, deriva de la familia de lenguajes formada por *C*, *C++* y *Java*. Es conveniente aclarar desde un principio que *JavaScript* y *Java* son lenguajes totalmente distintos, desarrollados por distintas compañías (NETSCAPE y SUN MICROSYSTEMS respectivamente) y que, en principio, sólo comparten parte de la sintaxis y del nombre y poco más. Se puede emplear en la mayoría de los navegadores (Microsoft Internet Explorer, Netscape Communicator y Opera), en el servidor web Netscape Enterprise Server y para programar con la tecnología **ASP**.

Aunque tanto *VBScript* como *JavaScript* nacieron con el objetivo de proporcionar capacidades de programación en los clientes web, en la actualidad ambos también se pueden emplear en el servidor web.

Entonces, ¿qué lenguaje elegir? La principal razón para elegir uno u otro reside en la siguiente sencilla pregunta: **¿la plataforma que yo uso soporta ese lenguaje?** Mientras que los dos navegadores más extendidos, Microsoft Internet Explorer y Netscape Communicator, admiten *JavaScript*, sólo el primero admite *VBScript*. Por tanto, si elegimos este lenguaje de *script*, estaremos limitando el acceso a las páginas web que desarrollemos.

8.3. Para qué sirven

Las aplicaciones más habituales de los lenguajes de *script* en las páginas **HTML** son:

- Validar datos en el cliente y comprobar la consistencia de los valores antes de mandar un formulario al servidor web (como, por ejemplo, comprobar que una fecha tiene un valor adecuado y un formato correcto).

- Actualizar campos relacionados en formularios (por ejemplo, establecer las opciones de una lista desplegable en función del valor seleccionado en unos botones de radio).

- Realizar procesamientos que no requieran la utilización de información centralizada (por ejemplo, convertir pesetas en euros, visualizar el calendario del mes actual, etc.).
- Servir de base para la utilización de otras tecnologías (**DHTML**, *Java*, *ActiveX*, etc.).

Los lenguajes de *script* también pueden actuar sobre el navegador a través de objetos integrados que representan al documento, la ventana activa, cada uno de los controles de un formulario, etc. Para ello se emplea un modelo de objetos denominado *Document Object Model* (**DOM**), que veremos en el Capítulo 10.

Los lenguajes de *script* presentan fuertes restricciones de acceso a los recursos de la máquina en la que se ejecutan. Estas restricciones no se deben a limitaciones tecnológicas, sino a normas impuestas por los diseñadores de los lenguajes de *script* para evitar que la ejecución del código de una página web pueda dañar la integridad del sistema del usuario.

8.4. Como se usa un lenguaje de script en un navegador

Existen tres formas de incluir e invocar *scripts* dentro de una página **HTML**:

1. Usando la etiqueta `<SCRIPT> ... </SCRIPT>`. Esta etiqueta permite incluir código de *script* directamente en la página o que apunte a un fichero externo usando el atributo `SRC`. Esta etiqueta se puede incluir tanto en la sección `HEAD` como en `BODY`. Esta etiqueta posee el atributo `LANGUAGE` que permite indicar en que lenguaje (y en que versión, si el lenguaje posee varias[1]) se ha escrito el *script*. Si se omite este atributo, cada navegador tiene definido por defecto un lenguaje[2]. El siguiente ejemplo muestra como combinar múltiples lenguajes de *script* en la misma página[3].

Ejemplo 8.1

```
1 <HTML>
2 <BODY>
3 <SCRIPT LANGUAGE="VBScript">
4   document.write "Esto lo ha escrito VBScript<BR>"
```

[1] Por ejemplo, para el lenguaje *JavaScript* podemos usar los identificadores *JavaScript*, *JavaScript1.1*, *JavaScript1.2* y *JavaScript1.3*.

[2] Tanto Netscape Communicator como Microsoft Internet Explorer, si no se indica un lenguaje con la etiqueta `LANGUAGE`, suponen que se trata de *JavaScript*.

[3] No se recomienda mezclar en una misma página distintos lenguajes de *script*: su mantenimiento es más difícil y su ejecución es más lenta, ya que se tiene que activar un intérprete distinto para cada lenguaje.

```
</SCRIPT>
<SCRIPT LANGUAGE="JavaScript">
  document.write("Esto lo ha escrito JavaScript<BR>");
</SCRIPT>
</BODY>
</HTML>
```

Mediante el atributo `SRC` se puede ocultar y proteger el código *script* de una página, aunque no es un sistema seguro, ya que cualquier usuario con unos conocimientos mínimos puede acceder al fichero que contiene el código. La ventaja principal de este sistema es que permite compartir el mismo código entre distintas páginas web. De este modo, se pueden crear librerías de código.

2. Usando los atributos de etiquetas **HTML** que soportan *scripts*. Existen una serie de elementos (`<FORM>`, `<INPUT>`, `<SELECT>`, `<TEXTAREA>`, `<BODY>` y `<A>`) pertenecientes al lenguaje **HTML** que permiten incluir código *script* en una serie de atributos que representan eventos. Cuando estos eventos se producen (por ejemplo, al pulsar sobre un botón o al cargar una página), el código correspondiente se ejecuta. Estas etiquetas poseen el atributo `LANGUAGE`, que al igual que en la etiqueta `SCRIPT`, permite indicar en que lenguaje se ha escrito el código. El siguiente ejemplo muestra como gestionar los eventos de pulsación sobre botones (`<INPUT TYPE="BUTTON">`) mediante el atributo `ONCLICK`.

Ejemplo 8.2

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="VBScript">
  Sub PulsadoVBS
    alert "Pulsado el botón VBScript"
  End Sub
</SCRIPT>
<SCRIPT LANGUAGE="JavaScript">
  function PulsadoJS()
  {
    alert("Pulsado el botón JavaScript");
  }
</SCRIPT>
</HEAD>
<BODY>
<FORM NAME="Formulario">
<INPUT TYPE="BUTTON" VALUE="VBScript" ONCLICK="PulsadoVBS"
 LANGUAGE="VBScript">
<BR>
<INPUT TYPE="BUTTON" VALUE="JavaScript" ONCLICK="PulsadoJS()"
```

```
 LANGUAGE="JavaScript">
</FORM>
</BODY>
</HTML>
```

3. Incluyendo código de *script* en una **URL**. Los *scripts* también se pueden invocar desde el elemento `<A> ... </A>` (*anchor*, ancla) mediante una **URL**. Esto permite que el *script* se ejecute cuando el usuario pulsa sobre un enlace. El siguiente ejemplo muestra una página con dos enlaces. Al pulsar sobre uno de ellos se abre una ventana nueva que muestra un mensaje. En la Figura 8.1 se muestra la ventana que se abre al ejecutar código de *JavaScript* y en la Figura 8.2 se muestra el resultado de pulsar el enlace que ejecuta código de *VBScript*.

Ejemplo 8.3

```
<HTML>
<HEAD>
<TITLE>Código de JavaScript y VBScript en una misma página</TITLE>
<SCRIPT LANGUAGE="VBScript">
  Sub PulsadoVBS
    alert "Pulsado el enlace VBScript"
  End Sub
</SCRIPT>
<SCRIPT LANGUAGE="JavaScript">
  function PulsadoJS()
  {
    alert("Pulsado el enlace JavaScript");
  }
</SCRIPT>
</HEAD>
<BODY>
<A HREF="javascript:PulsadoJS()">JavaScript</A>
<BR>
<A HREF="javascript:PulsadoVBS()">VBScript</A>
</BODY>
</HTML>
```

Como podemos observar en la **URL** del enlace, con la palabra `javascript:` indicamos el lenguaje que estamos empleando, y a continuación figura el código a ejecutar. Como vemos en el ejemplo, desde *JavaScript* se puede llamar a funciones escritas en *VBScript* y viceversa, pero como se ha comentado previamente, no se recomienda mezclar distintos lenguajes de *script* en una misma página.

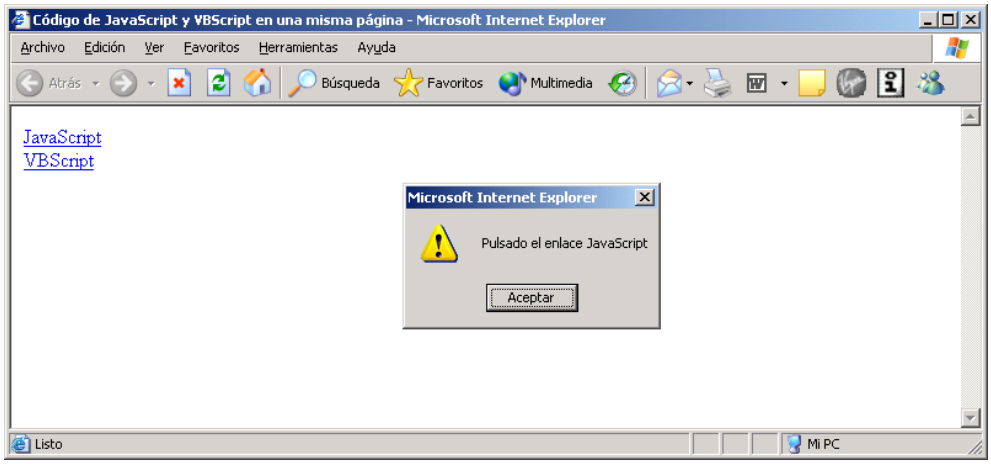


Figura 8.1: Código JavaScript en un enlace

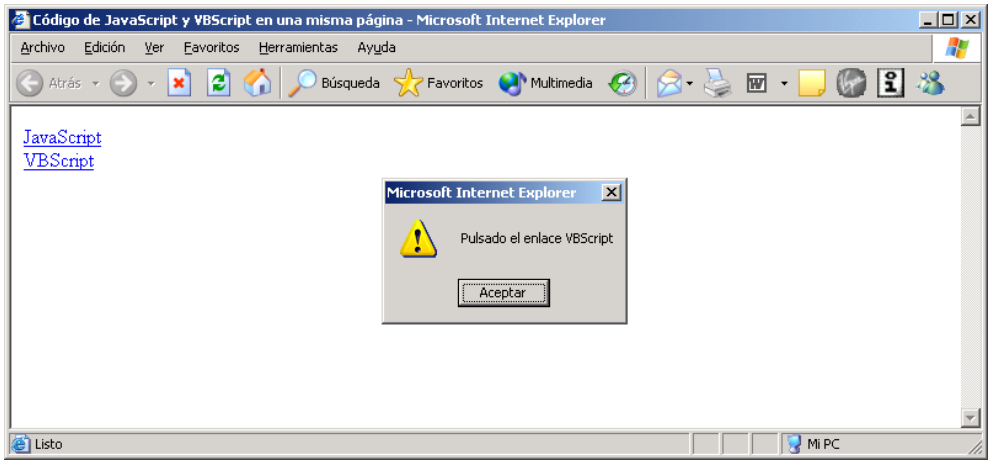


Figura 8.2: Código VBScript en un enlace

Capítulo 9

JavaScript

En el capítulo anterior hemos visto los lenguajes de script en general. En este capítulo vamos a estudiar un lenguaje de script concreto: JavaScript. Este lenguaje es el más empleado en Internet y se puede considerar el lenguaje estándar de programación de clientes web. Veremos sus características básicas, sus diferentes sentencias, las funciones que incorpora y, por último, como validar formularios.

Índice General

9.1. Introducción

JavaScript es un lenguaje interpretado, basado en objetos (no es un lenguaje orientado a objetos "puro") y multiplataforma, inventado por NETSCAPE COMMUNICATIONS CORPORATION. Los navegadores de NETSCAPE fueron los primeros que usaron *JavaScript*. El primer nombre oficial de este lenguaje fue *LiveScript* y apareció por primera vez en la versión beta de Netscape Navigator 2.0 en septiembre de 1995, pero poco después fue rebautizado *JavaScript* en un comunicado conjunto con SUN MICROSYSTEMS el 4 de diciembre de 1995[1].

El núcleo de *JavaScript* (*Core JavaScript*) contiene una serie de objetos, como `Array`, `Date`, `Math`, `Number` y `String`, y un conjunto de elementos del lenguaje como operadores, estructuras de control y sentencias (`%`, `++`, `if`, `for`, `break`, etc.). El núcleo de *JavaScript* se puede ampliar añadiendo objetos adicionales, como por ejemplo:

- *JavaScript* en el cliente (*Client-side JavaScript*) amplía el lenguaje añadiendo objetos que permiten controlar un navegador y su **DOM**.
- *JavaScript* en el servidor (*Server-side JavaScript*) amplía el lenguaje añadiendo objetos que son útiles cuando *JavaScript* se ejecuta en un servidor (lectura de ficheros, acceso a bases de datos, etc.).

[1] *"Netscape and Sun announce Javascript, the open, cross-platform object scripting language for enterprise networks and the Internet"*, nota de prensa disponible en `http://home.netscape.com/newsref/pr/newsrelease67.html`.

JavaScript permite crear aplicaciones que se ejecuten a través de Internet, basadas en el paradigma cliente/servidor. La parte del cliente se ejecuta en un navegador, como Netscape Communicator o Microsoft Internet Explorer, mientras que la parte del servidor se ejecuta en un servidor, como Netscape Enterprise Server.

Mediante la característica *JavaScript's LiveConnect*, se pueden intercomunicar el código *JavaScript* con *Java*. Desde *JavaScript*, se pueden instanciar objetos *Java* y acceder a sus propiedades y métodos públicos. A su vez, desde *Java* se puede acceder a los objetos de *JavaScript*, a sus propiedades y a sus métodos.

9.1.1. Aplicaciones

Una de las aplicaciones principales de *JavaScript* consiste en validar la entrada introducida por el usuario a través de un formulario, que luego recibirán aplicaciones que se ejecutan en el servidor (hechas en **ASP**, **CGI**, **JSP** o cualquier otra tecnología). La utilidad de esto reside en:

- Reduce la carga en el servidor. Los datos incorrectos se filtran en el cliente y no se envían al servidor.
- Reduce los retrasos producidos por errores cometidos por el usuario. De otro modo la validación se tendría que realizar en el servidor, y los datos deberían viajar del cliente al servidor, ser procesados y entonces devueltos al cliente para que los corrigiese.
- Simplifica los programas que se ejecutan en el servidor al dividir el trabajo entre el cliente y el servidor.

Normalmente, la validación de los datos se puede realizar como mínimo en tres ocasiones:

- Cuando el usuario introduce los datos, con un manejador del evento `onChange` en cada control que se quiere validar del formulario.
- Cuando el usuario envía (*submit*) el formulario, con un manejador del evento `onClick` en el botón que envía el formulario.
- Cuando el usuario envía (*submit*) el formulario, con un manejador del evento `onSubmit` en el formulario.

Otra de las aplicaciones de *JavaScript* consiste en proporcionar dinamismo a las páginas **HTML**. Si se emplea junto con **DHTML** se pueden conseguir efectos sorprendentes.

9.1.2. Qué necesito para programar en JavaScript

Para poder programar en *JavaScript* no hace falta ningún equipo especial ni comprar un entorno de desarrollo específico. Sólo es necesario un editor de textos como Bloc de notas de Microsoft Windows o joe de Linux y un navegador como Netscape Communicator o Microsoft Internet Explorer que soporte *JavaScript* para poder comprobar el código.

No es necesario disponer de una conexión a Internet, ya que se puede comprobar localmente el código creado.

Lo que sí que es recomendable es utilizar un buen editor de textos, que sea cómodo, configurable, soporte macros, etc. y que sea *syntax highlight*. Esta última característica significa que el editor es capaz de comprender el lenguaje en el que se programa, y colorea las palabras diferenciándolas según sean variables, palabras reservadas, comentarios, etc.

9.1.3. JavaScript y Java

JavaScript y *Java* se confunden a veces entre sí. Son dos lenguajes similares en algunos aspectos, pero diferentes entre sí. *JavaScript* no tiene la estricta comprobación de tipos que posee *Java*, pero ambos si que comparten la sintaxis de las expresiones, de los operadores y de las estructuras de control.

JavaScript es un lenguaje interpretado, mientras que *Java* posee una fase de compilación (aunque no se genera código máquina, sino los llamados *bytecodes* que luego son interpretados por la *máquina virtual Java*). *JavaScript* tiene un pequeño número de tipos de datos: números (enteros y en coma flotante), booleanos y cadenas. *JavaScript* posee un modelo de objetos basado en prototipos mientras que el de *Java* se basa en clases.

JavaScript es un lenguaje con mucha libertad si se compara con *Java*. No es necesario declarar las variables, clases y métodos. No hay que preocuparse con métodos públicos, privados o protegidos, y no hay que implementar interfaces. En el Cuadro 9.1 se resumen las principales diferencias existentes entre *JavaScript* y *Java*.

9.1.4. Versiones

Cada versión de los navegadores soporta una versión diferente de *JavaScript*. Cada nueva versión de *JavaScript* añade nuevas características al lenguaje (y corrige fallos de las anteriores). El Cuadro 9.2 muestra las distintas versiones de *JavaScript* soportadas por las diferentes versiones de Netscape Navigator. Las versiones anteriores a la 2.0 no soportaban *JavaScript*. Actualmente se encuentra en desarrollo *JavaScript 2.0*[2].

Si queremos detectar la versión de *JavaScript* que admite nuestro navegador podemos emplear el siguiente código:

[2]En `http://www.mozilla.org/js/language/js20/` se puede consultar la evolución de la nueva versión.

JavaScript	**Java**
Interpretado (no compilado) por el cliente.	Compilado a *bytecodes* que se descargan desde el servidor y se ejecutan en el cliente.
Basado en objetos. No distingue entre tipos de objetos. La herencia se realiza a través del mecanismo de prototipado, y las propiedades y los métodos se pueden añadir a cualquier objeto de forma dinámica.	Basado en clases. Distinción entre clases e instancias, la herencia se realiza a través de la jerarquía de clases. No se pueden añadir a las clases ni a las instancias propiedades o métodos de forma dinámica.
Código integrado y embebido con el código **HTML**.	Los *applets* se distinguen del código **HTML** (aunque se acceden a través de **HTML**), ya que se almacenan en ficheros independientes.
El tipo de dato de las variables no se declara (tipos dinámicos).	El tipo de dato de las variables se tiene que declarar (tipos estáticos).

Cuadro 9.1: JavaScript frente a Java

Versión de JavaScript	**Versión de Navigator**
JavaScript 1.0	Navigator 2.0
JavaScript 1.1	Navigator 3.0
JavaScript 1.2	Navigator 4.0 - 4.05
JavaScript 1.3	Navigator 4.06 - 4.78
JavaScript 1.4	Ningún navegador (sólo en los productos de servidor de Netscape)
JavaScript 1.5	Navigator 6.0 y Mozilla
JavaScript 2.0	En desarrollo

Cuadro 9.2: Relación entre las versiones de JavaScript y de Netscape Navigator

Ejemplo 9.1

```
<HTML>
<HEAD>
<TITLE>Ejemplo version JavaScript</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<SCRIPT LANGUAGE="JavaScript">
<!--
  var version = "JavaScript"
// -->
</SCRIPT>
<SCRIPT LANGUAGE="JavaScript1.1">
<!--
  version = "JavaScript 1.1"
// -->
</SCRIPT>
<SCRIPT LANGUAGE="JavaScript1.2">
<!--
  version = "JavaScript 1.2"
// -->
</SCRIPT>
<SCRIPT LANGUAGE="JavaScript1.3">
<!--
  version = "JavaScript 1.3"
// -->
</SCRIPT>
<SCRIPT LANGUAGE="JavaScript1.4">
<!--
  version = "JavaScript 1.4"
// -->
</SCRIPT>
<SCRIPT LANGUAGE="JavaScript1.5">
<!--
  version = "JavaScript 1.5"
// -->
</SCRIPT>
<SCRIPT LANGUAGE="JavaScript">
<!--
  document.write("Soporta " + version);
// -->
</SCRIPT>
</BODY>
</HTML>
```

Este código funciona de la siguiente forma: mediante el atributo `LANGUAGE` de la

etiqueta `<SCRIPT> ... </SCRIPT>` se indica la versión de *JavaScript* que se emplea. Como las versiones aparecen de menor a mayor, cuando se llegue a una versión que no se admita, la variable `version` almacenará la última versión admitida, que será la mayor de todas las que se admiten.

En el ejemplo anterior se puede ver que las instrucciones de *JavaScript* aparecen entre comentarios de **HTML** (`<!-- Comentario -->`). Este sistema impide que el código de *script* se visualice en aquellos navegadores que no entienden la etiqueta `<SCRIPT> ... </SCRIPT>`. En el resto de ejemplos de este libro no se incluye, pero es una "buena práctica" hacerlo siempre con el fin de lograr la máxima compatibilidad.

9.1.5. JavaScript y ECMA

Aunque NETSCAPE inventó *JavaScript* y sus navegadores fueron los primeros que lo soportaron, NETSCAPE trabaja estrechamente con *European Computer Manufacturers Association* (**ECMA**) para obtener un lenguaje de programación estándar basado en *JavaScript*.

La versión estandarizada de *JavaScript*, llamada *ECMAScript* se "debe de" comportar de forma idéntica en todas las aplicaciones que soportan el estándar. La primera versión de *ECMAScript* se detalla en la especificación *ECMA-262 Edition 1*. Este estándar también ha sido aprobado por *International Organization for Standards* (**ISO**) bajo la denominación *ISO-16262*.

JavaScript siempre incluirá características que no formen parte de la especificación de **ECMA**. Pero estas características no impiden que *JavaScript* sea compatible con **ECMA**.

9.1.6. JScript

JScript es otro lenguaje de *script* que puede emplearse tanto en el cliente (dentro de páginas **HTML**) como en el servidor (por ejemplo, en páginas **ASP**). *JScript* es la implementación realizada por MICROSOFT del lenguaje de script estándar *ECMA-262*.

En los navegadores de MICROSOFT, un aspecto importante a tener en cuenta es que la versión del lenguaje de *script* es independiente de la versión del navegador. Por ejemplo, se puede instalar el motor de *JScript* v5 en Microsoft Internet Explorer 3 y usar todas las características del lenguaje (por seguridad, no se puede hacer lo contrario, instalar *JScript* v1 en Microsoft Internet Explorer 5).

En el Cuadro 9.4 se muestran las distintas versiones de *JScript* existentes.

9.1.7. Diferencias entre JavaScript y JScript

En las primeras versiones, *JavaScript* y *JScript* poseían muchas diferencias entre ellos. Sin embargo, en las últimas versiones las diferencias prácticamente han desaparecido, ya que ambos aseguran que cumplen el estándar *ECMA-262*. Pero aún quedan pequeñas diferencias que pueden hacer "perder el tiempo" al programador.

Versión de JavaScript	**Versión de ECMA**
JavaScript 1.1	ECMA-262 Edition 1 se basa en JavaScript 1.1
JavaScript 1.2	ECMA-262 Edition 1 no se había completado cuando apareció JavaScript 1.2 (la compatibilidad entre ambos no es total).
JavaScript 1.3	JavaScript 1.3 es completamente compatible con ECMA-262 Edition 1, pero posee características adicionales que no forman parte de ECMA-262 Edition 1.
JavaScript 1.4	JavaScript 1.4 es completamente compatible con ECMA-262 Edition 1, pero con características adicionales. Cuando apareció, aún no se había completado ECMA-262 Edition 3.
JavaScript 1.5	JavaScript 1.5 es completamente compatible con ECMA-262 Edition 3.

Cuadro 9.3: Relación entre las versiones de JavaScript y de ECMA

Versión de JScript	**Productos de Microsoft**
JScript v1	Internet Explorer 3
JScript v2	Internet Information Server 3.0
JScript v3	Internet Explorer 4
JScript v4	Visual Interdev 6.0
JScript v5	Windows 2000, Office 2000, Internet Explorer 5 y 6

Cuadro 9.4: Relación entre las versiones de JScript y los productos de Microsoft

Por ejemplo, el siguiente código produce distintos resultados al ejecutarse en Netscape Communicator o Microsoft Internet Explorer. La diferencia se encuentra en que el primero convierte automáticamente a entero el valor de la variable a, que almacena una cadena, mientras que el segundo navegador no lo hace.

Ejemplo 9.2

```
<HTML>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
a = "1";

switch(a)
{
  case 0:
    document.write("0");
    break;

  case 1:
    document.write("1");
    break;

  case 2:
    document.write("2");
    break;

  default:
    document.write("Ninguno");
    break;
}
</SCRIPT>
</BODY>
</HTML>
```

La salida que produce Netscape Communicator 4.7 es: 1.
La salida que produce Microsoft Internet Explorer 5.0 es: Ninguno.

9.2. El lenguaje

9.2.1. Características básicas

La sintaxis de *JavaScript* es prácticamente idéntica a la de *C*, *C++* y *Java*[3]. Si se conoce alguno de estos lenguajes, aprender *JavaScript* resulta muy sencillo y requiere

[3] A partir de ahora, cuando hagamos referencia a *C*, se entiende que también se incluyen *C++* y *Java*.

poco esfuerzo. Los comentarios, las funciones, las sentencias de control, etc., poseen la misma sintaxis que en *C*. *JavaScript* es un lenguaje basado en objetos, pero no orientado a objetos como *Java* y *C++*. El objetivo de *JavaScript* es conseguir código sencillo y reducido, no crear programas grandes y complejos.

JavaScript es un lenguaje *case sensitive* (sensible a minúsculas/mayúsculas). Esto quiere decir que todas las palabras que usemos para referirnos a los distintos identificadores del lenguaje tendrán que ser escritas correctamente para que sean comprendidas por el intérprete de *JavaScript*. Por ejemplo, en el siguiente código se declaran tres variables que son distintas:

Ejemplo 9.3

```
1 var nombre, Nombre, NOMBRE;
```

En *JavaScript* no es necesario acabar las sentencias con un punto y coma (sí que es obligatorio en *C*), excepto para separar código que se encuentra en la misma línea. Pero es una buena recomendación acabar siempre las sentencias con punto y coma. Por ejemplo, las tres líneas siguientes de código son correctas:

Ejemplo 9.4

```
1 var nombre; var apellido1
2 var apellido2
3 var ciudad;
```

Al igual que en *C*, el código se estructura por bloques (*block-structured computer language*). Un bloque de código se indica delimitando las sentencias que lo componen mediante las llaves ({ y }). Por ejemplo, las siguientes sentencias forman un bloque de código:

Ejemplo 9.5

```
1 {
2   document.write("Una sentencia ");
3   document.write("en un ");
4   document.write("bloque de código");
5 }
```

Los espacios en blanco, tabuladores y saltos de línea no poseen valor en *JavaScript*, por lo que el código anterior se podría escribir de otra forma, tal como se ve en el siguiente ejemplo, y sería totalmente equivalente. El "aspecto visual" del código depende del estilo que cada uno quiera emplear.

Ejemplo 9.6

```
{
  document.write("Una sentencia "); document.write("en un ");
                document.write("bloque de código");}
```

9.2.2. Comentarios

Los comentarios son totalmente transparentes al código, y no afectan al mismo ni en el modo de ejecutarse ni en la velocidad de ejecución. Los comentarios se implementan de dos maneras diferentes.

La primera es utilizando dos barras inclinadas (//). Desde que el intérprete de *JavaScript* encuentra las dos barras hasta que llega al final de la línea, todo el texto que haya entremedias será ignorado.

La segunda manera de comentar fragmentos de código es usando los símbolos barra inclinada y asterisco (/*) y viceversa (*/). Estos símbolos definen zonas de comentario que pueden extenderse a través de varias líneas.

Ejemplo 9.7

```
// Esta linea esta comentada
var a; // El comentario no afecta al codigo

/* Todo este codigo esta comentado
var b = 0;
alert("Hola a todos");
*/
```

9.2.3. Declaración de variables

Para definir variables en *JavaScript* usaremos la palabra reservada `var`. Al contrario que en otros lenguajes, no es necesario declarar las variables que se emplean, pero es una buena costumbre.

No todas las palabras sirven para definir variables. Existen tres normas que hay que cumplir:

- Un nombre de variable válido se compone únicamente de una palabra (sin espacios), pudiendo estar formada por letras, números o el guión de subrayado (_).
- El primer carácter del nombre de la variable deberá ser siempre una letra o el carácter de guión de subrayado. No podrá ser un número.
- El nombre de la variable no debe coincidir con ninguna de las palabras reservadas.

Al declarar una variable es posible asignarle un valor inicial. Si no se le asigna un valor inicial, posee un valor nulo representado por la palabra clave `null`. El empleo de variables sin valor puede producir errores.

Ejemplo 9.8

```
1 // Declaraciones correctas
2 var _nombre, _apellido1, _apellido2;
3 var Provincia = "Alicante";
4 var ciudad1, ciudad2;
5 var codigo_postal = "03001";
6
7 // Declaraciones incorrectas
8 var 12edad;
9 var var;
```

JavaScript es un lenguaje sin tipos, es decir, no necesita una declaración del tipo de las variables para trabajar con ellas, porque él mismo se encarga de reconocer que es lo que se quiere almacenar. Los distintos tipos de datos que puede almacenar una variable en *JavaScript* son:

- **Cadenas**. Las cadenas de texto se delimitan por comillas dobles (") o comillas simples (') y pueden contener cualquier tipo de combinación de números, letras, espacios y símbolos. El uso de los dos tipos de comillas tiene su sentido, ya que debido a que *JavaScript* se usa conjuntamente con el lenguaje **HTML**, y como éste usa las comillas dobles en su sintaxis, tiene que haber algún método de distinguir las cadenas de *JavaScript* de las cadenas de **HTML**. Por ejemplo: `var ciudad = "Elche", provincia = "Alicante";`.
- **Números enteros**. Un número entero se representa como cualquier sucesión de dígitos (0 a 9), precedidos por el signo menos (-) si es un número negativo. Por ejemplo: `var a = 1, b = 100000, c = -123456789;`.
- **Números en coma flotante**. Se representan de igual forma que los números enteros, pero añaden una parte decimal que se representa con un punto (.) seguido de tantas cifras como compongan la parte decimal del número. Por ejemplo: `var a = 9.12, b = -3.141592, c = 10000000000000.1;`.
- **Booleanos**. Representa únicamente dos valores, verdadero (`true`) o falso (`false`) y se suele utilizar en sentencias condicionales. Por ejemplo: `var rojo = true, verde = false;`. Además, el 0 se considera como falso y cualquier valor numérico distinto de 0 se considera verdadero.
- **Nulo**. Se utiliza únicamente para comprobar si una variable que se ha definido tiene un valor asignado o no. Se representa con la palabra clave `null`. Por ejemplo:

Ejemplo 9.9

```
var nombre;
if(nombre == null)
{
  alert("Introduzca su nombre, por favor");
}
```

9.2.4. Ámbito de las variables

Ya se ha comentado antes que no es necesario declarar las variables antes de usarlas. Sin embargo, el hecho de declarar una variable sí que influye en el ámbito de existencia (dónde existe y cuánto tiempo existe una variable). Siempre que se declare una variable local a un ámbito (a una función, por ejemplo), dominará sobre cualquier otra variable que se hubiese declarado fuera de ese ámbito. Además, una variable declarada en una función no puede ser accedida desde otra función.

Por ejemplo, en el siguiente código se declaran tres variables globales `a`, `b` y `c`. Aunque la variable `b` no haya sido declarada con `var`, el solo hecho de asignarle un valor ya supone declararla. En la función `fun()`, se declara una variable `c` local (`"Maria"`), por lo que no se puede acceder a la variable `c` global (`"Ana"`). Desde esta función se llama a la función `fun2()`; desde esta función sí que se puede acceder a la variable global `c`.

Ejemplo 9.10

```
<HTML>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
var a = "Juan";
b = "Jose";
var c = "Ana";

function fun2()
{
  document.write("fun2-c- " + c + "<BR>");
}

function fun()
{
  var c = "Maria";

  document.write("fun1-a- " + a + "<BR>");
  document.write("fun1-b- " + b + "<BR>");
  fun2();
  document.write("fun1-c- " + c + "<BR>");
}
```

```
22
23 fun();
24
25 document.write("-c-" + c + "<BR>");
26 </SCRIPT>
27 </BODY>
28 </HTML>
```

El código anterior produce la siguiente salida en un navegador:

Ejemplo 9.11

```
1 fun1-a- Juan
2 fun1-b- Jose
3 fun2-c- Ana
4 fun1-c- Maria
5 -c- Ana
```

9.2.5. Caracteres especiales

Además de los caracteres normales, en una cadena también se pueden incluir caracteres especiales que permiten generar saltos de líneas o caracteres a partir de su código **ASCII**. El Cuadro 9.5 muestra los caracteres especiales que se pueden emplear dentro de una cadena.

Por ejemplo, el siguiente código produce el resultado que se muestra en la Figura 9.1.

Ejemplo 9.12

```
1 <HTML>
2 <HEAD>
3 <TITLE>Ejemplo de caracteres especiales</TITLE>
4 </HEAD>
5 <BODY>
6 <SCRIPT LANGUAGE="JavaScript">
7 document.write("Distintas formas de representar el carácter 'A':");
8 document.write("<BR>");
9 document.write("\\101 produce: \101<BR>");
10 document.write("\\x41 produce: \x41<BR>");
11 document.write("\\u0041 produce: \u0041<BR>");
12 document.write("<BR>");
13 document.write("Algunos caracteres interesantes:");
14 document.write("<BR>");
15 document.write("\\251 produce: \251<BR>");
16 document.write("\\252 produce: \252<BR>");
17 document.write("\\256 produce: \256<BR>");
```

Carácter	Significado
\b	Retroceso (*backspace*)
\f	Salto de página (*form feed*)
\n	Salto de línea (*new line*)
\r	Retorno de carro (*carriage return*)
\t	Tabulador
\'	Apóstrofe o comilla simple
\"	Comilla doble
\\	Barra invertida (*backslash*)
\XXX	El carácter de la codificación Latin-1 especificado por los tres dígitos octales entre 0 y 377.
\xXX	El carácter de la codificación Latin-1 especificado por los dos dígitos hexadecimales entre 00 y FF.
\uXXXX	El carácter Unicode especificado por los cuatro dígitos hexadecimales entre 0000 y FFFF.

Cuadro 9.5: Caracteres especiales

```
18 document.write("\\274 produce: \274<BR>");
19 document.write("\\275 produce: \275<BR>");
20 document.write("\\276 produce: \276<BR>");
21 </SCRIPT>
22 </BODY>
23 </HTML>
```

9.2.6. Operadores

Los operadores son los signos que se usan para referirse a distintas operaciones que se pueden realizar con las variables y con las constantes, tales como suma, resta, incremento, etc.

La precedencia de los operadores determina el orden en que se aplican cuando se evalúa una expresión. Se puede anular la precedencia usando paréntesis. El Cuadro 9.6 muestra la precedencia de los operadores, de menos a más. Los operadores que se encuentran en un mismo nivel poseen la misma precedencia.

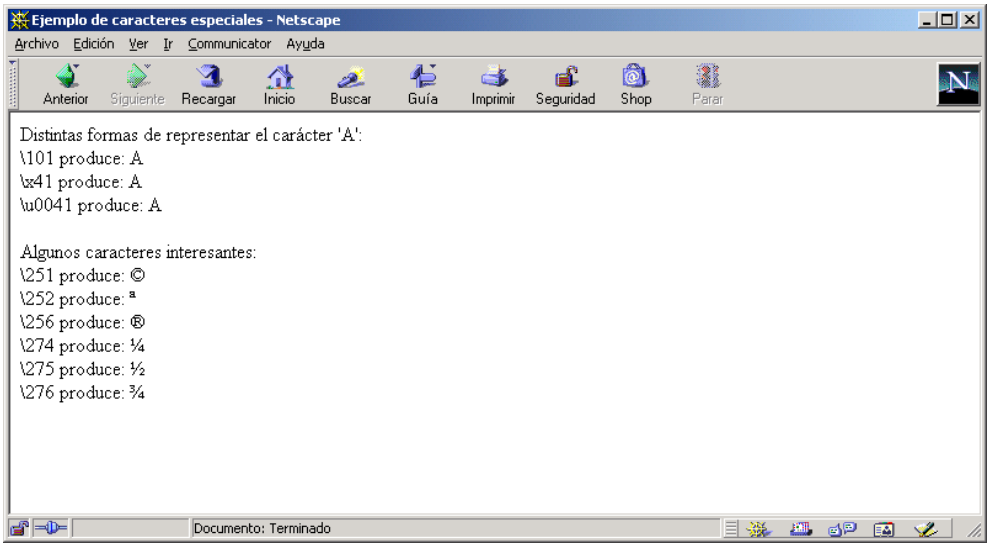


Figura 9.1: Ejemplo de caracteres especiales

Tipo de operador	**Operadores**
Coma	`,`
Asignación	`= += -= *= /= %= <<= >>= >>>= &= ^= |=`
Condicional	`?:`
O lógico (OR)	`||`
Y lógico (AND)	`&&`
O bit a bit (OR)	`|`
O exclusiva bit a bit (XOR)	`^`
Y bit a bit (AND)	`&`
Igualdad	`== != === !==`
Relacional	`< <= > >=`
Desplazamiento bit a bit	`<< >> >>>`
Suma y resta	`+ -`
Multiplicación y división	`* / %`
Negación e incremento	`! ~ - + ++ -- typeof void delete`
Llamada a función	`()`
Creación de instancias	`new`
Miembro	`. []`

Cuadro 9.6: Precedencia de los operadores de JavaScript

9.2.7. Palabras reservadas

Las palabras reservadas (*reserved words*) no se pueden usar como nombres de variables, funciones, métodos u objetos. Algunas de las palabras reservadas son palabras clave (*keywords*) de *JavaScript*, mientras que otras se han reservado para futuros usos.

En el Cuadro 9.7 se muestran las palabras reservadas de *JavaScript* 1.3.

abstract	boolean	break	byte
case	catch	char	class
const	continue	debugger	default
delete	do	double	else
enum	export	extends	false
final	finally	float	for
function	goto	if	implements
import	in	instanceof	int
interface	long	native	new
null	package	private	protected
public	return	short	static
super	switch	synchronized	this
throw	throws	transient	true
try	typeof	var	void
volatile	while	with	

Cuadro 9.7: Palabras reservadas de JavaScript

9.3. Sentencias

Vamos a detallar las sentencias que incluye el lenguaje *JavaScript*. Todas estas sentencias son anidables: se pueden incluir unas dentro de otras tantas veces como se quiera.

9.3.1. Condicionales

Una sentencia condicional es un conjunto de comandos que se ejecutan si una condición es cierta (`true`). *JavaScript* proporciona dos sentencias condicionales: `if ... else` y `switch`.

if ... else

La sentencia de selección simple posee la siguiente sintaxis (la parte entre corchetes es opcional):

Ejemplo 9.13

```
if(condicion)
{
  sen1
}
[else
{
  sen2
}]
```

El conjunto de sentencias `sen1` se ejecuta si `condicion` es cierta; en caso contrario se ejecuta `sen2`.

Como ejemplo tenemos el siguiente código:

Ejemplo 9.14

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
var a;

a = prompt("Escriba un número, por favor", "");
if(a <= 1)
{
  alert("Es menor o igual que 1");
}
else if(a <= 5)
{
  alert("Es mayor que 1, pero menor o igual que 5");
}
else
{
  alert("Es mayor que 1");
}
</SCRIPT>
<TITLE>Ejemplo if</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
</BODY>
</HTML>
```

switch

La sentencia de selección múltiple posee la siguiente sintaxis:

Ejemplo 9.15

```
switch(expresion)
{
  case et1 :
    sen1;
    [break;]
  case et2 :
    sen2;
    [break;]
  ...
  [default :
    sen;]
}
```

La sentencia `switch` evalúa una expresión e intenta encontrar una etiqueta (valor) que case con el resultado de la evaluación. Si se logra casar, se ejecutan las sentencias asociadas con esa etiqueta. Si no se logra ningún emparejamiento, se busca la etiqueta opcional `default` y se ejecutan sus sentencias asociadas.

La sentencia `break` opcional en cada etiqueta asegura que el programa salga del `switch` una vez que se han ejecutado las sentencias correspondientes a una opción `case` y continúa la ejecución por la línea siguiente a la sentencia `switch`. Si no se incluye un `break`, el flujo de ejecución pasará de una opción `case` a otra hasta que encuentre un `break` o finalice la sentencia `switch`.

Como ejemplo tenemos el siguiente código:

Ejemplo 9.16

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
var a;

a = prompt("¿Cuántos hijos tiene?", "");
switch(a)
{
  case 0: alert("Anímese un día de estos");
    break;
  case 1: alert("Puede tener más");
    break;
  case 2: alert("Ya tiene la parejita, ahora a por el trío");
    break;
  default: alert("Una familia como las de antes: muchos hijos");
    break;
}
</SCRIPT>
```

```
<TITLE>Ejemplo switch</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
</BODY>
</HTML>
```

9.3.2. De repetición

Un bucle es un conjunto de comandos que se ejecutan repetidamente hasta que una condición especificada deja de cumplirse. *JavaScript* ofrece tres tipos de sentencias de repetición: `for`, `do ... while` y `while`.

for

La sintaxis de la repetición con contador es la siguiente:

Ejemplo 9.17

```
for ([expInicial]; [condicion]; [expIncremento])
{
  sentencias
}
```

Cuando un bucle `for` se ejecuta, ocurre lo siguiente:

1. Si existe la expresión de inicialización `expInicial`, se ejecuta. La expresión inicial suele inicializar uno o más contadores, pero se pueden emplear expresiones de cualquier grado de complejidad.
2. Se evalúa la expresión `condicion`.
3. Si el valor de `condicion` es cierto (`true`), las `sentencias` del bucle se ejecutan. Si el valor de `condicion` es falso (`false`), el bucle termina.
4. La expresión de actualización `expIncremento` se ejecuta y el flujo de ejecución vuelve al paso 2.

Como ejemplo tenemos el siguiente código con un bucle que se repite 10 veces. En la Figura 9.2 se muestra esta página en un navegador.

Ejemplo 9.18

```
<HTML>
<HEAD>
<TITLE>Ejemplo for</TITLE>
</HEAD>
```

```
5  <BODY BGCOLOR="#FFFFFF">
6  <SCRIPT LANGUAGE="JavaScript">
7  var a;
8
9  for(a = 1; a <= 10; a++)
10 {
11   document.write("Vamos por el número " + a + "<BR>\n");
12 }
13 </SCRIPT>
14 </BODY>
15 </HTML>
```

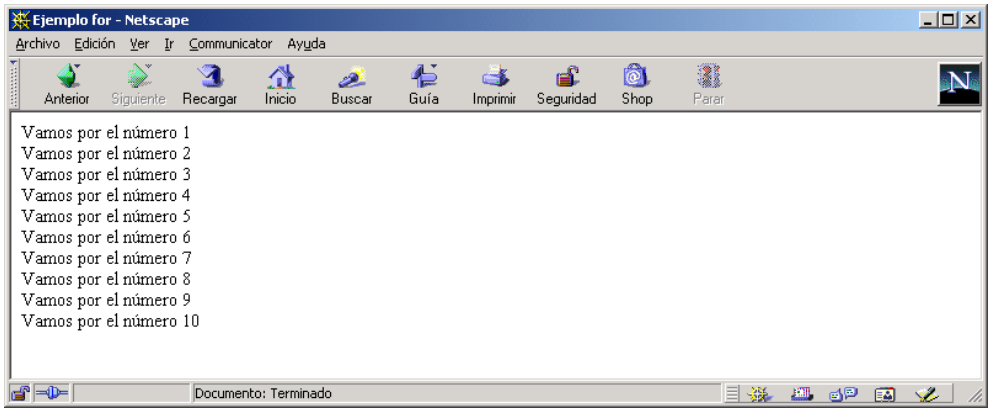


Figura 9.2: Ejemplo de uso de la instrucción for

do ... while

La sentencia de repetición con condición final posee la siguiente sintaxis:

Ejemplo 9.19

```
1 do
2 {
3   sentencias
4 } while(condicion)
```

Las `sentencias` se ejecutan siempre al menos una vez antes de que `condicion` se evalúe. Si se evalúa a cierto, se repite el proceso: las `sentencias` se ejecutan una vez más y `condicion` se vuelve a evaluar.

Como ejemplo tenemos el siguiente código:

Ejemplo 9.20

```
<HTML>
<HEAD>
<TITLE>Ejemplo do while</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<PRE>
<SCRIPT LANGUAGE="JavaScript">
var a, b;

a = 1;
do
{
  b = a;
  do
  {
    document.write(" ");
    b--;
  }
  while(b > 0);
  document.write(a + "<BR>\n");
  a++;
}
while(a < 10);
</SCRIPT>
</PRE>
</BODY>
</HTML>
```

while

La sintaxis de la sentencia de repetición con condición inicial es la siguiente:

Ejemplo 9.21

```
while(condicion)
{
  sentencias
}
```

Si `condicion` es falsa, las `sentencias` del bucle dejan de ejecutarse y el control pasa a la siguiente sentencia después del bucle. La evaluación de `condicion` ocurre

antes de que se ejecuten `sentencias`. Por tanto, puede ocurrir que las `sentencias` no se ejecuten nunca.

Como ejemplo tenemos el siguiente código, que crea las tablas de multiplicar del 1 al 10. En la Figura 9.3 se muestra esta página en un navegador.

Ejemplo 9.22

```
<HTML>
<HEAD>
<TITLE>Ejemplo while</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<PRE>
<SCRIPT LANGUAGE="JavaScript">
var a, b;

a = 1;
while(a <= 10)
{
  document.write("Tabla del " + a + "\n");
  document.write("------------\n");
  b = 1;
  while(b <= 10)
  {
    document.write(a + " x " + b + " = " + (a * b) + "\n");
    b++;
  }
  document.write("\n");
  a++;
}
</SCRIPT>
</PRE>
</BODY>
</HTML>
```

break

La sintaxis de esta sentencia es:

Ejemplo 9.23

```
break
o
break etiqueta
```

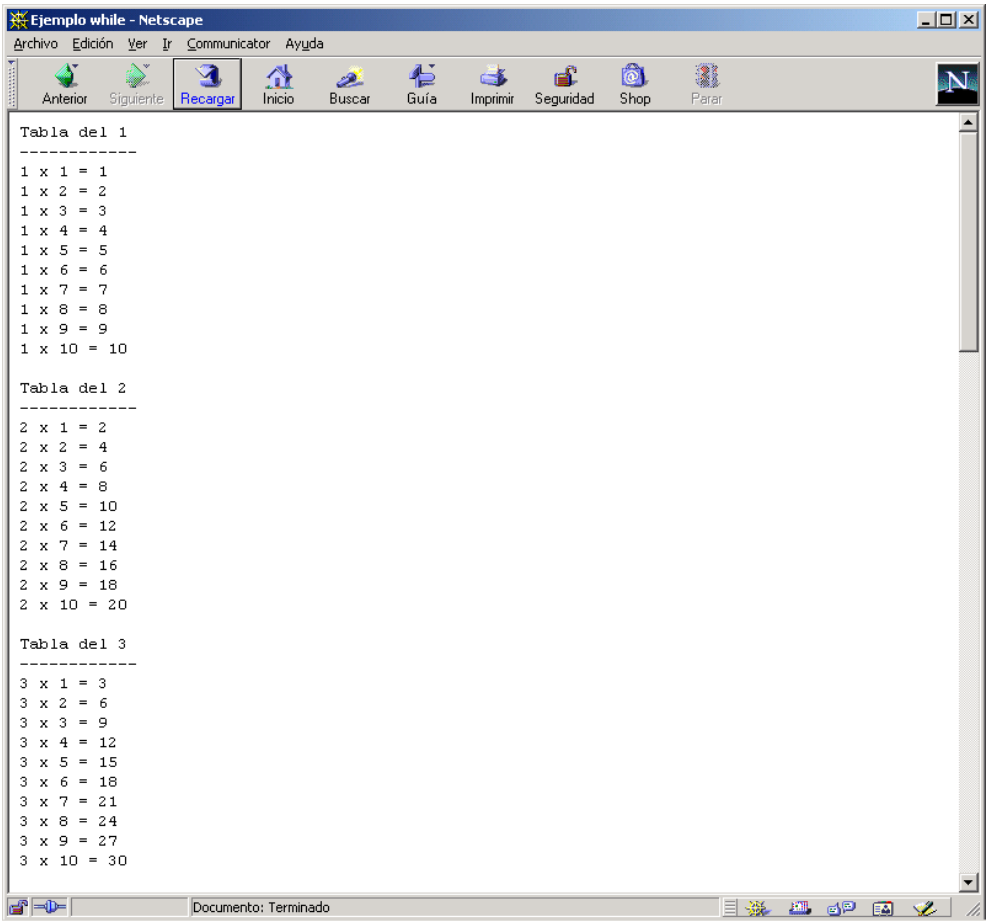


Figura 9.3: Ejemplo de uso de la instrucción while

La sentencia `break` se usa para finalizar un bucle (`for`, `do..while`, `while`) o una sentencia de selección múltiple (`switch`).

continue

La sintaxis de esta sentencia es:

Ejemplo 9.24

```
1 continue
2 o
3 continue etiqueta
```

La sentencia `continue` se usa para pasar a la siguiente iteración en un bucle (`for`, `do ... while`, `while`).

9.3.3. De manipulación de objetos

Existen dos sentencias de manipulación de objetos: `for(... in ...)` y `with`.

for(... in ...)

La sintaxis de esta sentencia es:

Ejemplo 9.25

```
1 for(variable in objeto)
2 {
3   sentencias
4 }
```

Esta sentencia permite que una `variable` recorra todas las propiedades de un `objeto`. Para cada propiedad del objeto, se ejecutan las `sentencias`. Para acceder a las propiedades de un objeto se emplea el operador punto (`.`).

Como ejemplo tenemos el siguiente código que muestra las propiedades del objeto `window` de **DOM**[4]. En la Figura 9.4 se muestra esta página en un navegador.

Ejemplo 9.26

```
1 <HTML>
2 <HEAD>
3 <SCRIPT LANGUAGE="JavaScript">
4 function Propiedades(obj, obj_nombre)
5 {
```

[4] **DOM** se explica en el Capítulo 10.

```
6   var resultado = "";
7
8   for (var i in obj)
9   {
10    resultado += obj_nombre + "." + i + " = " + obj[i] + "<BR>"
11  }
12
13  return resultado;
14 }
15 </SCRIPT>
16 <TITLE>Ejemplo for_in</TITLE>
17 </HEAD>
18 <BODY BGCOLOR="#FFFFFF">
19 <PRE>
20 <SCRIPT LANGUAGE="JavaScript">
21 document.write(Propiedades(window, "window"));
22 </SCRIPT>
23 </PRE>
24 </BODY>
25 </HTML>
```

with

La sintaxis de esta sentencia es:

Ejemplo 9.27

```
1 with(objeto)
2 {
3   sentencias
4 }
```

La sentencia `with` establece el `objeto` por defecto sobre el que se ejecutan un conjunto de `sentencias` que pueden acceder a las propiedades o métodos del objeto. De este modo, se evita tener que escribir el nombre del objeto cada vez.

Como ejemplo tenemos el siguiente código:

Ejemplo 9.28

```
1 <HTML>
2 <HEAD>
3 <TITLE>Ejemplo with</TITLE>
4 </HEAD>
5 <BODY BGCOLOR="#FFFFFF">
6 <PRE>
```

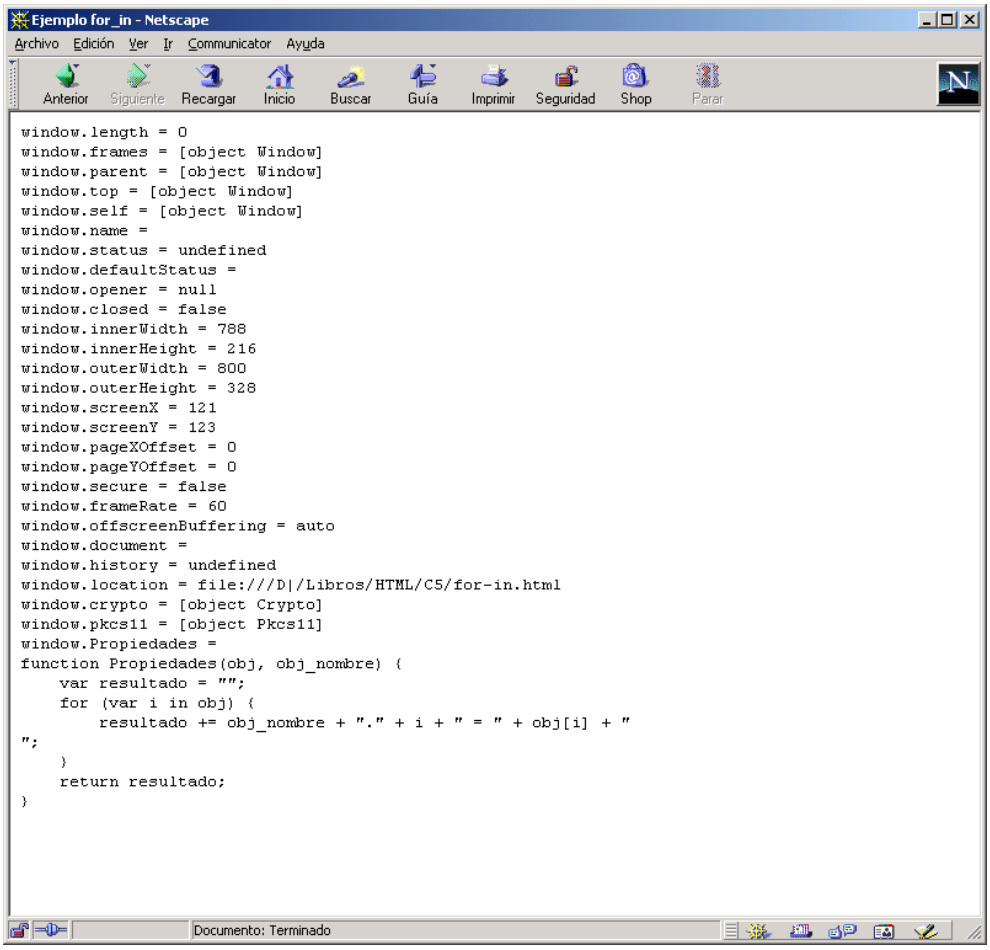


Figura 9.4: Ejemplo de uso de la instrucción for(... in ...)

```
<SCRIPT LANGUAGE="JavaScript">
with(Math)
{
  document.write("E: " + E + "\n");
  document.write("LN10: " + LN10 + "\n");
  document.write("LN2: " + LN2 + "\n");
  document.write("LOG10E: " + LOG10E + "\n");
  document.write("LOG2E: " + LOG2E + "\n");
  document.write("Pi: " + PI + "\n");
  document.write("SQRT1_2: " + SQRT1_2 + "\n");
  document.write("SQRT2: " + SQRT2 + "\n");
}
</SCRIPT>
</PRE>
</BODY>
</HTML>
```

9.4. Funciones

Una función es un fragmento de código al que se suministran unos datos determinados (parámetros o argumentos), ejecuta un conjunto de sentencias y puede devolver un resultado.

9.4.1. Definición de una función

La definición de una función se compone de las siguientes partes:

- La palabra clave `function`.
- El nombre de la función.
- Una lista de argumentos, encerrados entre paréntesis y separados por comas. Una función puede tener cero o más argumentos.
- Las sentencias que definen la función, encerradas entre llaves. Las sentencias dentro de una función pueden llamar a su vez a otras funciones o a la misma función (recursividad).

Por tanto, la sintaxis de una función es:

Ejemplo 9.29

```
function nombre([arg1 [, arg2 [, ...]]])
{
  sentencias
}
```

En general, todas las funciones se deberían de definir en la sección `<HEAD> ... </HEAD>` de la página. Así, cuando el usuario carga la página, las funciones es lo primero que se carga. De otro modo, el usuario podría realizar alguna acción (por ejemplo, lanzar un evento) que llamase a una función que no ha sido definida aún, lo que produciría un error.

Por ejemplo, el siguiente código define dos funciones llamadas `cuadrado` y `cubo` que calculan el cuadrado y el cubo del número que reciben como argumento respectivamente:

Ejemplo 9.30

```
<HTML>
<HEAD>
<TITLE>Ejemplo function</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<PRE>
<SCRIPT LANGUAGE="JavaScript">
var a;

function cuadrado(numero)
{
  return numero * numero;
}

function cubo(numero)
{
  return numero * numero * numero;
}

a = prompt("Escribe un número entero:", "");
alert("El cuadrado de " + a + " es " + cuadrado(a));
a = prompt("Escribe un número entero:", "");
alert("El cubo de " + a + " es " + cubo(a));
</SCRIPT>
</PRE>
</BODY>
</HTML>
```

La instrucción `return` se emplea para finalizar la ejecución de una función y también para devolver un valor.

9.4.2. Funciones predefinidas

JavaScript posee un conjunto de funciones predefinidas: `eval`, `isFinite`, `isNaN`, `parseInt`, `parseFloat`, `Number`, `String`, `escape` y `unescape`.

eval

Evalúa una expresión que contiene código de *JavaScript*. La sintaxis es `eval(expr)`, donde `expr` es la expresión que va a ser evaluada.

Si la cadena representa una expresión, se evalúa; si representa una o más sentencias, se ejecutan. No hace falta llamar a `eval` para evaluar expresiones aritméticas, ya que *JavaScript* evalúa las expresiones aritméticas automáticamente. El siguiente ejemplo muestra el funcionamiento de esta función:

Ejemplo 9.31

```
<HTML>
<HEAD>
<TITLE>Ejemplo eval</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<SCRIPT LANGUAGE="JavaScript">
var a = 1, b = 2, c = 3, d = 4, e = 5, v;

v = prompt("Escriba una letra de la 'a' a la 'e'", "");
alert("El valor de '" + v + "' es " + eval(v));
</SCRIPT>
</BODY>
</HTML>
```

isFinite

Determina si el argumento representa un número finito. La sintaxis es `isFinite(num)` donde `num` es el numero que se quiere evaluar.

Si el argumento es `"NaN"` (*not a number*, no un número), devuelve `false`, en caso contrario devuelve `true`. Por ejemplo, el siguiente código permite filtrar la entrada del usuario y averiguar si ha introducido un número correcto o cualquier otra cosa:

Ejemplo 9.32

```
<HTML>
<HEAD>
<TITLE>Ejemplo isFinite</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<SCRIPT LANGUAGE="JavaScript">
v = prompt("Escriba un número", "");
if(isFinite(v))
  alert("Correcto, es un número");
else
  alert("Te has equivocado");
</SCRIPT>
```

```
13 </BODY>
14 </HTML>
```

isNaN

Determina si el argumento es `"NaN"` (*not a number*, no un número). La sintaxis es `isNaN(num)` donde `num` es el numero que se quiere evaluar.

Las funciones `parseFloat` y `parseInt` devuelven `"NaN"` cuando evaluan un valor que no es un número. La función `isNaN` devuelve true si el argumento es `"NaN"` y false en caso contrario.

parseInt y parseFloat

Las dos funciones de conversión `parseInt` y `parseFloat` devuelven un valor numérico a partir de una cadena.

La sintaxis de `parseFloat` es `parseFloat(cad)`, donde `cad` es la cadena que se quiere convertir a un número en coma flotante. Si encuentra un carácter distinto de un signo (+ o -), un dígito (0-9), un punto decimal o un exponente, devuelve el valor hallado hasta ese punto e ignora ese carácter y el resto. Si el primer carácter no puede convertirse a un número, devuelve `"NaN"` (*not a number*, no un número).

La sintaxis de `parseInt` es `parseInt(cad [, base])`, donde `cad` es la cadena que se quiere convertir a la base indicada en el segundo argumento opcional. Por ejemplo, 10 indica que se convierta a decimal, 8 a octal, 16 a hexadecimal. Para bases mayores que diez, las letras del alfabeto indican numerales mayores que nueve. Por ejemplo, para los números hexadecimales (base 16), las letras de la A a la F se usan. Si encuentra un carácter que no es un numeral válido en la base especificada, ese carácter y todos los caracteres que le siguen se ignoran y devuelve el valor convertido hasta ese punto. Si el primer carácter no puede convertirse a un número en la base especificada, devuelve `"NaN"` (*not a number*, no un número).

Number y String

Las funciones `Number` y `String` convierten un objeto a un número o a una cadena, respectivamente. La sintaxis de estas funciones es `Number(objRef)` y `String(objRef)`, donde `objRef` es una referencia a un objeto. Por ejemplo, el siguiente código convierte un objeto `Date` a un número y a una cadena:

Ejemplo 9.33

```
1 <HTML>
2 <HEAD>
3 <TITLE>Ejemplo Number-String</TITLE>
4 </HEAD>
5 <BODY BGCOLOR="#FFFFFF">
```

```
<SCRIPT LANGUAGE="JavaScript">
// Obtiene la fecha actual
var d = new Date();

// Milisegundos transcurridos desde el 1 de enero de 1970
alert("Milisegundos transcurridos: " + Number(d));

// Formato mas adecuado
alert("Fecha: " + String(d));
</SCRIPT>
</BODY>
</HTML>
```

escape y unescape

Las funciones `escape` y `unescape` permiten codificar y decodificar cadenas. La función `escape` devuelve la codificación hexadecimal de su argumento codificado en **ISO** Latin-1. La función `unescape` devuelve la cadena correspondiente a la codificación hexadecimal pasada como argumento. La sintaxis de estas funciones es `escape(string)` y `unescape(string)`. Por ejemplo, el siguiente código genera una página con un formulario con dos botones. Un botón aplica la función `escape` a la cadena que introduce el usuario y el otro aplica la función `unescape`.

Ejemplo 9.34

```
<HTML>
<HEAD>
<TITLE>Función predefinida escape()</TITLE>
<SCRIPT LANGUAGE="JavaScript"> function hazEscape()
{
  var cadena;

  cadena = prompt("Escriba una cadena", "");
  alert(escape(cadena));
}

function hazUnescape() {
  var cadena;

  cadena = prompt("Escriba una cadena", "");
  alert(unescape(cadena));
}
</SCRIPT>
</HEAD>
<BODY>
<CENTER>
```

```
22 <FORM NAME="formulario">
23 <INPUT TYPE="BUTTON" VALUE="escape" ONCLICK="hazEscape()">
24 <INPUT TYPE="BUTTON" VALUE="unescape" ONCLICK="hazUnescape()">
25 </FORM>
26 </CENTER>
27 </BODY>
28 </HTML>
```

9.5. Objetos

JavaScript es un lenguaje basado en objetos. Pero no se puede considerar que sea "orientado a objetos", ya que no posee características básicas como la herencia simple o múltiple, la jerarquía de clases, el polimorfismo, la ocultación de datos (visibilidad), etc.

En *JavaScript* un objeto es un constructor con *propiedades* (que pueden ser variables u otros objetos) y *métodos* (funciones asociadas a objetos).

En la Sección 9.3.3 se vieron las sentencias `for(... in ...)` y `with` que permiten manipular objetos. Para acceder a las propiedades o métodos de un objeto se emplea el operador punto (`.`). Por ejemplo, en el siguiente código se accede a la propiedad `status` y al método `alert` del objeto `window`:

Ejemplo 9.35

```
1 window.status = "Bienvenido a JavaScript";
2 window.alert("2 + 2 = " + (2 + 2));
```

También se puede acceder a las propiedades y métodos de un objeto mediante la notación de *arrays asociativos*. Se puede acceder al objeto como si fuera un array. Cada índice de acceso es una cadena que representa una propiedad o un método.

El código del ejemplo anterior se expresa mediante arrays asociativos de la siguiente forma:

Ejemplo 9.36

```
1 window["status"] = "Bienvenido a JavaScript";
2 window["alert"]("2 + 2 = " + (2 + 2));
```

9.5.1. Creación de objetos

Existen dos formas de crear un objeto nuevo: inicializadores de objetos y funciones constructoras.

Inicializadores de objetos

La sintaxis de este método es:

Ejemplo 9.37

```
nombreObjeto = {prop1:val1, prop2:val2, ..., propN:valN};
```

donde `nombreObjeto` es el nombre del nuevo objeto, `propI` es el nombre de una propiedad y `valI` el valor asociado a la propiedad. En *JavaScript*, un objeto puede tener como propiedad otro objeto.

Por ejemplo, el siguiente código define dos objetos llamados `profesor` y `asignatura`. El objeto `profesor` forma parte del objeto `asignatura`.

Ejemplo 9.38

```
profesor = {nombre:"Sergio", apellidos:"Luján Mora"};
asignatura = {cod:"PI", cred:6, prof:profesor};
document.write("Código: " + asignatura.cod);
document.write("<BR>");
document.write("Créditos: " + asignatura["cred"]);
document.write("<BR>");
document.write("Profesor: " + asignatura.prof.nombre
               + " " + asignatura.prof.apellidos);
document.write("<BR>");
```

Funciones constructoras

En este sistema se emplea una función que realiza el papel de "constructor". Cuando se crea el objeto, se emplea el operador `new` y el nombre de la función constructora. Este método es mejor que el anterior, ya que permite crear un "tipo" para distintos objetos. La sintaxis de este método es:

Ejemplo 9.39

```
function ObjConstructor(arg1, arg2, ..., argN)
{
  this.prop1 = arg1;
  this.prop2 = arg2;
  ...
  this.propN = argN;
}

objeto = new ObjConstructor(val1, val2, ..., valN);
```

En la función constructora se indican las propiedades y métodos del objeto. Una característica especial de *JavaScript* es que en cualquier momento se pueden añadir nuevas propiedades o métodos a un objeto concreto (no se añaden a todos los objetos del mismo tipo). Dentro de la función constructora se emplea la palabra reservada `this` para hacer referencia al objeto.

Por ejemplo, el siguiente código define un objeto de tipo `Persona` y dos objetos de tipo `Pagina`. Estos últimos poseen una propiedad que es un objeto de tipo `Persona`.

Ejemplo 9.40

```
1  function Persona(nombre, apellidos)
2  {
3    this.nombre = nombre;
4    this.apellidos = apellidos;
5  }
6
7  function Pagina(autor, url, fecha)
8  {
9    this.autor = autor;
10   this.url = url;
11   this.fecha = fecha;
12 }
13
14 slm = new Persona("Sergio", "Luján Mora");
15 p1 = new Pagina(slm, "index.html", "18/04/2001");
16 p2 = new Pagina(slm, "home.html", "23/03/2001");
17
18 document.write(p1.url + ": " + p1.fecha);
19 document.write("<BR>");
20 document.write(p2.url + ": " + p2.fecha);
```

9.5.2. Métodos de un objeto

Para definir un método de un objeto, simplemente hay que asignar a una propiedad del objeto el nombre de una función. En la función, si se quiere acceder a las propiedades del objeto, se tiene que emplear la palabra reservada `this`.

Por ejemplo, el siguiente código crea dos objetos de tipo `Ordenador`. Estos objetos poseen un método llamado `ver` que muestra las propiedades del objeto.

Ejemplo 9.41

```
1  function ver()
2  {
3    document.write("CPU: " + this.cpu + "<BR>");
4    document.write("Velocidad: " + this.vel + " MHz<BR>");
5    document.write("Memoria: " + this.mem + " Mb<BR>");
```

```
6   document.write("Disco duro: " + this.dd + " Gb<BR>");
7 }
8
9 function Ordenador(cpu, vel, mem, dd)
10 {
11   this.cpu = cpu;
12   this.vel = vel;
13   this.mem = mem;
14   this.dd = dd;
15   this.ver = ver;
16 }
17
18 o1 = new Ordenador("Pentium", 200, 32, 3);
19 o2 = new Ordenador("Pentium II", 500, 128, 15);
20 o1.ver();
21 o2.ver();
```

9.5.3. Eliminación de objetos

Para eliminar un objeto se emplea el operador `delete`. Este operador devuelve `true` si el borrado es correcto y `false` en caso contrario.

9.6. Tratamiento de cadenas

Las cadenas en *JavaScript* se pueden representar de dos formas:

- Directamente como literales: caracteres encerrados entre comillas simples o dobles.
- Como un objeto `String`: este objeto "envuelve" al tipo de dato cadena.

No se debe confundir una cadena literal con un objeto `String`. Por ejemplo, el siguiente código crea la cadena literal `s1` y el objeto de tipo `String` `s2`:

Ejemplo 9.42

```
1 // crea una cadena literal
2 s1 = "algo"
3
4 // crea un objeto String
5 s2 = new String("algo")
```

Todo los métodos del objeto `String` se pueden emplear directamente en una cadena literal: *JavaScript* convierte automáticamente la cadena literal a un objeto `String`

temporal, ejecuta el método sobre ese objeto y finaliza eliminando el objeto `String` temporal. También se puede emplear la propiedad `String.length` con una cadena literal: devuelve el número de caracteres (longitud) que posee la cadena.

A continuación mostramos los principales métodos que posee el objeto `String`: `charAt`, `concat`, `fromCharCode`, `indexOf`, `lastIndexOf`, `replace`, `slice` y `split`.

charAt(indice)

Devuelve el carácter especificado por `indice` en la cadena. El `indice` toma valores entre 0 y la longitud de la cadena menos 1. Ejemplo:

Ejemplo 9.43

```
var cadena = "Hola";

document.writeln("Carácter en 0 es " + cadena.charAt(0) + "<BR>");
document.writeln("Carácter en 1 es " + cadena.charAt(1) + "<BR>");
document.writeln("Carácter en 2 es " + cadena.charAt(2) + "<BR>");
```

El ejemplo anterior produce la siguiente salida:

Ejemplo 9.44

```
Carácter en 0 es H
Carácter en 1 es o
Carácter en 2 es l
```

concat(cadena1, cadena2, ..., cadenaN)

Concatena varias cadenas en una sola. Ejemplo:

Ejemplo 9.45

```
var cad1 = "Hola";
var cad2 = " a";
var cad3 = " todo";
var cad4 = " el mundo";
var cad5 = cad1.concat(cad2, cad3, cad4);

document.writeln(cad5);
```

El código anterior produce el siguiente resultado:

Ejemplo 9.46

```
Hola a todo el mundo
```

También se puede emplear el operador +, que además de realizar la suma también concatena cadenas. El ejemplo anterior se puede escribir como:

Ejemplo 9.47

```
var cad1 = "Hola";
var cad2 = " a";
var cad3 = " todo";
var cad4 = " el mundo";
var cad5 = cad1 + cad2 + cad3 + cad4;

document.writeln(cad5);
```

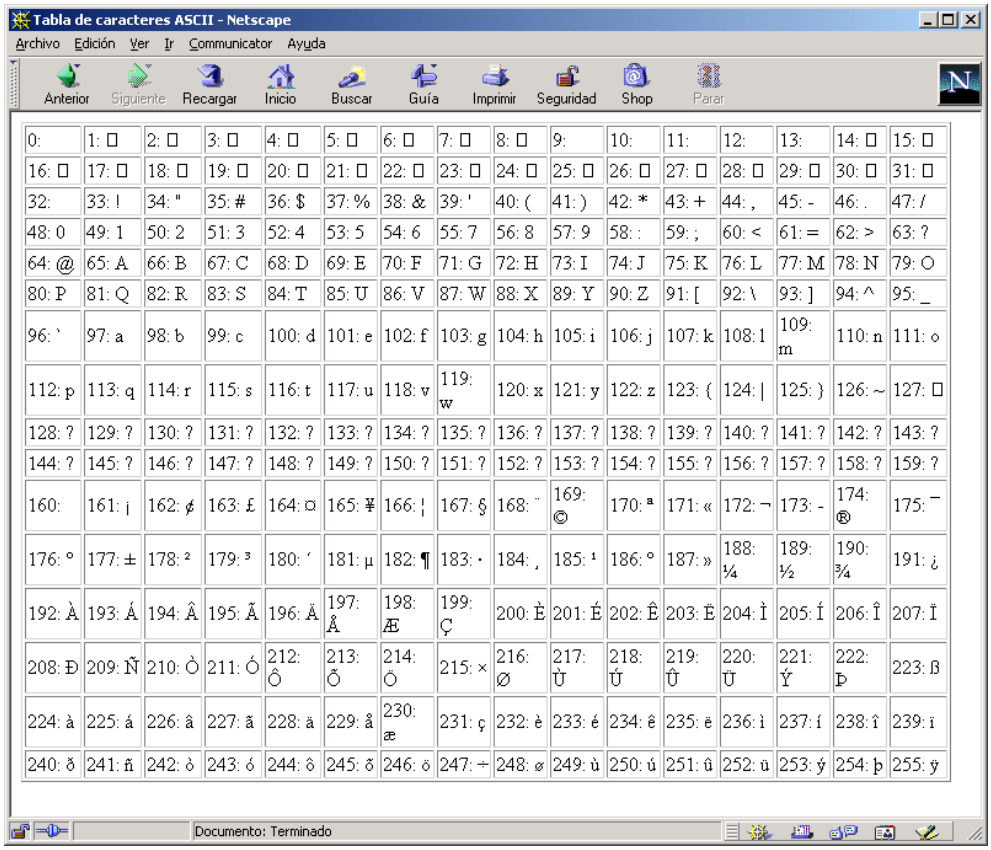
fromCharCode(num1 [, num2, ...])

Devuelve una cadena formada por los caracteres representados mediante valores Unicode que recibe la función.

Por ejemplo, el siguiente código crea una tabla con los 256 caracteres **ASCII** Figura 9.5.

Ejemplo 9.48

```
<HTML>
<HEAD>
<TITLE>Tabla de caracteres ASCII</TITLE>
</HEAD>
<BODY>
<TABLE BORDER="1">
<SCRIPT LANGUAGE="JavaScript">
for(i = 0; i < 16; i++)
{
  document.writeln("<TR>");
  for(j = 0; j < 16; j++)
  {
    document.write("<TD>");
    document.write((i * 16 + j) + ": " +
                   String.fromCharCode(i * 16 + j));
    document.write("</TD>");
  }
  document.writeln("</TR>");
}
</SCRIPT>
</TABLE>
</BODY>
</HTML>
```



0:	1: □	2: □	3: □	4: □	5: □	6: □	7: □	8: □	9:	10:	11:	12:	13:	14: □	15: □
16: □	17: □	18: □	19: □	20: □	21: □	22: □	23: □	24: □	25: □	26: □	27: □	28: □	29: □	30: □	31: □
32:	33: !	34: "	35: #	36: $	37: %	38: &	39: '	40: (	41:)	42: *	43: +	44: ,	45: -	46: .	47: /
48: 0	49: 1	50: 2	51: 3	52: 4	53: 5	54: 6	55: 7	56: 8	57: 9	58: :	59: ;	60: <	61: =	62: >	63: ?
64: @	65: A	66: B	67: C	68: D	69: E	70: F	71: G	72: H	73: I	74: J	75: K	76: L	77: M	78: N	79: O
80: P	81: Q	82: R	83: S	84: T	85: U	86: V	87: W	88: X	89: Y	90: Z	91: [	92: \	93:]	94: ^	95: _
96: `	97: a	98: b	99: c	100: d	101: e	102: f	103: g	104: h	105: i	106: j	107: k	108: l	109: m	110: n	111: o
112: p	113: q	114: r	115: s	116: t	117: u	118: v	119: w	120: x	121: y	122: z	123: {	124: |	125: }	126: ~	127: □
128: ?	129: ?	130: ?	131: ?	132: ?	133: ?	134: ?	135: ?	136: ?	137: ?	138: ?	139: ?	140: ?	141: ?	142: ?	143: ?
144: ?	145: ?	146: ?	147: ?	148: ?	149: ?	150: ?	151: ?	152: ?	153: ?	154: ?	155: ?	156: ?	157: ?	158: ?	159: ?
160:	161: ¡	162: ¢	163: £	164: ¤	165: ¥	166: ¦	167: §	168: ¨	169: ©	170: ª	171: «	172: ¬	173: -	174: ®	175: ¯
176: °	177: ±	178: ²	179: ³	180: ´	181: µ	182: ¶	183: ·	184: ¸	185: ¹	186: º	187: »	188: ¼	189: ½	190: ¾	191: ¿
192: À	193: Á	194: Â	195: Ã	196: Ä	197: Å	198: Æ	199: Ç	200: È	201: É	202: Ê	203: Ë	204: Ì	205: Í	206: Î	207: Ï
208: Ð	209: Ñ	210: Ò	211: Ó	212: Ô	213: Õ	214: Ö	215: ×	216: Ø	217: Ù	218: Ú	219: Û	220: Ü	221: Ý	222: Þ	223: ß
224: à	225: á	226: â	227: ã	228: ä	229: å	230: æ	231: ç	232: è	233: é	234: ê	235: ë	236: ì	237: í	238: î	239: ï
240: ð	241: ñ	242: ò	243: ó	244: ô	245: õ	246: ö	247: ÷	248: ø	249: ù	250: ú	251: û	252: ü	253: ý	254: þ	255: ÿ

Figura 9.5: Tabla de caracteres ASCII

indexOf(valorBuscado [, inicio])

Devuelve la primera posición de `valorBuscado` dentro de la cadena a partir del principio de la cadena (o de `inicio` que es opcional) o -1 si no se encuentra. Ejemplo:

Ejemplo 9.49

```
var cuenta = 0;
var pos;
var cad = "Cadena con unas letras"

pos = cad.indexOf("a");
while(pos != -1)
{
  cuenta++;
  pos = cad.indexOf("a", pos + 1);
}
document.write("La cadena contiene " + cuenta + " aes");
```

El ejemplo anterior produce la siguiente salida:

Ejemplo 9.50

```
La cadena contiene 4 aes
```

lastIndexOf(valorBuscado [, inicio])

Similar a la función `indexOf`, pero busca desde el final de la cadena o el `inicio` que se indique hacia el principio. Ejemplo:

Ejemplo 9.51

```
var pos;
var cad = "Cadena con unas letras"

pos = cad.lastIndexOf("a");
document.write("La última a está en " + pos + "<BR>");
pos = cad.lastIndexOf("a", pos - 1);
document.write("La penúltima a está en " + pos + "<BR>");
```

Este ejemplo genera el siguiente resultado:

Ejemplo 9.52

```
La última a está en 20
La penúltima a está en 13
```

replace(exRegular, nuevaCadena)

Busca una expresión regular (`exRegular`) en una cadena y cada vez que se encuentra se sustituye por `nuevaCadena`. Si se quiere buscar varias veces, hay que incluir el modificador `g` en la expresión regular para realizar una búsqueda global. Si no se quiere distinguir minúsculas y mayúsculas, hay que incluir el modificador `i`. Ejemplo:

Ejemplo 9.53

```
1 var cad = "Juan es hermano de Juan Luis";
2 var aux;
3
4 aux = cad.replace(/Juan/gi, "Jose");
5 document.write(cad + "<BR>");
6 document.write(aux + "<BR>");
```

El código anterior muestra el siguiente resultado:

Ejemplo 9.54

```
1 Juan es hermano de Juan Luis
2 Jose es hermano de Jose Luis
```

slice(inicio [, fin])

Extrae un trozo de una cadena desde `inicio` hasta `fin - 1`. Si no se indica `fin`, se extrae hasta el final de la cadena. Ejemplo:

Ejemplo 9.55

```
1 var cad = "Juan es hermano de Juan Luis";
2 var aux;
3
4 aux = cad.slice(8);
5 document.write(aux + "<BR>");
6 aux = cad.slice(8, 15);
7 document.write(aux + "<BR>");
```

Este ejemplo produce el siguiente resultado:

Ejemplo 9.56

```
1 hermano de Juan Luis
2 hermano
```

split(separador [, limite])

Divide una cadena en función del `separador`. El resultado de la división se almacena en un array. El valor de `limite` indica el número máximo de divisiones que se pueden hacer. Ejemplo:

Ejemplo 9.57

```
var cad = "Juan es hermano de Juan Luis";
var aux, i;

aux = cad.split(" ");
for(i = 0; i < aux.length; i++)
  document.write("Posicion " + i + ": " + aux[i] + "<BR>");
```

El ejemplo anterior genera el siguiente resultado:

Ejemplo 9.58

```
Posicion 0: Juan
Posicion 1: es
Posicion 2: hermano
Posicion 3: de
Posicion 4: Juan
Posicion 5: Luis
```

9.7. Operaciones matemáticas

JavaScript dispone del objeto `Math` que posee una serie de propiedades y métodos que permiten trabajar con constantes y funciones matemáticas. Todos los métodos y propiedades son estáticos, por lo que no hace falta crear un objeto de tipo `Math` para usarlos. En el Cuadro 9.8 se muestran las principales propiedades del objeto `Math`.

Los principales métodos del objeto Math son: `abs`, `acos`, `asin`, `atan`, `ceil`, `cos`, `sin`, `tan`, `exp`, `floor`, `log`, `max`, `min`, `pow`, `random`, `round` y `sqrt`.

abs(num)

Devuelve el valor absoluto de un número.

Ejemplo 9.59

```
document.write(Math.abs(3.5) + "<BR>");
document.write(Math.abs(2) + "<BR>");
document.write(Math.abs(-0.5) + "<BR>");
document.write(Math.abs(-4) + "<BR>");
```

Propiedad	Descripción
E	Constante de Euler, la base de los logaritmos naturales (neperianos). Aproximadamente 2.718
LN10	El logaritmo natural de 10. Aproximadamente 2.302
LN2	El logaritmo natural de 2. Aproximadamente 0.693
LOG10E	El logaritmo decimal (base 10) de E. Aproximadamente 0.434
LOG2E	El logaritmo en base 2 de E. Aproximadamente 1.442
PI	Razón de la circunferencia de un círculo a su diámetro. Aproximadamente 3.141
SQRT1_2	Raíz cuadrada de 1 / 2. Aproximadamente 0.707
SQRT2	Raíz cuadrada de 2. Aproximadamente 1.414

Cuadro 9.8: Propiedades del objeto Math

El código anterior produce la siguiente salida:

Ejemplo 9.60

```
3.5
2
.5
4
```

acos(num), asin(num), atan(num)

Devuelve el arcocoseno, arcoseno y arcotangente en radianes de un número.

ceil(num)

Devuelve el entero menor mayor o igual que un número. Ejemplo:

Ejemplo 9.61

```
document.write("ceil(2): " + Math.ceil(2) + "<BR>");
document.write("ceil(2.0): " + Math.ceil(2.0) + "<BR>");
document.write("ceil(2.05): " + Math.ceil(2.05) + "<BR>");
```

El código anterior genera la siguiente salida:

Ejemplo 9.62

```
ceil(2): 2
ceil(2.0): 2
ceil(2.05): 3
```

cos(num), sin(num), tan(num)

Devuelve el coseno, seno y tangente de un ángulo expresado en radianes.

exp(num)

Devuelve E elevado a un número.

floor(num)

Devuelve el mayor entero menor o igual que un número. Ejemplo:

Ejemplo 9.63

```
1 document.write("floor(2): " + Math.floor(2) + "<BR>");
2 document.write("floor(2.0): " + Math.floor(2.0) + "<BR>");
3 document.write("floor(2.05): " + Math.floor(2.05) + "<BR>");
4 document.write("floor(2.99): " + Math.floor(2.99) + "<BR>");
```

Este ejemplo muestra el siguiente resultado:

Ejemplo 9.64

```
1 floor(2): 2
2 floor(2.0): 2
3 floor(2.05): 2
4 floor(2.99): 2
```

log(num)

Devuelve el logaritmo natural (en base E) de un número.

max(num1, num2), min(num1, num2)

Devuelve el mayor (mínimo) de dos números. Ejemplo:

Ejemplo 9.65

```
1 document.write("max(3, 5): " + Math.max(3, 5) + "<BR>");
2 document.write("min(3, 5): " + Math.min(3, 5) + "<BR>");
```

El código anterior genera la siguiente salida:

Ejemplo 9.66

```
1 max(3, 5): 5
2 min(3, 5): 3
```

pow(base, exponente)

Devuelve la base elevada al exponente. Ejemplo:

Ejemplo 9.67

```
1 document.write("pow(2, 3): " + Math.pow(2, 3) + "<BR>");
2 document.write("pow(3, 4): " + Math.pow(3, 4) + "<BR>");
3 document.write("pow(5, 0): " + Math.pow(5, 0) + "<BR>");
```

El ejemplo anterior genera el siguiente resultado:

Ejemplo 9.68

```
1 pow(2, 3): 8
2 pow(3, 4): 81
3 pow(5, 0): 1
```

random()

Produce un número pseudoaleatorio entre 0 y 1. Ejemplo:

Ejemplo 9.69

```
1 document.write("random(): " + Math.random() + "<BR>");
2 document.write("random(): " + Math.random() + "<BR>");
```

Este ejemplo produce la siguiente salida:

Ejemplo 9.70

```
1 random(): .32779162476197754
2 random(): .8945828404144374
```

round(num)

Devuelve el entero más cercano a un número. Si la parte decimal es menor que 0.5, lo redondea al entero mayor menor o igual que el número; si la parte decimal es igual o mayor que 0.5, lo redondea al entero menor mayor o igual que el número. Ejemplo:

Ejemplo 9.71

```
1 document.write("round(3.49): " + Math.round(3.49) + "<BR>");
2 document.write("round(3.5): " + Math.round(3.5) + "<BR>");
3 document.write("round(3.51): " + Math.round(3.51) + "<BR>");
```

Produce la siguiente salida:

Ejemplo 9.72

```
round(3.49): 3
round(3.5): 4
round(3.51): 4
```

sqrt(num)

Devuelve la raíz cuadrada de un número.

9.8. Validación de formularios

Una de las principales ventajas que ofrece *JavaScript* es la posibilidad de realizar validaciones inmediatas de la información introducida por un usuario en un formulario. Aunque existe la alternativa de utilizar un programa **CGI** o una página **ASP** que realice la misma función en el servidor, poder llevar a cabo este proceso en el cliente, ahorra tiempo de espera a los usuarios, disminuye el número de conexiones al servidor y limita la carga del servidor.

Antes de estudiar la validación de formularios, conviene leer el Capítulo 10, ya que en él se explica **DOM**, una representación de los distintos elementos que componen una página web. Mediante **DOM** se accede a los formularios y sus controles.

A continuación se muestran cuatro casos típicos de validación de formularios: validación campo nulo, validación alfabética, validación numérica y validación de una fecha.

9.8.1. Validación campo nulo

Muchas veces interesa comprobar si se ha introducido información en un campo: que no se deje en blanco algún dato. En el siguiente ejemplo, la función `compruebaVacio` determina si en un campo se ha introducido información, teniendo en cuenta que se considera información todo lo que sea distinto de vacío, espacios en blanco o tabuladores. En la Figura 9.6 vemos como se visualiza el siguiente código en un navegador.

Ejemplo 9.73

```
<HTML>
<HEAD>
<TITLE>Validación de un campo vacío</TITLE>
<SCRIPT LANGUAGE="JavaScript">
error = new creaError();
errores = new Array();
errores[0] = "Campo vacío";
```

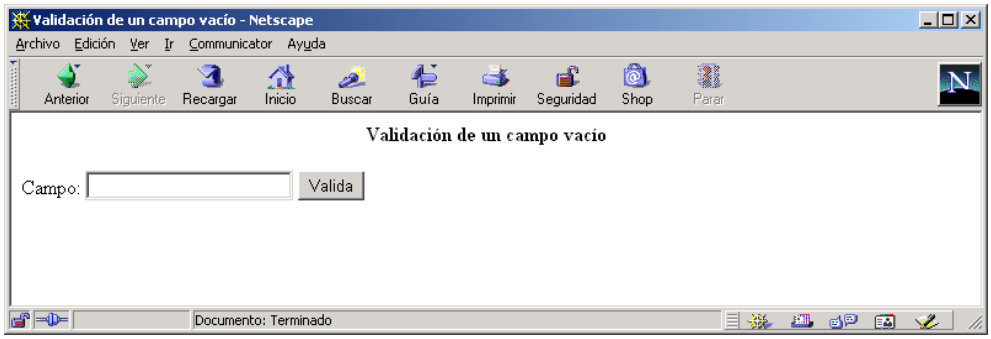


Figura 9.6: Validación campo nulo

```
8  errores[1] = "Campo sólo contiene blancos o tabuladores";
9
10 function creaError()
11 {
12   this.valor = 0;
13   return this;
14 }
15
16 function compruebaVacio(contenido, error)
17 {
18   if(contenido.length == 0)
19   {
20     error.valor = 0;
21     error.posicion = 0;
22     return true;
23   }
24   for(var i = 0; i < contenido.length; i++)
25     if(contenido.charAt(i) != ' ' &&
26        contenido.charAt(i) != '\t')
27       return false;
28
29   error.valor = 1;
30   return true;
31 }
32
33 function valida()
34 {
35   if(compruebaVacio(document.formulario.campo.value, error))
36     alert("El campo no es correcto: " + errores[error.valor]);
37   else
```

```
38      alert("El campo es correcto: Campo no vacío");
39  }
40  </SCRIPT>
41  </HEAD>
42  <BODY>
43  <FORM NAME="formulario">
44  <B><CENTER>Validación de un campo vacío</CENTER></B>
45  <BR>
46  Campo: <INPUT TYPE="TEXT" NAME="campo">
47  <INPUT TYPE="BUTTON" VALUE="Valida" ONCLICK="valida()">
48  </FORM>
49  </BODY>
50  </HTML>
```

9.8.2. Validación alfabética

Un tipo elemental de información que aparece en muchos formularios es el tipo alfabético compuesto por los caracteres alfabéticos del idioma en particular con el que se trabaje. En el siguiente ejemplo, las funciones `compruebaAlfa`, `compruebaAlfaMin` y `compruebaAlfaMay` validan que un valor sea una palabra, una palabra en minúsculas y una palabra en mayúsculas, incluyendo la posibilidad de que existan letras acentuadas o con diéresis. En la Figura 9.7 vemos el aspecto de la página visualizada en un navegador.

Ejemplo 9.74

```
1  <HTML>
2  <HEAD>
3  <TITLE>Validación alfabética de un campo</TITLE>
4  <SCRIPT LANGUAGE="JavaScript">
5  mayusculas = "ABCDEFGHIJKLMNÑOPQRSTUVWXYZÁÉÍÓÚÜ";
6  minusculas = "abcdefghijklmnñopqrstuvwxyzáéíóúü";
7
8  error = new creaError();
9  errores = new Array();
10 errores[1] = "Falta carácter alfabético";
11 errores[2] = "Falta carácter alfabético en minúsculas";
12 errores[3] = "Falta carácter alfabético en mayúsculas";
13
14 function creaError()
15 {
16   this.valor = 0;
17   this.posicion = 0;
18   return this;
19 }
```

```
20
21 function esMinuscula(c)
22 {
23   return minusculas.indexOf(c) >= 0;
24 }
25
26
27 function esMayuscula(c)
28 {
29   return mayusculas.indexOf(c) >= 0;
30 }
31
32 function esLetra(c)
33 {
34   return esMinuscula(c) || esMayuscula(c);
35 }
36
37 function compruebaAlfa(contenido, error)
38 {
39   for(var i = 0; i < contenido.length; i++)
40     if(!esLetra(contenido.charAt(i)))
41     {
42       error.valor = 1;
43       error.posicion = i + 1;
44       return false;
45     }
46
47   return true;
48 }
49
50 function compruebaAlfaMin(contenido, error)
51 {
52   for(var i = 0; i < contenido.length; i++)
53     if(!esMinuscula(contenido.charAt(i)))
54     {
55       error.valor = 2;
56       error.posicion = i + 1;
57       return false;
58     }
59
60   return true;
61 }
62
63 function compruebaAlfaMay(contenido, error)
64 {
65   for(var i = 0; i < contenido.length; i++)
```

```
66      if(!esMayuscula(contenido.charAt(i)))
67      {
68        error.valor = 3;
69        error.posicion = i + 1;
70        return false;
71      }
72
73    return true;
74  }
75
76  function valida(valor)
77  {
78    var correcto;
79    var contenido = document.formulario.campo.value;
80
81    switch(valor)
82    {
83      case 1:
84        correcto = compruebaAlfa(contenido, error);
85        break;
86
87      case 2:
88        correcto = compruebaAlfaMin(contenido, error);
89        break;
90
91      case 3:
92        correcto = compruebaAlfaMay(contenido, error);
93        break;
94
95      default:
96        break;
97    }
98
99    if(correcto)
100     alert("El campo es válido");
101   else
102     alert("El campo no es válido: " + errores[error.valor] +
103           " en la posición " + error.posicion);
104 }
105 </SCRIPT>
106 <BODY>
107 <FORM NAME="formulario">
108 <B><CENTER>Validación alfabética de un campo</CENTER></B>
109 <BR>
110 Campo: <INPUT TYPE="TEXT" NAME="campo">
111 <INPUT TYPE="BUTTON" VALUE="Palabra" ONCLICK="valida(1)">
```

```
112 <INPUT TYPE="BUTTON" VALUE="Minúsculas" ONCLICK="valida(2)">
113 <INPUT TYPE="BUTTON" VALUE="Mayúsculas" ONCLICK="valida(3)">
114 </FORM>
115 </BODY>
116 </HTML>
```

Figura 9.7: Validación alfabética

9.8.3. Validación numérica

Se pueden distinguir tres tipos de formatos numéricos básicos:

- Número natural: formado por los caracteres numéricos.
- Número entero: formado por un signo inicial opcional (+, -), seguido de un número natural.
- Número real: formado por un signo inicial opcional (+, -), seguido de caracteres numéricos y, opcionalmente, seguido del separador decimal y otra serie de caracteres numéricos.

En el siguiente ejemplo, las funciones `compruebaNatural`, `compruebaEntero` y `compruebaReal` validan que un valor sea un número natural, un número entero y un número real[5] respectivamente. En la Figura 9.8 vemos el aspecto de la página visualizada en un navegador.

Ejemplo 9.75

```
1 <HTML>
2 <HEAD>
```

[5]Se permite el punto y la coma como separadores decimales.

```
3  <TITLE>Validación numérica de un campo</TITLE>
4  <SCRIPT LANGUAGE="JavaScript">
5  numeros = "0123456789";
6  puntoDecimal = ".,";
7  signos = "+-";
8
9  error = new creaError();
10 errores = new Array();
11 errores[1] = "Campo vacío no contiene ningún valor";
12 errores[2] = "Carácter ilegal en un número";
13 errores[3] = "Carácter ilegal";
14 errores[4] = "Sólo ha insertado un signo";
15 errores[5] = "Parte decimal vacía";
16
17 function creaError()
18 {
19   this.valor = 0;
20   this.posicion = 0;
21   return this;
22 }
23
24 function numero(c)
25 {
26   return numeros.indexOf(c) >= 0;
27 }
28
29 function signo(c)
30 {
31   return signos.indexOf(c) >= 0;
32 }
33
34 function compruebaNatural(contenido, error)
35 {
36   if(contenido.length == 0)
37   {
38     error.valor = 1;
39     error.posicion = 1;
40     return false;
41   }
42
43   for(var i = 0; i < contenido.length; i++)
44     if(!numero(contenido.charAt(i)))
45     {
46       error.valor = 2;
47       error.posicion = i + 1;
48       return false;
```

```
49      }
50
51    return true;
52  }
53
54  function signoCorrecto(contenido, error)
55  {
56    if(contenido.length == 0)
57    {
58      error.valor = 1;
59      error.posicion = 1;
60      return false;
61    }
62
63    if(!numero(contenido.charAt(0)) &&
64       !signo(contenido.charAt(0)))
65    {
66      error.valor = 3;
67      error.posicion = 1;
68      return false;
69    }
70
71    return true;
72  }
73
74  function compruebaEntero(contenido, error)
75  {
76    if(!signoCorrecto(contenido, error))
77      return false;
78
79    if(numero(contenido.charAt(0)))
80      var aux = compruebaNatural(contenido, error);
81    else
82    {
83      var aux = compruebaNatural(contenido.substring(1,
84                                 contenido.length), error);
85      if(!aux)
86        error.posicion++;
87      if(error.valor == 1)
88      {
89        error.valor = 4;
90        error.posicion = 1;
91      }
92    }
93
94    return aux;
```

```
95 }
96
97 function compruebaReal(contenido, error)
98 {
99   var aux = compruebaEntero(contenido, error);
100  var posicion = error.posicion - 1;
101
102  if(!aux && error.valor == 2 &
103     puntoDecimal.indexOf(contenido.charAt(posicion)) >= 0)
104  {
105    var aux = compruebaNatural(contenido.substring(
106                      error.posicion, contenido.length), error);
107    if(!aux && error.valor == 1)
108    {
109      error.valor = 5;
110    }
111    if(!aux)
112      error.posicion += posicion + 1;
113  }
114
115  return aux;
116 }
117
118 function valida(valor)
119 {
120   var correcto;
121   var contenido = document.formulario.campo.value;
122
123   switch(valor)
124   {
125     case 1:
126       correcto = compruebaNatural(contenido, error);
127       break;
128
129     case 2:
130       correcto = compruebaEntero(contenido, error);
131       break;
132
133     case 3:
134       correcto = compruebaReal(contenido, error);
135       break;
136
137     default:
138       break;
139   }
140
```

```
141   if(correcto)
142     alert("El campo es válido");
143   else
144     alert("El campo no es válido: " + errores[error.valor] +
145           " en la posición " + error.posicion);
146 }
147 </SCRIPT>
148 <BODY>
149 <FORM NAME="formulario">
150 <B><CENTER>Validación numérica de un campo</CENTER></B>
151 <BR>
152 Campo: <INPUT TYPE="TEXT" NAME="campo">
153 <INPUT TYPE="BUTTON" VALUE="Natural" ONCLICK="valida(1)">
154 <INPUT TYPE="BUTTON" VALUE="Entero" ONCLICK="valida(2)">
155 <INPUT TYPE="BUTTON" VALUE="Real" ONCLICK="valida(3)">
156 </FORM>
157 </BODY>
158 </HTML>
```

Figura 9.8: Validación numérica

9.8.4. Validación de una fecha

La validación de una fecha es una de las más complejas, debido a la diversidad de formatos que existen, tanto entre distintos países como en un mismo país. Además hay que tener en cuenta una serie de consideraciones (número de días de cada mes, años bisiestos, etc.) que complican la validación.

En el siguiente ejemplo, se valida una fecha de acuerdo al formato estándar "dd/mm/aaaa", donde "dd" son los dígitos (1 o 2) que representan el día, "mm" son los

dígitos que representan el mes (1 o 2) y "aaaa" son los dígitos que representan el año (de 1 a 4). En la Figura 9.9 se puede ver como se muestra la página en un navegador.

La función `compruebaFecha` comprueba que un fecha sea correcta según el formato anterior. La primera parte de la función comprueba que la fecha introducida concuerde con el formato deseado; para ello, extrae de la fecha el día, el mes y el año. La segunda parte de la función comprueba que la fecha introducida sea una fecha real; para ello, en vez de realizar complejos cálculos, se emplea el objeto `Date` de *JavaScript*:

- Se crea una nueva fecha a partir del día, el mes y el año introducidos.
- Se comparan el día, el mes y el año de la nueva fecha con los valores de la fecha introducida.
- Si todos los valores coinciden, significa que la fecha introducida es correcta.

Por ejemplo, si se introduce `29/02/2002`, como es una fecha que no existe, el objeto `Date` la convierte en `01/03/2002`, por lo que se detecta que es una fecha incorrecta.

Ejemplo 9.76

```
<HTML>
<HEAD>
<TITLE>Validación de una fecha</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function compruebaFecha(fecha)
{
  // Comprueba el formato de la fecha
  // Dia
  i1 = fecha.indexOf("/");
  if(i1 == -1)
  {
    alert("Formato de fecha incorrecto");
    return;
  }
  dia = fecha.substring(0, i1);

  // Mes
  i2 = fecha.indexOf("/", i1 + 1)
  if(i2 == -1)
  {
    alert("Formato de fecha incorrecto");
    return;
  }
  mes = fecha.substring(i1 + 1, i2)

  // Anyo
  anyo = fecha.substring(i2 + 1, fecha.length);
```

```
28
29    // Comprueba formato de dia, mes y anyo
30    if(isNaN(dia) || isNaN(mes) || isNaN(anyo))
31    {
32      alert("Formato de fecha incorrecto");
33      return;
34    }
35
36    if(dia == "" || mes == "" || anyo == "")
37    {
38      alert("Formato de fecha incorrecto");
39      return;
40    }
41
42    // Comprueba que la fecha sea real
43    // Crea una nueva fecha
44    // Los meses empiezan desde 0
45    f = new Date(anyo, mes - 1, dia);
46
47    if(parseInt(dia, 10) != f.getDate() ||
48       parseInt(mes, 10) != (f.getMonth() + 1) ||
49       parseInt(anyo, 10) != f.getFullYear())
50    {
51      alert("Formato de fecha incorrecto");
52      return;
53    }
54  }
55  </SCRIPT>
56  <BODY>
57  <FORM NAME="formulario">
58  <B><CENTER>Validación de una fecha</CENTER></B>
59  <BR>
60  Fecha: <INPUT TYPE="TEXT" NAME="fecha" MAXLENGTH="10">
61  <INPUT TYPE="BUTTON" VALUE="Valida" ONCLICK="compruebaFecha(fecha.value)">
62  </FORM>
63  </BODY>
64  </HTML>
```

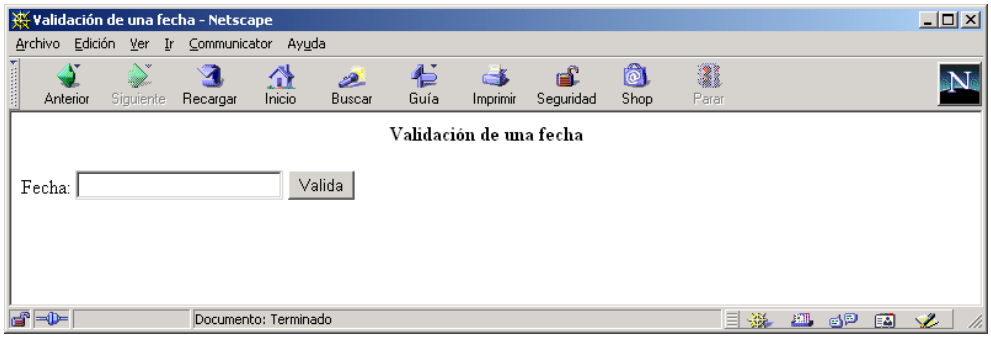


Figura 9.9: Validación de una fecha

Capítulo 10

Modelo de Objetos de Documento

El Modelo de Objetos de Documento es un interfaz que permite acceder y modificar la estructura y contenido de una página web. Especifica como se puede acceder a los distintos elementos (enlaces, imágenes, formularios, etc.) de una página y como se pueden modificar. En este capítulo vamos a describir los objetos que componen este modelo, sus propiedades, métodos y eventos.

Índice General

10.1. Introducción

El Modelo de Objetos de Documento **DOM** se suele confundir con los lenguajes de *script*: se piensa que forma parte de ellos, cuando en realidad son independientes

uno del otro.

El modelo de objetos permite acceder a los elementos **HTML** de un documento desde los lenguajes de *script*. Para lograrlo, se crean una serie de objetos que representan (exponen) dichos elementos. Estos objetos guardan entre ellos unas relaciones de parentesco (se establece una jerarquía) que refleja la estructura lógica de una página **HTML**: una página se presenta en una ventana, que posee un documento, el cual a su vez puede tener una serie de formularios, que pueden contener elementos como botones, cuadros de texto, etc., los cuales tienen a su vez una serie de propiedades, etc.

Un problema que presenta el modelo de objetos es que se suele modificar de una versión de navegador a otra, añadiéndose nuevos objetos y modificándose la jerarquía existente entre ellos.

Además, al igual que ocurre con los lenguajes de *script*, cada navegador posee su propio modelo de objetos, que coinciden entre ellos en algunos objetos, propiedades y métodos. Existe un intento de estandarización por parte de **W3C** del modelo de objetos de documento.

10.2. Modelo de objetos en Netscape Communicator

Cuando se carga una página en el navegador, se crean una serie de objetos *JavaScript* llamados objetos del navegador (*Navigator objects*) con propiedades basadas en el documento **HTML** que se está mostrando. Estos objetos poseen una jerarquía que refleja la estructura de la página **HTML**. En la Figura 10.1 vemos los principales objetos de la jerarquía del modelo de objetos en Netscape Communicator 4.0 y superiores (en versiones superiores puede ser que se vea ampliado).

En esta jerarquía, los descendiente de un objeto se consideran propiedades de dicho objeto, aunque por sí mismos también son objetos. Por ejemplo, un formulario llamado `form1` es un objeto y también una propiedad del objeto `document` y se accede como `document.form1`.

Como todos los objetos cuelgan del objeto `window`, éste se pude obviar y no escribir cuando se accede a un objeto de la ventana actual[1]. Así, `window.navigator.appName` y `navigator.appName` representan la misma propiedad y las sentencias `window.document.writeln()` y `document.writeln()` llaman al mismo método.

Cualquier página posee los siguientes objetos:

- `document`: contiene una serie de propiedades basadas en el contenido del documento, como su título, su color de fondo, sus enlaces y formularios.

- `history`: contiene una serie de propiedades que representan las URLs que el cliente ha visitado previamente (contiene el historial de navegación).

[1] Cuando se quiera acceder a un objeto de otra ventana, habrá que indicar la ventana.

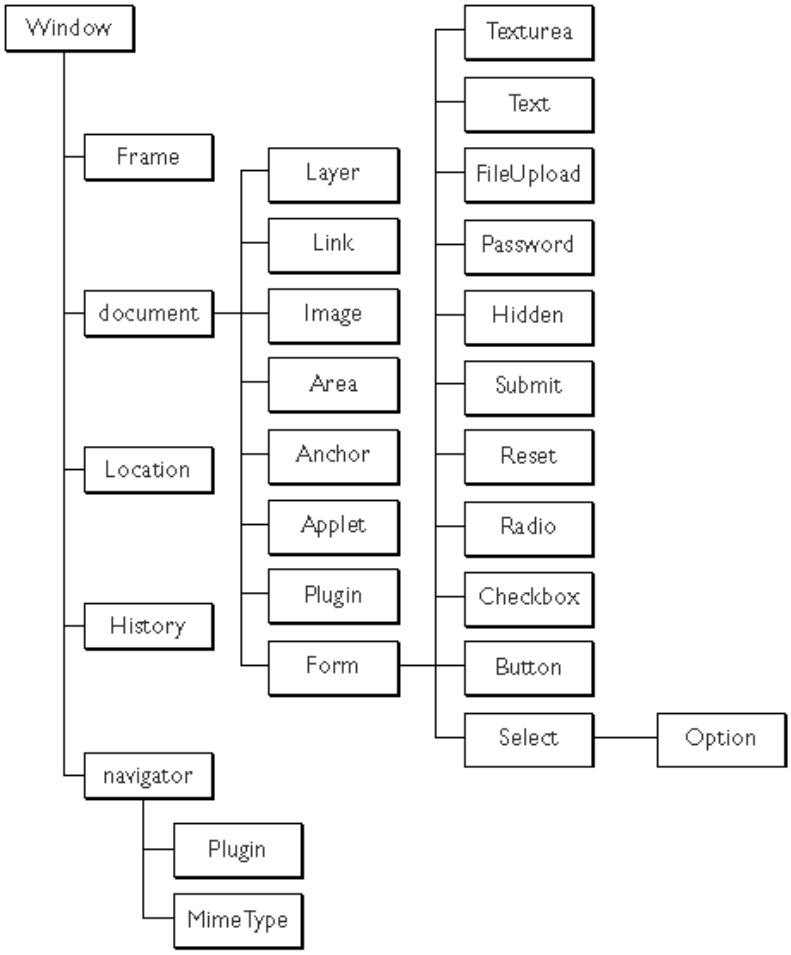


Figura 10.1: Modelo de objetos en Netscape Communicator

- `location`: posee propiedades basadas en la **URL** actual.
- `navigator`: posee propiedades que representan el nombre y la versión de navegador que se está usando, propiedades para los tipos *Multipurpose Internet Mail Extensions* (**MIME**) que acepta el cliente y para los *plug-ins* que están instalados en el cliente.
- `window`: se trata del objeto de más alto nivel; tiene propiedades que se aplican a la ventana completa. Cada ventana hija en un documento dividido en marcos (*frames*) posee su propio objeto `window`.

Dependiendo del contenido de la página, el documento puede poseer otra serie de objetos. Por ejemplo, un formulario (definido mediante la etiqueta `<FORM> ... </FORM>`) en el documento tiene su correspondiente objeto `form` en la jerarquía de objetos.

10.2.1. Objeto document

Posee información sobre el documento actual y posee métodos que permiten mostrar una salida en formato **HTML** al usuario. Se crea un objeto `document` por cada etiqueta `<BODY> ... </BODY>`.

Propiedades

alinkColor Atributo `ALINK` de la etiqueta `BODY`.

anchors Array que contiene una entrada para cada ancla del documento.

applets Array que contiene una entrada por cada *applet* en el documento.

bgColor Atributo `BGCOLOR` de la etiqueta `BODY`.

classes Crea un objeto Style que se usa para especificar el estilo de las etiquetas **HTML**.

cookie Especifica una *cookie*.

domain Nombre de dominio del servidor que sirvió el documento.

embeds Array que contiene una entrada por cada *plug-in* del documento.

fgColor Atributo `TEXT` de la etiqueta `BODY`.

formName Una propiedad por cada formulario con nombre en el documento.

forms Array que contiene una entrada por cada formulario en el documento.

height Altura del documento en pixels.

ids Crea un objeto Style que permite especificar el estilo de cada etiqueta **HTML**.

images Array que contiene una entrada para cada imagen del documento.

lastModified Fecha en la que el documento se modificó por última vez.

layers Array que contiene un entrada por cada capa (*layer*) en el documento.

linkColor Atributo `LINK` de la etiqueta `BODY`.

links Array que contiene una entrada por cada enlace en el documento.

plugins Array que contiene una entrada por cada *plug-in* en el documento.

referrer Especifica la **URL** del documento que llamó al actual.

tags Crea un objeto Style que permite especificar los estilos de las etiquetas **HTML**.

title Contenido de la etiqueta `TITLE` de la sección `HEAD`.

URL **URL** completa del documento.

vlinkColor Atributo `VLINK` de la etiqueta `BODY`.

width Anchura del documento en pixels.

Métodos

captureEvents Establece que el documento capture todos los eventos del tipo especificado.

close Cierra un flujo de salida y fuerza que la información se muestre.

contextual Especifica un objeto Style que puede fijar el estilo de las etiquetas **HTML**.

getSelection Devuelve una cadena que contiene el texto seleccionado.

handleEvent Invoca el manejador del evento especificado.

open Abre un flujo que recibirá la salida de los métodos `write` y `writeln`.

releaseEvents Establece que el documento no capture los eventos del tipo especificado.

routeEvent Pasa un evento a lo largo de la jerarquía de eventos.

write Escribe una o más expresiones **HTML** en el documento.

writeln Escribe una o más expresiones **HTML** en el documento y finaliza con un salto de línea.

Eventos

onClick Se produce cuando un objeto de un formulario o un enlace es pulsado con el botón del ratón.

onDblClick Se produce cuando un objeto de un formulario o un enlace es pulsado dos veces con el botón del ratón.

onKeyDown Se produce cuando el usuario pulsa una tecla.

onKeyPress Se produce cuando el usuario presiona o mantiene pulsada una tecla.

onKeyUp Se produce cuando el usuario deja de pulsar una tecla.

onMouseDown Se produce cuando el usuario pulsa un botón del ratón.

onMouseUp Se produce cuando el usuario deja de pulsar un botón del ratón.

El siguiente código de ejemplo muestra los enlaces que posee un documento.

Ejemplo 10.1

```
<HTML>
<HEAD>
<TITLE>Ejemplo de uso del objeto document</TITLE>
</HEAD>
<BODY>
<A HREF="pag1.html">Enlace a la pagina 5</A>
<BR>
<A HREF="pag2.html">Enlace a la pagina 2</A>
<BR>
<A HREF="pag3.html">Enlace a la pagina 3</A>
<BR><BR>
<SCRIPT LANGUAGE="JavaScript">
document.write("Hay " + document.links.length + " enlaces<BR>\n");

for(i = 0; i < document.links.length; i++)
{
  document.write(document.links[i]);
  document.write("<BR>\n");
}
</SCRIPT>
</BODY>
</HTML>
```

El siguiente ejemplo muestra el valor de los atributos `BGCOLOR`, `TEXT`, `LINK`, `ALINK` y `VLINK`. Además, también aparece la fecha de la última modificación (`lastModified`). En la Figura 10.2 vemos como se visualiza esta página en un navegador.

Ejemplo 10.2

```
<HTML>
<HEAD>
<TITLE>Ejemplo de uso del objeto document</TITLE>
</HEAD>
<BODY BGCOLOR="WHITE" TEXT="BLACK" LINK="NAVY"
 ALINK="BLUE" VLINK="RED">
<SCRIPT LANGUAGE="JavaScript">
document.write("BGCOLOR = " + document.bgColor + "<BR>\n");
document.write("TEXT = " + document.fgColor + "<BR>\n");
document.write("LINK = " + document.linkColor + "<BR>\n");
document.write("ALINK = " + document.alinkColor + "<BR>\n");
document.write("VLINK = " + document.vlinkColor + "<BR>\n");
document.write("<BR>\n");
document.write("lastModified = " + document.lastModified + "<BR>\n");
```

```
15 </SCRIPT>
16 </BODY>
17 </HTML>
```

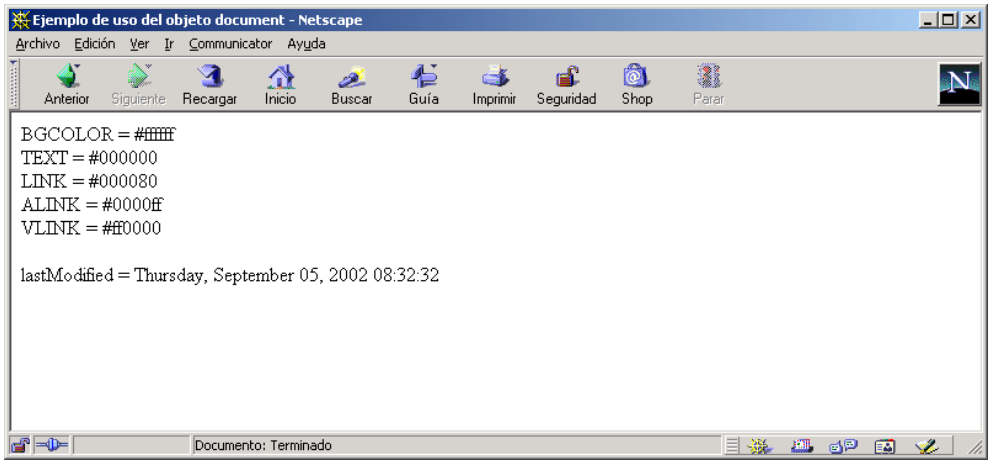


Figura 10.2: Propiedades del objeto document

En el ejemplo anterior, la fecha de la última modificación (lastModified) se muestra con formato un formato inglés. El siguiente ejemplo muestra como transformar cualquier fecha a un formato español. En la Figura 10.3 vemos como se muestra esta página en un navegador.

Ejemplo 10.3

```
1  <HTML>
2  <HEAD>
3  <SCRIPT LANGUAGE="JavaScript">
4  function nombreDia(dia)
5  {
6    var d;
7
8    switch(dia)
9    {
10     case 0:
11       d = "Domingo";
12       break;
13     case 1:
14       d = "Lunes";
```

```
15        break;
16      case 2:
17        d = "Martes";
18        break;
19      case 3:
20        d = "Miércoles";
21        break;
22      case 4:
23        d = "Jueves";
24        break;
25      case 5:
26        d = "Viernes";
27        break;
28      case 6:
29        d = "Sábado";
30        break;
31    }
32
33    return d;
34  }
35
36  function nombreMes(mes)
37  {
38    var m;
39
40    switch(mes)
41    {
42      case 0:
43        m = "Enero";
44        break;
45      case 1:
46        m = "Febrero";
47        break;
48      case 2:
49        m = "Marzo";
50        break;
51      case 3:
52        m = "Ábril";
53        break;
54      case 4:
55        m = "Mayo";
56        break;
57      case 5:
58        m = "Junio";
59        break;
60      case 6:
```

```
61        m = "Julio";
62        break;
63      case 7:
64        m = "Agosto";
65        break;
66      case 8:
67        m = "Septiembre";
68        break;
69    }
70
71    return m;
72  }
73
74  function formatoFecha(fecha)
75  {
76
77    return nombreDia(fecha.getDay()) + " " + fecha.getDate() +
78           " de " + nombreMes(fecha.getMonth()) + " de " +
79           fecha.getFullYear();
80  }
81
82  </SCRIPT>
83  </HEAD>
84  <BODY>
85  <SCRIPT LANGUAGE="JavaScript">
86  var ultima = document.lastModified;
87  var fecha = new Date(ultima);
88
89  document.write("Última fecha de modificación: " + ultima);
90  document.write("<BR><BR>");
91  document.write("Última fecha de modificación: " + formatoFecha(fecha));
92  </SCRIPT>
93  </BODY>
94  </HTML>
```

10.2.2. Cómo acceder a los controles de un formulario

Un documento (`document`) puede poseer varios formularios, cada uno con sus correspondientes controles. Para acceder a los controles de un formulario se tiene que recorrer la jerarquía que se puede ver en la Figura 10.1: objeto `window`[2], objeto `document`, nombre del formulario[3], nombre del control y propiedad del control.

[2] Recordemos que se puede obviar.

[3] También se puede acceder a través de `forms`, un array donde cada posición representa un formulario.

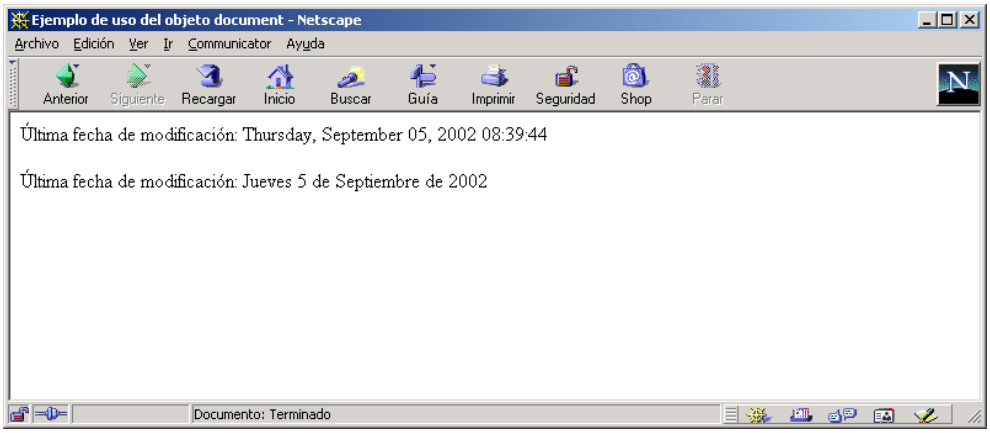


Figura 10.3: Propiedades del objeto document

Por ejemplo, si tenemos el siguiente formulario:

Ejemplo 10.4

```
1 <FORM NAME="miForm">
2 Nombre: <INPUT TYPE="TEXT" NAME="nombre">
3 </FORM>
```

para acceder al valor (propiedad `value`) que contiene el campo de texto `nombre` se tiene que escribir:

Ejemplo 10.5

```
1 document.miForm.nombre.value
```

Hay que prestar mucha atención a las mayúsculas y minúsculas, ya que como se explicó en el Capítulo 9, *JavaScript* es un lenguaje sensible a minúsculas/mayúsculas.

Cada uno de los controles de un formulario tiene una serie de propiedades, métodos y eventos. Todos los controles poseen el método `focus()` que permite fijar el foco sobre el objeto y `blur()` que permite quitarlo.

A continuación mostramos las propiedades más importantes de los controles más usados.

Campos de verificación

Los campos de verificación poseen las siguiente propiedades:

checked Valor booleano que indica si se encuentra seleccionado.

defaultChecked Valor booleano que indica si por defecto se encuentra seleccionado.

value Valor asociado con el control.

Por ejemplo, el siguiente código muestra en una ventana el estado de los campos de verificación de un formulario (si están o no están activados). Además, cuando un campo está activado, muestra también el valor asociado al campo de verificación. En la Figura 10.4 se puede observar como se visualiza esta página en un navegador.

Ejemplo 10.6

```
<HTML>
<HEAD>
<TITLE>Ejemplo de uso del objeto document</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function valida()
{
  var df = document.miForm;

  if(df.ingles.checked)
    alert("Inglés: SÍ" + "\n" +
          "Valor: " + df.ingles.value);
  else
    alert("Inglés: NO");
  if(df.aleman.checked)
    alert("Alemán: SÍ" + "\n" +
          "Valor: " + df.aleman.value);
  else
    alert("Alemán: NO");
  if(df.frances.checked)
    alert("Francés: SÍ" + "\n" +
          "Valor: " + df.frances.value);
  else
    alert("Francés: NO");
}
</SCRIPT>
</HEAD>
<BODY>
<FORM NAME="miForm">
Idiomas:
<BR><BR>
Inglés <INPUT TYPE="CHECKBOX" NAME="ingles" VALUE="ing" CHECKED>
Alemán <INPUT TYPE="CHECKBOX" NAME="aleman" VALUE="ale">
```

```
33 Francés <INPUT TYPE="CHECKBOX" NAME="frances" VALUE="fra">
34 <BR><BR>
35 <INPUT TYPE="BUTTON" VALUE="Comprobar" ONCLICK="valida()">
36 </FORM>
37 </BODY>
38 </HTML>
```

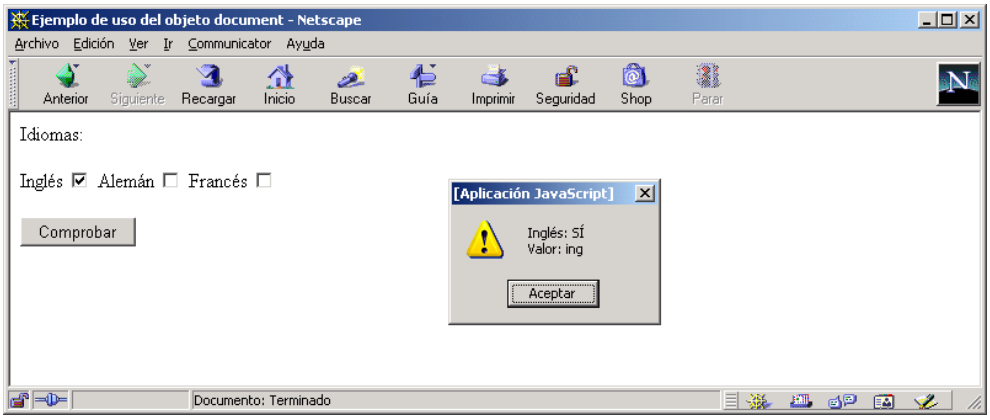


Figura 10.4: Acceso a los controles de un formulario

Campos excluyentes

Posee las mismas propiedades que los campos de verificación:

checked Valor booleano que indica si se encuentra seleccionado.

defaultChecked Valor booleano que indica si por defecto se encuentra seleccionado.

value Valor asociado con el control, que se devuelve al servidor cuando el formulario se envía.

Por ejemplo, el siguiente código muestra en una ventana el valor del campo excluyente seleccionado. En la Figura 10.5 se puede observar como se visualiza esta página en un navegador.

Ejemplo 10.7

```
<HTML>
<HEAD>
<TITLE>Ejemplo de uso del objeto document</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function valida()
{
  var df = document.miForm;

  for(var i = 0; i < df.contrato.length; i++)
  {
    if(df.contrato[i].checked)
      alert("Valor seleccionado: " + df.contrato[i].value);
  }
}
</SCRIPT>
</HEAD>
<BODY>
<FORM NAME="miForm">
Tipo de contrato:
<BR><BR>
Alquiler <INPUT TYPE="RADIO" NAME="contrato" VALUE="1" CHECKED>
Prueba <INPUT TYPE="RADIO" NAME="contrato" VALUE="2">
Venta <INPUT TYPE="RADIO" NAME="contrato" VALUE="3">
<BR><BR>
<INPUT TYPE="BUTTON" VALUE="Comprobar" ONCLICK="valida()">
</FORM>
</BODY>
</HTML>
```

Campos de texto y áreas de texto

Ambos controles poseen estas dos propiedades:

defaultValue Valor por defecto del control.

value Valor actual de control, que se devuelve al servidor cuando el formulario se envía.

Al final del Capítulo 9, cuando se explica la validación de formularios, se pueden ver varios ejemplos donde se hace uso de la propiedad `value`.

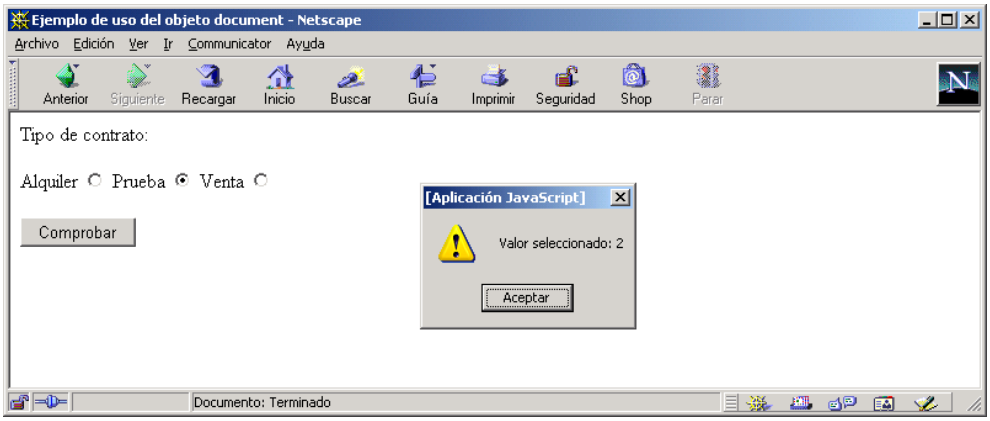


Figura 10.5: Acceso a los controles de un formulario

Listas de selección

Las listas de selección poseen las siguientes propiedades:

length Número de opciones en la lista.

options Array donde cada posición representa una opción (objeto `option`) de la lista.

selectedIndex Índice de la primera opción seleccionada, empezando desde 0. Si ninguna opción está seleccionada, vale -1.

Además, si se quiere obtener el texto y el valor asociado a una opción, se tienen que emplear las propiedades del objeto `option`:

selected Valor booleano que indica si la opción se encuentra seleccionada.

text Texto de la opción que se muestra en la lista.

value Valor asociado con la opción, que se devuelve al servidor cuando la opción está seleccionada.

El siguiente ejemplo muestra como acceder a una lista de selección normal y a una lista de selección múltiple. En la Figura 10.6 se puede observar como se visualiza esta página en un navegador.

Ejemplo 10.8

```
<HTML>
<HEAD>
<TITLE>Ejemplo de uso del objeto document</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function valida()
{
  var df = document.miForm;
  var sel, i;

  // Para un campo sin selección múltiple
  sel = df.provincias.selectedIndex;
  alert("Número de opciones: " + df.length + "\n" +
        "Seleccionado: " + sel + "\n" +
        "Texto: " +
        df.provincias.options[sel].text + "\n" +
        "Valor: " +
        df.provincias.options[sel].value);

  // Para un campo con selección múltiple
  for(i = 0; i < df.paises.length; i++)
  {
    if(df.paises.options[i].selected)
      alert("Seleccionado: " + i + "\n" +
            "Texto: " +
            df.paises.options[i].text + "\n" +
            "Valor: " +
            df.paises.options[i].value);
  }
}
</SCRIPT>
</HEAD>
<BODY>
<FORM NAME="miForm">
Provincias:
<BR><BR>
<SELECT NAME="provincias">
<OPTION VALUE="Alc">Alicante</OPTION>
<OPTION VALUE="Cas">Castellón</OPTION>
<OPTION VALUE="Val">Valencia</OPTION>
</SELECT>
<BR><BR>
Países:
<BR><BR>
<SELECT NAME="paises" MULTIPLE>
<OPTION VALUE="1">España</OPTION>
```

```
<OPTION VALUE="2">Francia</OPTION>
<OPTION VALUE="3">Portugal</OPTION>
</SELECT>
<BR><BR>
<INPUT TYPE="BUTTON" VALUE="Comprobar" ONCLICK="valida()">
</FORM>
</BODY>
</HTML>
```

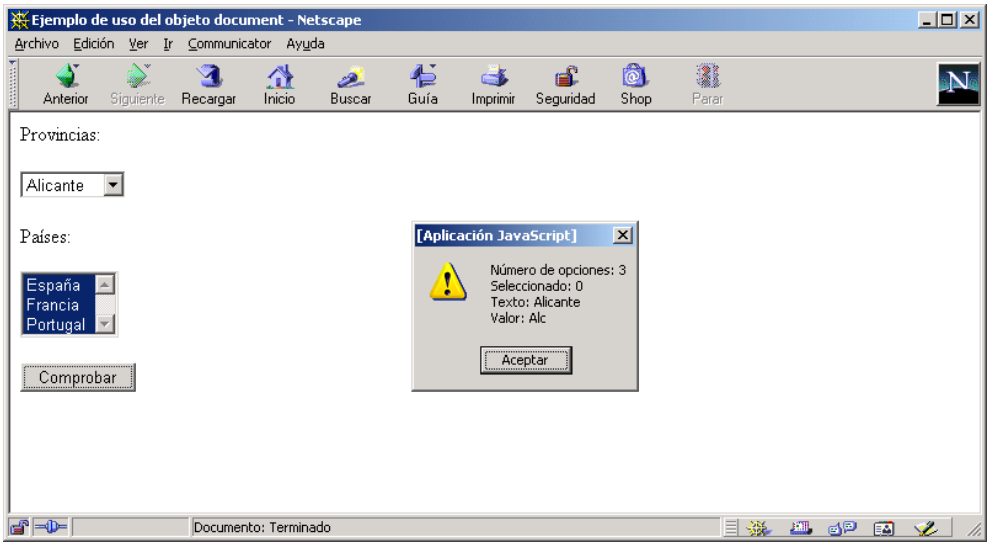


Figura 10.6: Acceso a los controles de un formulario

10.2.3. Objeto history

Contiene un array que almacena las URLs que el usuario ha visitado en la ventana actual. Mediante los métodos que posee este objeto se puede navegar hacia adelante o hacia atrás en el historial.

Propiedades

current Especifica la **URL** actual del historial.

length Indica el número de entradas que almacena el historial.

next Especifica la **URL** de la siguiente entrada en el historial.

previous Especifica la **URL** de la entrada previa en el historial.

Métodos

back Carga la **URL** previa del historial.

forward Carga la siguiente **URL** del historial.

go Carga la **URL** indicada en el historial.

10.2.4. Objeto location

Representa la **URL** completa asociada a un objeto `window`. Cada una de las propiedades de este objeto representa una porción de la **URL**. Se accede mediante la propiedad `location` de un objeto `window`.

Una **URL** se compone de las siguientes partes:

Ejemplo 10.9

```
1 protocol://host:port/pathname#hash?search
```

Propiedades

hash Especifica el valor de un ancla en una **URL**.

host Especifica el nombre de dominio o dirección IP de la **URL**.

hostname Especifica la parte `host:port` de la **URL**.

href Especifica la **URL** entera.

pathname Especifica la ruta de la **URL**.

port Especifica el puerto de comunicaciones que usa el servidor.

protocol Especifica el protocolo de la **URL**.

search Especifica la consulta incluida en la **URL**.

Métodos

reload Recarga el documento actual de la ventana.

replace Carga la **URL** especificada sobre la entrada actual del historial de navegación.

El siguiente ejemplo muestra todas las propiedades del objeto `location`. En la Figura 10.7 vemos el resultado de visualizar la siguiente página en un navegador.

Ejemplo 10.10

```
<HTML>
<HEAD>
<TILLE>Ejemplo de uso del objeto location</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
document.writeln("location.hash = " + location.hash + "<BR>");
document.writeln("location.host = " + location.host + "<BR>");
document.writeln("location.hostname = " + location.hostname + "<BR>");
document.writeln("location.href = " + location.href + "<BR>");
document.writeln("location.pathname = " + location.pathname + "<BR>");
document.writeln("location.port = " + location.port + "<BR>");
document.writeln("location.protocol = " + location.protocol + "<BR>");
document.writeln("location.search = " + location.search + "<BR>");
</SCRIPT>
</BODY>
</HTML>
```

10.2.5. Objeto navigator

Contiene información sobre la versión del navegador que se está empleando. Todas las propiedades de este objeto son de solo lectura.

Propiedades

appCodeName Nombre en clave del navegador.

appName Nombre del navegador.

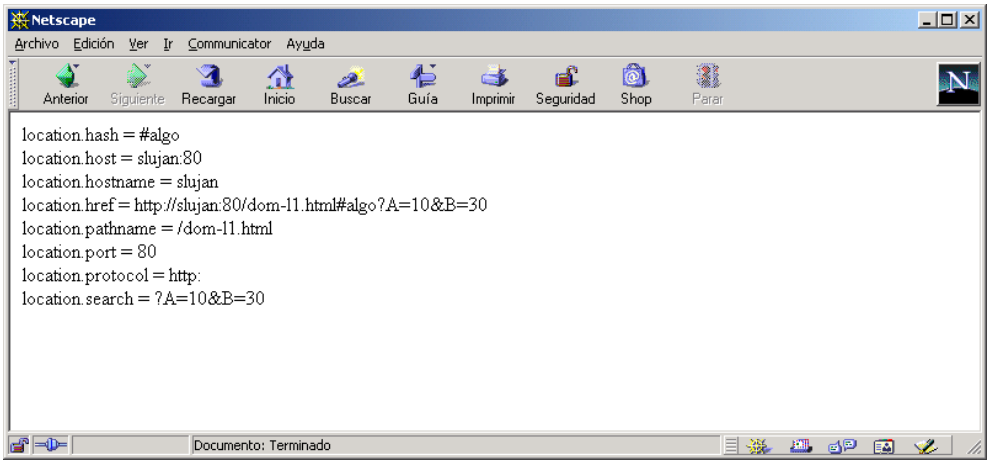


Figura 10.7: Propiedades del objeto location

appVersion Versión del navegador.

language Idioma que emplea el navegador.

mimeTypes Array que contiene todos los tipos **MIME** que soporta el navegador.

platform Indica el tipo de máquina para la que se compiló el navegador.

plugins Array que contiene todos los *plug-ins* instalados.

userAgent Especifica la cabecera **HTTP** `user-agent`.

Métodos

javaEnabled Chequea si la opción *Java* está activada.

plugins.refresh Recarga los documentos que contienen *plug-ins*.

preference Permite a un *script* firmado acceder (lectura y escritura) a ciertas preferencias del navegador.

savePreferences Guarda las preferencias del navegador en `prefs.js` (fichero de preferencias que posee cada usuario).

taintEnabled Especifica si *data tainting* está activo.

El siguiente ejemplo muestra todas las propiedades del objeto `navigator`. En la Figura 10.8 se puede ver como se muestra esta página en Netscape Communicator 4.78 y en la Figura 10.9 se visualiza la misma página en Microsoft Internet Explorer 6.0.

Ejemplo 10.11

```
<HTML>
<HEAD>
<TITLE>Ejemplo de uso del objeto navigator</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
document.writeln("navigator.appCodeName = " + navigator.appCodeName);
document.writeln("<BR>");
document.writeln("navigator.appName = " + navigator.appName);
document.writeln("<BR>");
document.writeln("navigator.appVersion = " + navigator.appVersion);
document.writeln("<BR>");
document.writeln("navigator.language = " + navigator.language);
document.writeln("<BR>");
document.writeln("navigator.mimeTypes = " + navigator.mimeTypes);
document.writeln("<BR>");
document.writeln("navigator.platform = " + navigator.platform);
document.writeln("<BR>");
document.writeln("navigator.plugins = " + navigator.plugins);
document.writeln("<BR>");
document.writeln("navigator.userAgent = " + navigator.userAgent);
document.writeln("<BR>");
</SCRIPT>
</BODY>
</HTML>
```

10.2.6. Objeto screen

Este objeto contiene una serie de propiedades que describen las características físicas de la pantalla: resolución y colores.

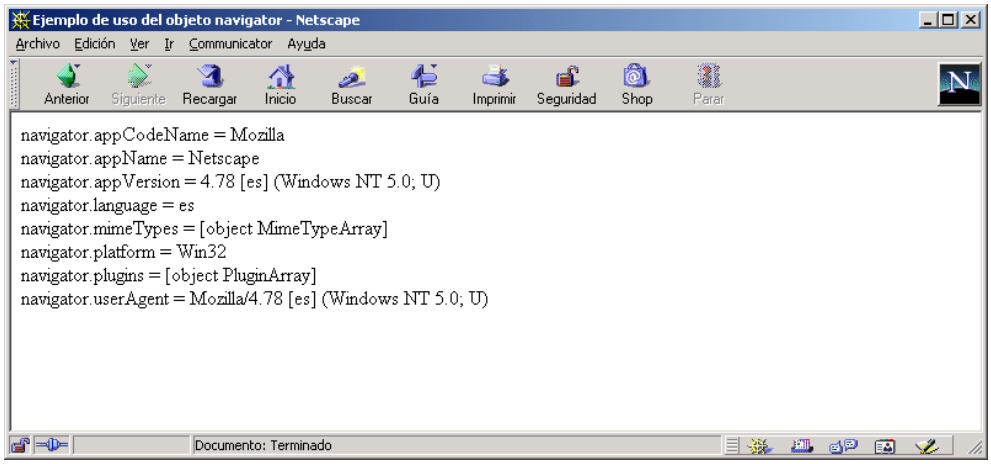


Figura 10.8: Propiedades del objeto navigator en Netscape Communicator

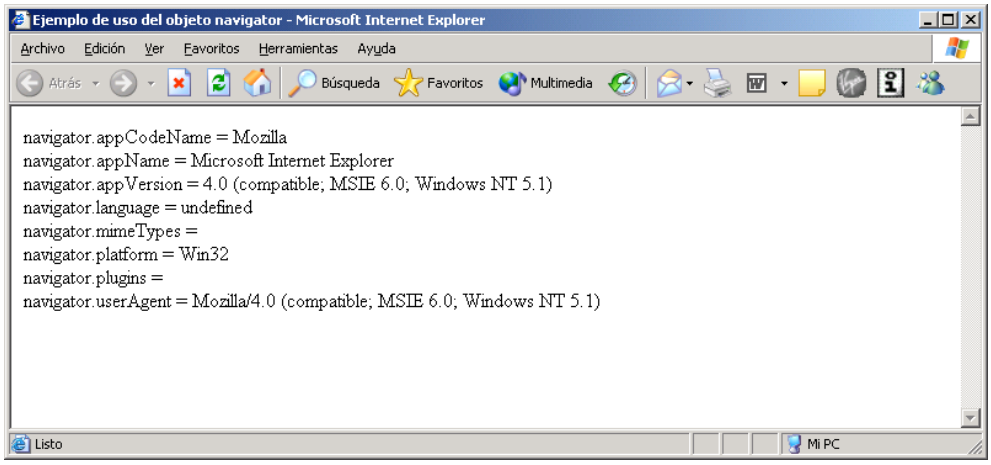


Figura 10.9: Propiedades del objeto navigator en Microsoft Internet Explorer

Propiedades

availHeight Altura de la pantalla en pixels, menos el espacio empleado por el sistema operativo para mostrar elementos permanentes o semipermanentes, como la barra de tareas en Microsoft Windows.

availLeft Coordenada x del primer pixel que no está reservado por el sistema operativo para mostrar elementos permanentes o semipermanentes.

availTop Coordenada y del primer pixel que no está reservado por el sistema operativo para mostrar elementos permanentes o semipermanentes.

availWidth Anchura de la pantalla en pixels, menos el espacio empleado por el sistema operativo para mostrar elementos permanentes o semipermanentes.

colorDepth La profundidad en bits de la paleta de color. Si no está definido toma el mismo valor que `screen.pixelDepth`.

height Altura de la pantalla.

pixelDepth Número de bits por pixel (profundidad de color) de la pantalla.

width Anchura de la pantalla.

El siguiente ejemplo muestra todas las propiedades del objeto `screen`. En la Figura 10.10 se puede ver como se muestra esta página en un navegador.

Ejemplo 10.12

```
<HTML>
<HEAD>
<TITLE>Ejemplo de uso del objeto screen</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
  document.write("availHeight: " + screen.availHeight + "<BR>");
  document.write("availLeft: " + screen.availLeft + "<BR>");
  document.write("availTop: " + screen.availTop + "<BR>");
  document.write("availWidth: " + screen.availWidth + "<BR>");
  document.write("colorDepth: " + screen.colorDepth + "<BR>");
  document.write("height: " + screen.height + "<BR>");
  document.write("pixelDepth: " + screen.pixelDepth + "<BR>");
  document.write("width: " + screen.width + "<BR>");
</SCRIPT>
```

```
16 </BODY>
17 </HTML>
```

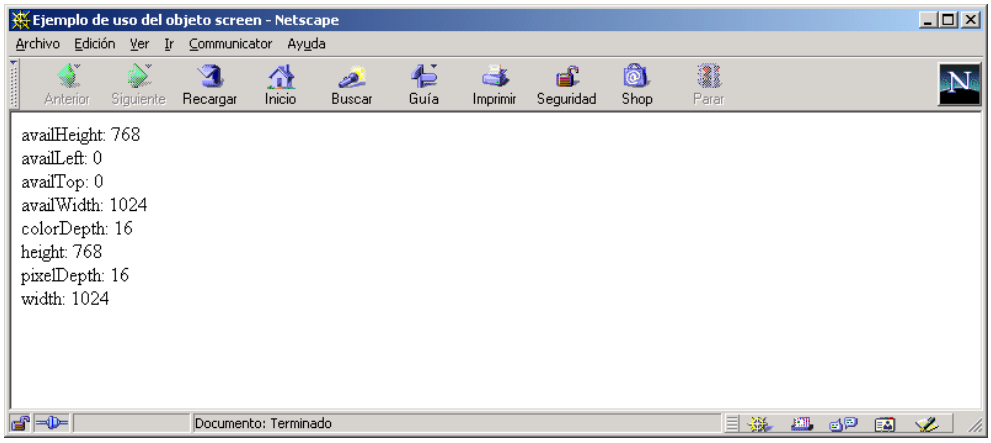


Figura 10.10: Propiedades del objeto screen en Netscape Communicator

10.2.7. Objeto window

Representa una ventana o, cuando la ventana está dividida en marcos, un marco concreto del navegador. Es el objeto en la posición superior en la jerarquía de objetos, por encima de todos los demás. Se crea un objeto `window` por cada etiqueta `<BODY> ... </BODY>` o `<FRAMESET> ... <FRAMESET>`. También se crea un objeto `window` para representar cada marco definido con una etiqueta `<FRAME>`. Además, también se pueden crear otras ventanas nuevas llamando al método `window.open`.

Propiedades

closed Especifica si una ventana ha sido cerrada.

crypto Permite acceder a las características de encriptación de Netscape Navigator.

defaultStatus Mensaje mostrado por defecto en la barra de estado.

document Contiene información sobre el documento actual y métodos que permiten mostrar contenido **HTML** al usuario.

frames Array que permite acceder a los marcos que posee una ventana.

history Posee información sobre las URLs que el cliente ha visitado dentro de la ventana.

innerHeight Dimensión vertical, en pixels, del área de contenido de la ventana.

innerWidth Dimensión horizontal, en pixels, del área de contenido de la ventana.

length Número de marcos (*frames*) de la ventana.

location Información sobre la **URL** actual.

locationbar Representa la barra de localización de la ventana del navegador.

menubar Representa la barra de menús de la ventana del navegador.

name Nombre único usado para identificar a la ventana.

offscreenBuffering Especifica si las actualizaciones de la ventana se realizan en un buffer.

opener Nombre de la ventana que ha abierto esta ventana usando el método `open`.

outerHeight Dimensión vertical, en pixels, del límite exterior de la ventana.

outerWidth Dimensión horizontal, en pixels, del límite exterior de la ventana.

pageXOffset Posición actual sobre el eje X, en pixels, de una página vista en la ventana.

pageYOffset Posición actual sobre el eje Y, en pixels, de una página vista en la ventana.

parent Ventana o marco que contiene al marco actual.

personalbar Representa la barra personal (también llamada barra de directorios) de la ventana del navegador.

screenX Posición sobre el eje X del extremo izquierdo de la ventana.

screenY Posición sobre el eje Y del extremo superior de la ventana.

scrollbars Representa las barras de desplazamiento de la ventana.

self Sinónimo de la ventana actual.

status Especifica un mensaje en la barra de estado de la ventana.

statusbar Representa la barra de estado de la ventana.

toolbar Representa la barra de herramientas de la ventana del navegador.

top Sinónimo de la ventana más superior en la jerarquía de ventanas.

window Sinónimo de la ventana actual.

Métodos

alert Muestra un cuadro de diálogo de alerta con un mensaje y un botón **OK**.

atob Decodifica una cadena de información que ha sido codificada usando la codificación *base-64*.

back Deshace la última navegación en la ventana de nivel superior.

blur Elimina el foco del objeto especificado.

btoa Crea una cadena codificada en *base-64*.

captureEvents Configura la ventana o documento para que capture todos los eventos del tipo especificado.

clearInterval Cancela un temporizador (*timeout*) que se había fijado con el método `setInterval`.

clearTimeout Cancela un temporizador (*timeout*) que se había fijado con el método `setTimeout`.

close Cierra la ventana.

confirm Muestra un cuadro de diálogo de confirmación con un mensaje y los botones **OK** y **Cancel**.

crypto.random Devuelve una cadena pseudoaleatoria cuya longitud es el número de bytes especificados.

crypto.signText Devuelve una cadena de información codificada que representa un objeto firmado.

disableExternalCapture Desactiva la captura de eventos externos fijado por el método `enableExternalCapture`.

enableExternalCapture Permite que una ventana con marcos capture los eventos en las páginas cargadas desde distintos sitios (servidores).

find Busca la cadena de texto especificada en el contenido de la ventana especificada.

focus Otorga el foco al objeto especificado.

forward Carga la siguiente **URL** en la lista del historial.

handleEvent Invoca el manejador del método especificado.

home Navega a la **URL** especificada en las preferencias del navegador como la página inicial (*home*).

moveBy Mueve la ventana según el desplazamiento indicado.

moveTo Mueve la esquina superior izquierda de la ventana a la posición de la pantalla indicada.

open Abre una nueva ventana del navegador.

print Imprime el contenido de la ventana o marco.

prompt Muestra un cuadro de diálogo de petición de datos con un mensaje y un campo de entrada.

releaseEvents Configura la ventana para que libere todos los eventos capturados del tipo indicado, enviando el evento a los siguientes objetos en la jerarquía de eventos.

resizeBy Redimensiona la ventana moviendo la esquina inferior derecha la cantidad indicada.

resizeTo Redimensiona la ventana según la altura y anchura indicada.

routeEvent Pasa un evento capturado a través de la jerarquía de eventos normal.

scroll Realiza un *scroll* a las coordenadas indicadas.

scrollBy Realiza un *scroll* en el área de visión según la cantidad especificada.

scrollTo Realiza un *scroll* en el área de visión a las coordenadas especificadas.

setHotKeys Activa o desactiva las *hot keys* en una ventana que no posee menú.

setInterval Evalúa una expresión o llama a una función cada vez que transcurre el número de milisegundos indicado.

setResizable Especifica si el usuario puede redimensionar una ventana.

setTimeout Evalúa una expresión o llama a una función una vez que ha transcurrido el número de milisegundos indicado.

setZOptions Especifica el orden sobre el eje Z.

stop Detiene la carga actual de la página.

Eventos

onBlur Se produce cuando un elemento pierde el foco.

onDragDrop Se produce cuando el usuario deja un objeto sobre la ventana del navegador.

onError Se produce cuando la carga de un documento o una imagen produce un error.

onFocus Se produce cuando un elemento recibe el foco.

onLoad Se produce cuando el navegador termina de cargar una ventana.

onMove Se produce cuando se mueve una ventana o marco.

onResize Se produce cuando se redimensiona una ventana o marco.

onUnload Se produce cuando un usuario cierra un documento.

El siguiente ejemplo muestra como se puede establecer una comunicación entre varias ventanas. A partir de una ventana, se puede abrir una ventana nueva en la que se puede escribir texto introducido en la primera ventana. En la Figura 10.11 se muestra la ventana principal y la ventana que se crea mediante el método `open`.

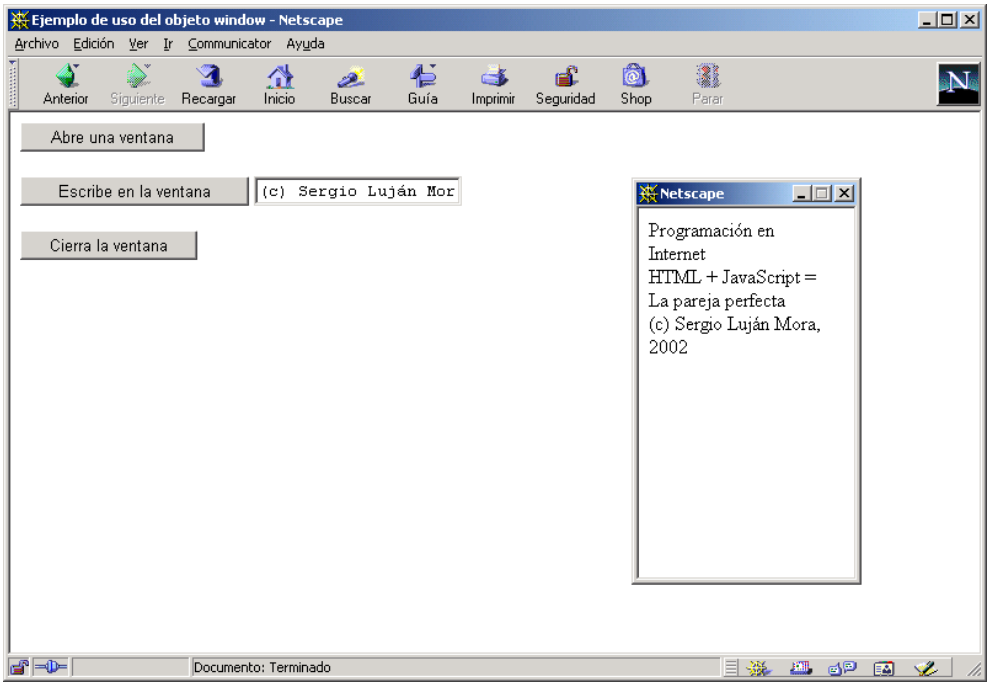


Figura 10.11: Interacción entre varias ventanas a través del objeto window

Ejemplo 10.13

```
1 <HTML>
2 <HEAD>
```

```
<TITLE>Ejemplo de uso del objeto window</TITLE>
<SCRIPT LANGUAGE="JavaScript">
var win = null;

function abre() {
  win = window.open("", "Ventana",
                    "scrollbars=yes,width=175,height=300");
}

function escri() {
  if(win != null)
  {
    win.document.write(document.formulario.algo.value + "<BR>\n");
    win.focus();
  }
}

function cierra() {
  if(win != null)
  {
    win.close();
    win = null;
  }
}
</SCRIPT>
</HEAD>
<BODY>
<FORM NAME="formulario">
<P>
<INPUT TYPE="BUTTON" VALUE="Abre una ventana" ONCLICK="abre()">
<P>
<INPUT TYPE="BUTTON" VALUE="Escribe en la ventana" ONCLICK="escri()">
<INPUT TYPE="TEXT" NAME="algo">
<P>
<INPUT TYPE="BUTTON" VALUE="Cierra la ventana" ONCLICK="cierra()">
</FORM>
</BODY>
</HTML>
```

El siguiente ejemplo muestra como se puede emplear el método `setTimeout` del objeto `window` para refrescar automáticamente una página cada 5 segundos. En la Figura 10.12 se muestra esta página en un navegador; para comprobar que la página se refresca cada 5 segundos, se muestra la hora actual (expresada como el número de milisegundos desde el 1 de enero de 1970 a las 0 horas).

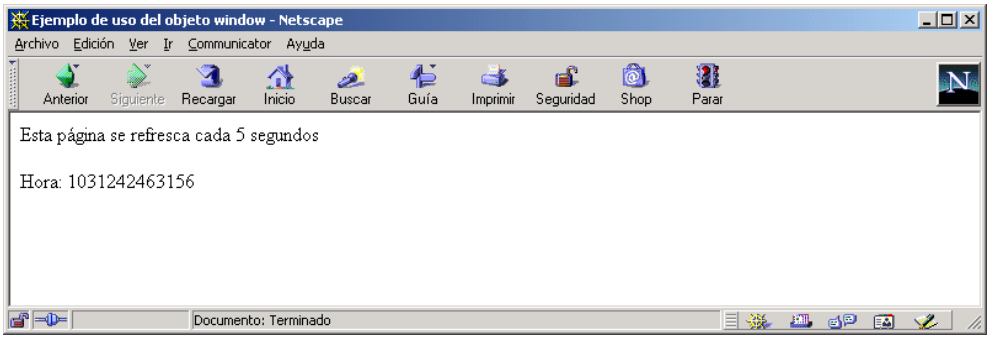


Figura 10.12: Refresco automático de una página mediante JavaScript

Ejemplo 10.14

```
1  <HTML>
2  <HEAD>
3  <TITLE>Ejemplo de uso del objeto window</TITLE>
4  <SCRIPT LANGUAGE="JavaScript">
5  function refrescar()
6  {
7    location.href = self.location.href;
8  }
9  </SCRIPT>
10 </HEAD>
11 <BODY ONLOAD="window.setTimeout('refrescar()', 5000)">
12 Esta página se refresca cada 5 segundos
13 <BR><BR>
14 <SCRIPT LANGUAGE="JavaScript">
15 var d;
16
17 d = new Date();
18 document.write("Hora: " + d.getTime());
19 </SCRIPT>
20 </BODY>
21 </HTML>
```

10.3. Modelo de objetos en Microsoft Internet Explorer

Incluimos en la Figura 10.13 una representación gráfica del modelo de objetos implementado en el navegador de MICROSOFT. Si se compara con el modelo de NETSCAPE (Figura 10.1), se pueden apreciar varias diferencias.

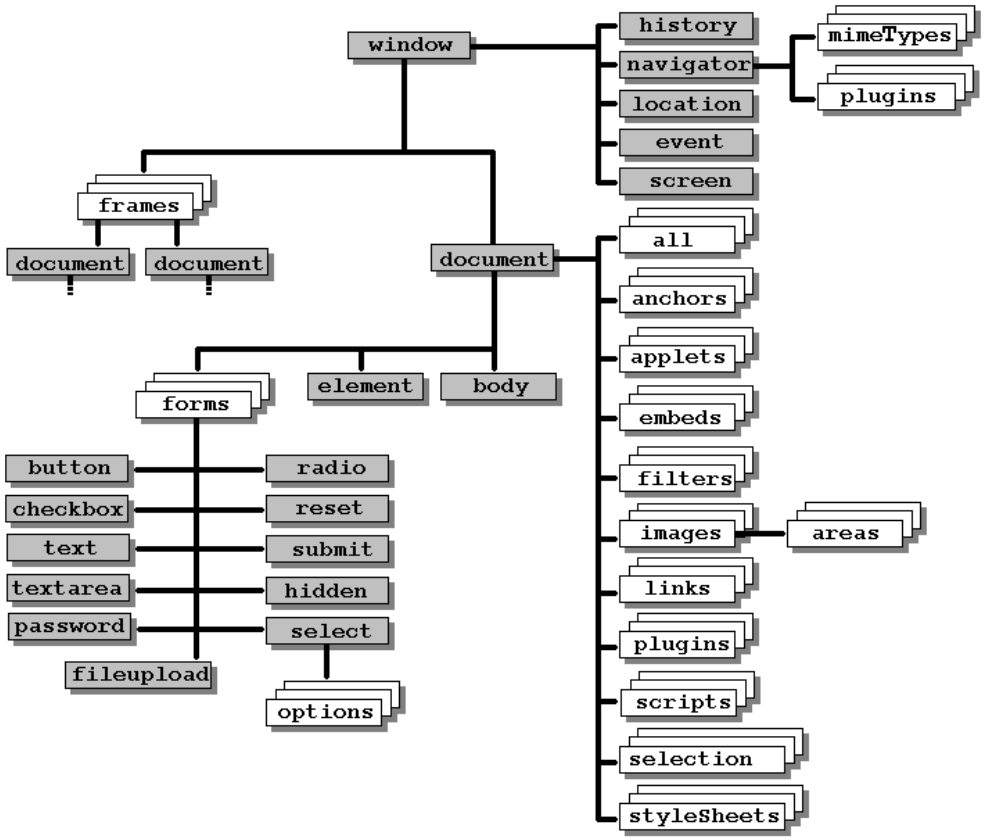


Figura 10.13: Modelo de objetos en Microsoft Internet Explorer

Apéndice A

Resumen de etiquetas de HTML

En este apéndice se incluye un resumen de la sintaxis de las etiquetas HTML. El objetivo de este apéndice es que sirva como una guía rápida de búsqueda en caso de duda.

Índice General

A.1. Introducción

Este resumen contiene todas las etiquetas **HTML** aceptadas por Netscape Navigator 4.0 y posteriores. Este resumen sólo contiene la sintaxis de las etiquetas. Conforme evoluciona **HTML** aparecen nuevas etiquetas, se añaden atributos nuevos a algunas etiquetas y otras quedan obsoletas (*deprecated*) y se desaconseja su uso.

Cuando una etiqueta se emplea por parejas (inicio y fin), se representa con unos puntos suspensivos, como `<HTML> ... </HTML>`, mientras que cuando la etiqueta es individual aparece como `<IMG>`. Cuando un atributo de una etiqueta puede tomar una serie de posibles valores, estos se han separado con una barra vertical, como por ejemplo `ALIGN="LEFT" | "RIGHT" | "CENTER"`. Otros atributos no reciben ningún valor, como el atributo `MULTIPLE` de la etiqueta `<SELECT> ... </SELECT>`.

Las etiquetas se han clasificado en los siguientes grupos:

- Etiquetas que definen la estructura del documento.
- Etiquetas que pueden ir en la cabecera `<HEAD>`.
- Etiquetas que definen bloques de texto.
- Etiquetas de listas.
- Etiquetas de características del texto.
- Etiquetas de anclas y enlaces.
- Etiquetas de imágenes y mapas de imágenes.
- Etiquetas de tablas.
- Etiquetas de formularios.
- Etiquetas de marcos.
- Etiquetas de situación de contenidos.
- Etiquetas de script.
- Etiquetas de *applets* y *plug-ins*.
- Etiquetas de ajuste del texto.
- Atributos universales.

Para incluir un comentario en una página **HTML** se utiliza la etiqueta `<!-- Comentario -->`. Se puede emplear para anular una sección de una página, de forma que no se visualice en el navegador, pero sin tener que eliminar dicha sección de la página.

A.2. Etiquetas que definen la estructura del documento

En este grupo se incluyen las etiquetas que definen la estructura básica de un documento **HTML**.

- Etiqueta más externa: `<HTML> ... </HTML>`.
- Encabezado del documento: `<HEAD> ... </HEAD>`.
- Contenido principal del documento (cuerpo): `<BODY> ... </BODY>`. Atributos:
  - `BACKGROUND="URL"`.
  - `BGCOLOR="color"`.
  - `TEXT="color"`.
  - `LINK="color"`.
  - `ALINK="color"`.
  - `VLINK="color"`.
  - `ONLOAD="códigoScript"`.
  - `ONUNLOAD="códigoScript"`.
  - `ONBLUR="códigoScript"`.
  - `ONFOCUS="códigoScript"`.

A.3. Etiquetas que pueden ir en la cabecera

En este grupo se clasifican las etiquetas que pueden usarse en la cabecera de un documento, entre las etiquetas `<HEAD>` y `</HEAD>`.

- Título del documento: `<TITLE> ... </TITLE>`.
- Dirección URL base: `<BASE>`. Atributos:
  - `HREF="URL"`.
  - `TARGET="nombreVentana"`.
- Información metadocumental: `<META>`. Atributos:
  - `NAME="nombre"`.
  - `HTTP-EQUIV="nombreCampo"`.
  - `CONTENT="valor"`.

- Definición de estilo: `<STYLE> ... </STYLE>`. Atributo:
  - `TYPE="tipoEstilo"`.
- Enlace a ficheros externos: `<LINK>`. Atributos:
  - `REL="tipoFichero"`.
  - `TYPE="tipo"`.
  - `SRC="URL"`.
- Elemento de búsqueda[1]: `<ISINDEX>`. Atributo:
  - `PROMPT="texto"`.
- Código *JavaScript* en el cliente[2]: `<SCRIPT> ... </SCRIPT>`. Atributos:
  - `LANGUAGE="nombreLenguajeScript"`.
  - `SRC="URL"`.

A.4. Etiquetas que definen bloques de texto

En esta sección se incluyen las etiquetas que definen bloques de texto, como encabezados, párrafos, citas, etc.

- Formato dirección: `<ADDRESS> ... </ADDRESS>`.
- Bloque de texto con sangría: `<BLOCKQUOTE> ... </BLOCKQUOTE>`.
- Sección de un documento: `<DIV> ... </DIV>`. Atributo:
  - `ALIGN="LEFT" | "CENTER" | "RIGHT" | "JUSTIFY"`[3].
- Encabezamientos predefinidos: `<H1> ... </H1>`, `<H2> ... </H2>`, `<H3> ... </H3>`, `<H4> ... </H4>`, `<H5> ... </H5>` y `<H6> ... </H6>`. Atributo:
  - `ALIGN="LEFT" | "CENTER" | "RIGHT" | "JUSTIFY"`.
- Párrafo: `<P> ... </P>`. Atributo:
  - `ALIGN="LEFT" | "CENTER" | "RIGHT" | "JUSTIFY"`.
- Texto preformateado (tipo de letra fija): `<PRE> ... </PRE>`. Atributos:
  - `COLS="columnas"`.
  - `WRAP`.
- Secuencia literal de caracteres (desactiva intérprete): `<XMP> ... </XMP>`.

[1] También puede emplearse en el cuerpo.
[2] También puede emplearse en el cuerpo.
[3] En la documentación de Netscape Communicator no figura el valor `JUSTIFY`.

A.5. Etiquetas de listas

En este grupo se encuentran clasificadas las distintas etiquetas que permiten crear listas.

- Lista de directorio: `<DIR> ... </DIR>`.
- Lista de definición: `<DL> ... </DL>`. Atributo:
  - `COMPACT`.
- Término de definición: `<DT> ... </DT>`.
- Descripción de definición: `<DD> ... </DD>`.
- Lista de elementos individuales: `<MENU> ... </MENU>`.
- Lista ordenada: `<OL> ... </OL>`. Atributos:
  - `START="valor"`.
  - `TYPE="A" | "a" | "I" | "i" | "1"`.
- Lista no ordenada: `<UL> ... </UL>`. Atributo:
  - `TYPE="CIRCLE" | "DISC" | "SQUARE"`.
- Elemento de una lista: `<LI> ... </LI>`. Atributos:
  - `TYPE="CIRCLE" | "DISC" | "SQUARE" | "A" | "a" | "I" | "i" | "1"`.
  - `VALUE="número"`.

A.6. Etiquetas de características del texto

En esta sección se muestran las etiquetas que definen las características del texto a nivel de carácter.

- Negrita: `<B> ... </B>`.
- Tamaño del tipo de letra por defecto: `<BASEFONT> ... </BASEFONT>`. Atributo:
  - `SIZE="número"`.
- Tamaño de letra mayor: `<BIG> ... </BIG>`.
- Parpadeante: `<BLINK> ... </BLINK>`.

- Cita: `<CITE> ... </CITE>`.
- Código: `<CODE> ... </CODE>`.
- Enfatizado: `<EM> ... </EM>`.
- Características del tipo de letra: `<FONT> ... </FONT>`. Atributos:
  - `COLOR="color"`.
  - `FACE="listaTiposDeLetra"`.
  - `POINT-SIZE="tamañoPunto"`.
  - `SIZE="número"`.
  - `WEIGHT="gradoNegrita"`.
- Cursiva: `<I> ... </I>`.
- Texto de teclado: `<KBD> ... </KBD>`.
- Muestra el resto del documento tal cual: `<PLAINTEXT>`.
- Tipo de letra menor: `<SMALL> ... </SMALL>`.
- Tachado: `<STRIKE> ... </STRIKE>` o `<S> ... </S>`.
- Énfasis fuerte: `<STRONG> ... </STRONG>`.
- Subíndice: `<SUB> ... </SUB>`.
- Superíndice: `<SUP> ... </SUP>`.
- Mecanografiado: `<TT> ... </TT>`.
- Subrayado: `<U> ... </U>`.
- Variable: `<VAR> ... </VAR>`.

A.7. Etiquetas de anclas y enlaces

En esta sección se muestra la etiqueta `<A> ... </A>`, que se usa tanto para definir las anclas (lugares a donde se puede crear un enlace) como los enlaces.

- Ancla: `<A NAME> ... </A>`. Atributo:
  - `NAME="nombreAncla"`.
- Enlace: `<A HREF> ... </A>`. Atributos:

- HREF="URL".
- ONCLICK="códigoScript".
- ONMOUSEOUT="códigoScript".
- ONMOUSEOVER="códigoScript".
- TARGET="_blank" | "_parent" | "_self" | "_top" | "nombre-Ventana".

A.8. Etiquetas de imágenes y mapas de imágenes

En esta sección se muestran las etiquetas que permiten insertar una imagen en un documento **HTML** y crear mapas de imágenes (también llamados mapas o imágenes sensibles).

- Imagen: <IMG>. Atributos:
  - SRC="URL".
  - LOWSRC="URL".
  - ALT="textoAlternativo".
  - ALIGN="LEFT" | "RIGHT" | "TOP" | "ABSMIDDLE" | "ABSBOTTOM" | "TEXTTOP" | "MIDDLE" | "BASELINE" | "BOTTOM".
  - BORDER="anchuraBorde".
  - HEIGHT="altura".
  - WIDTH="anchura".
  - HSPACE="margenHorizontal".
  - VSPACE="margenVertical".
  - ISMAP.
  - USEMAP="#nombreMapa".
  - NAME="nombreImagen".
  - ONABORT="códigoScript".
  - ONERROR="códigoScript".
  - ONLOAD="códigoScript".
  - SUPPRESS="TRUE" | "FALSE".
- Área de un mapa de imagen: <AREA>. Atributos:
  - COORDS="coordenadas".

- SHAPE="CIRCLE" | "RECT" | "POLY".
- HREF="URL".
- NOHREF.
- TARGET="nombreVentana".
- ONMOUSEOUT="códigoScript".
- ONMOUSEOVER="códigoScript".
- NAME="nombreArea".

- Mapa de imagen: <MAP> ... </MAP>. Atributo:
  - NAME="nombreMapa".

A.9. Etiquetas de tablas

Esta sección muestra las etiquetas que se emplean para crear tablas.

- Tabla: <TABLE> ... </TABLE>. Atributos:
  - ALIGN="LEFT" | "CENTER" | "RIGHT".
  - BGCOLOR="color".
  - BORDER="anchuraBorde".
  - CELLPADDING="valor".
  - CELLSPACING="valor".
  - HEIGHT="altura".
  - WIDTH="anchura".
  - COLS="numeroDeColumnas".
  - HSPACE="margenHorizontal".
  - VSPACE="margenVertical".
- Título de la tabla: <CAPTION > ... </CAPTION>. Atributo:
  - ALIGN="BOTTOM" | "TOP".
- Fila de la tabla: <TR> ... </TR>. Atributos:
  - ALIGN="LEFT" | "CENTER" | "RIGHT".
  - VALIGN="BASELINE" | "BOTTOM" | "MIDDLE" | "TOP".
  - BGCOLOR="color".
- Celda de una tabla: <TD> ... </TD>. Atributos:

- ALIGN="LEFT" | "CENTER" | "RIGHT".
- VALIGN="BASELINE" | "BOTTOM" | "MIDDLE" | "TOP".
- BGCOLOR="color".
- COLSPAN="valor".
- ROWSPAN="valor".
- HEIGHT="altura".
- WIDTH="anchura".
- NOWRAP.

- Encabezamiento de una columna o fila: <TH> ... </TH>. Atributos:
  - ALIGN="LEFT" | "CENTER" | "RIGHT".
  - VALIGN="BASELINE" | "BOTTOM" | "MIDDLE" | "TOP".
  - BGCOLOR="color".
  - COLSPAN="valor".
  - ROWSPAN="valor".
  - HEIGHT="altura".
  - WIDTH="anchura".
  - NOWRAP.

A.10. Etiquetas de formularios

En este grupo se encuentran las etiquetas que definen los formularios y sus posibles controles.

- Formulario: <FORM> ... </FORM>. Atributos:
  - NAME="nombreFormulario".
  - ACTION="URL".
  - ENCTYPE="tipoCodificación".
  - METHOD="GET" | "POST".
  - TARGET="nombreVentana".
  - ONRESET="códigoScript".
  - ONSUBMIT="códigoScript".
- Botón: <INPUT TYPE="BUTTON">. Atributos:
  - NAME="nombreBotón".

- `VALUE="etiqueta"`.
- `ONCLICK="códigoScript"`.

- Casilla de verificación: `<INPUT TYPE="CHECKBOX">`. Atributos:
  - `NAME="nombre"`.
  - `VALUE="valor"`.
  - `CHECKED`.
  - `ONCLICK="códigoScript"`.
- Envío de fichero: `<INPUT TYPE="FILE">`. Atributos:
  - `NAME="nombre"`.
  - `VALUE="nombreFichero"`.
- Elemento oculto: `<INPUT TYPE="HIDDEN">`. Atributos:
  - `NAME="nombre"`.
  - `VALUE="valor"`.
- Imagen como botón: `<INPUT TYPE="IMAGE">`. Atributos:
  - `NAME="nombre"`.
  - `ALIGN="LEFT" | "RIGHT" | "TOP" | "ABSMIDDLE" | "ABSBOTTOM" | "TEXTTOP" | "MIDDLE" | "BASELINE" | "BOTTOM"`.
  - `SRC="URL"`.
- Contraseña: `<INPUT TYPE="PASSWORD">`. Atributos:
  - `NAME="nombre"`.
  - `VALUE="texto"`.
  - `MAXLENGTH="máximoNúmeroCaracteres"`.
  - `SIZE="longitudCampo"`.
  - `ONSELECT="códigoScript"`.
- Botón de radio: `<INPUT TYPE="RADIO">`. Atributos:
  - `NAME="nombre"`.
  - `VALUE="valor"`.
  - `CHECKED`.
  - `ONCLICK="códigoScript"`.

- Restaurar (borrar): `<INPUT TYPE="RESET">`. Atributos:
  - `NAME="nombre"`.
  - `VALUE="etiqueta"`.
  - `ONCLICK="códigoScript"`.
- Botón de envío: `<INPUT TYPE="SUBMIT">`. Atributos:
  - `NAME="nombre"`.
  - `VALUE="etiqueta"`.
- Línea de texto: `<INPUT TYPE="TEXT">`. Atributos:
  - `NAME="nombre"`.
  - `VALUE="texto"`.
  - `MAXLENGTH="máximoNúmeroCaracteres"`.
  - `SIZE="longitudCampo"`.
  - `ONBLUR="códigoScript"`.
  - `ONCHANGE="códigoScript"`.
  - `ONFOCUS="códigoScript"`.
  - `ONSELECT="códigoScript"`.
- Lista de selección: `<SELECT> ... </SELECT>`. Atributos:
  - `NAME="nombre"`.
  - `MULTIPLE`.
  - `SIZE="longitudCampo"`.
  - `ONBLUR="códigoScript"`.
  - `ONCHANGE="códigoScript"`.
  - `ONCLICK="códigoScript"`.
  - `ONFOCUS="códigoScript"`.
- Opción en una lista de selección: `<OPTION> ... </OPTION>`. Atributos:
  - `VALUE="valor"`.
  - `SELECTED`.
- Área de texto: `<TEXTAREA> ... </TEXTAREA>`. Atributos:
  - `NAME="nombre"`.
  - `COLS="columnas"`.

  - ROWS="filas".
  - WRAP="OFF" | "HARD" | "SOFT".
  - ONBLUR="códigoScript".
  - ONCHANGE="códigoScript".
  - ONFOCUS="códigoScript".
  - ONSELECT="códigoScript".
- Generador de clave: <KEYGEN>. Atributos:
  - NAME="nombre".
  - CHALLENGE="desafío".
- Elemento de búsqueda[4]: <ISINDEX>. Atributo:
  - PROMPT="texto".

A.11. Etiquetas de marcos

En esta sección se muestran las etiquetas que permiten crear marcos y conjuntos de marcos. Un marco es una región de una ventana que actúa como una ventana ella misma.

- Región de una ventana (marco): <FRAME>. Atributos:
  - BORDERCOLOR="color".
  - FRAMEBORDER="YES" | "NO".
  - MARGINHEIGHT="alturaMargen".
  - MARGINWIDTH="anchuraMargen".
  - NAME="nombreMarco".
  - NORESIZE.
  - SCROLLING="YES" | "NO" | "AUTO".
  - SRC="URL".
- Conjunto de marcos: <FRAMESET> ... </FRAMESET>. Atributos:
  - COLS="listaAnchurasColumnas".
  - ROWS="listaAlturasFilas".
  - BORDER="anchura".

[4] También puede emplearse en la cabecera.

- BORDERCOLOR="color".
- FRAMEBORDER="YES" | "NO".
- ONBLUR="códigoScript".
- ONFOCUS="códigoScript".
- ONLOAD="códigoScript".
- ONUNLOAD="códigoScript".

- Texto alternativo a los marcos: <NOFRAMES> ... </NOFRAMES>.

A.12. Etiquetas de situación de contenidos

En esta sección se encuentran las etiquetas que permiten definir la posición de los distintos elementos en una página. Estas etiquetas se emplean en **DHTML**.

- Definición de una capa: <LAYER> ... </LAYER>. Atributos:

  - ID="nombreCapa".
  - LEFT="posición".
  - TOP="posición".
  - PAGEX="páginaX".
  - PAGEY="páginaY".
  - SRC="URL".
  - Z-INDEX="número".
  - ABOVE="nombreCapa".
  - BELOW="nombreCapa".
  - WIDTH="anchura".
  - HEIGHT="altura".
  - CLIP="número,número,número,número".
  - VISIBILITY="visibilidad".
  - BGCOLOR="color".
  - BACKGROUND="URL".
  - ONBLUR="códigoScript".
  - ONFOCUS="códigoScript".
  - ONLOAD="códigoScript".
  - ONMOUSEOVER="códigoScript".

  - ONMOUSEOUT="códigoScript".

- Capa posicionada de forma relativa: <ILAYER> ... </ILAYER>. Atributos:
  - ID="nombreCapa".
  - LEFT="posición".
  - TOP="posición".
  - PAGEX="páginaX".
  - PAGEY="páginaY".
  - SRC="URL".
  - Z-INDEX="número".
  - ABOVE="nombreCapa".
  - BELOW="nombreCapa".
  - WIDTH="anchura".
  - HEIGHT="altura".
  - CLIP="número,número,número,número".
  - VISIBILITY="visibilidad".
  - BGCOLOR="color".
  - BACKGROUND="URL".
  - ONBLUR="códigoScript".
  - ONFOCUS="códigoScript".
  - ONLOAD="códigoScript".
  - ONMOUSEOVER="códigoScript".
  - ONMOUSEOUT="códigoScript".
- Texto alternativo a las capas: <NOLAYER> ... </NOLAYER>.

A.13. Etiquetas de script

En esta sección se encuentran las etiquetas que permiten incluir código script en una página **HTML**.

- Código *JavaScript* en el cliente[5]: <SCRIPT> ... </SCRIPT>. Atributos:
  - LANGUAGE="nombreLenguajeScript".
  - SRC="localizacionURL".
- Texto alternativo al código *JavaScript*: <NOSCRIPT> ... </NOSCRIPT>.
- Código en el servidor: <SERVER> ... </SERVER>.

[5] También puede emplearse en la cabecera.

A.14. Etiquetas de applets y plug-ins

Esta sección muestra las etiquetas que se emplean para incluir *applets* y objetos que emplean *plug-ins*.

- *Applet Java*: `<APPLET> ... </APPLET>`. Atributos:
  - `NAME="nombre"`.
  - `CODE="nombreFicheroClass"`.
  - `CODEBASE="directorioFicheroClass"`.
  - `ARCHIVE="archivo"`.
  - `ALT="textoAlternativo"`.
  - `ALIGN="LEFT" | "RIGHT" | "TOP" | "ABSMIDDLE" | "ABSBOTTOM" | "TEXTTOP" | "MIDDLE" | "BASELINE" | "BOTTOM"`.
  - `HEIGHT="altura"`.
  - `WIDTH="anchura"`.
  - `HSPACE="margenHorizontal"`.
  - `VSPACE="margenVertical"`.
  - `MAYSCRIPT`.
- Parámetro para un *applet*: `<PARAM>`. Atributos:
  - `NAME="nombre"`.
  - `VALUE="valor"`.
- *Plug-in* incrustado: `<EMBED> ... </EMBED>`. Atributos:
  - `NAME="nombre"`.
  - `SRC="URL"`.
  - `TYPE="tipoMIME"`.
  - `PLUGINSPAGE="URLinstrucciones"`.
  - `PLUGINURL="URLplugin"`.
  - `ALIGN="LEFT" | "RIGHT" | "TOP" | "BOTTOM"`.
  - `BORDER="anchura"`.
  - `FRAMEBORDER="YES" | "NO"`.
  - `HEIGHT="altura"`.
  - `WIDTH="anchura"`.
  - `UNITS="unidades"`.

  - HIDDEN="TRUE" | "FALSE".
  - HSPACE="margenHorizontal".
  - VSPACE="margenVertical".
  - PALETTE="FOREGROUND" | "BACKGROUND".
- Texto alternativo para objetos incrustados: <NOEMBED> ... </NOEMBED>.
- Objeto incrustado: <OBJECT>. Atributos:
  - CLASSID="ficheroClase".
  - DATA="URL".
  - CODEBASE="directorioFicheroClase".
  - TYPE="tipoMIME".
  - ALIGN="LEFT" | "RIGHT" | "TOP" | "ABSMIDDLE" | "ABSBOTTOM" | "TEXTTOP" | "MIDDLE" | "BASELINE" | "BOTTOM".
  - HEIGHT="altura".
  - WIDTH="anchura".
  - ID="nombre".

A.15. Etiquetas de ajuste del texto

Las etiquetas de esta sección permiten ajustar la posición del texto.

- Salto de línea: <BR>. Atributo:
  - CLEAR="ALL" | "LEFT" | "RIGHT".
- Centrado: <CENTER> ... </CENTER>.
- Línea horizontal: <HR>. Atributos:
  - ALIGN="CENTER" | "LEFT" | "RIGHT".
  - NOSHADE.
  - SIZE="grosor.
  - WIDTH="anchura".
- Múltiples columnas: <MULTICOL> ... </MULTICOL>. Atributos:
  - COLS="númeroColumnas".
  - GUTTER="separaciónColumnas".

    - WIDTH="anchuraColumnas".

- No salto de línea: <NOBR> ... </NOBR>.
- Espacio extra: <SPACER>. Atributos:
    - TYPE="HORIZONTAL" | "VERTICAL" | "BLOCK".
    - ALIGN="LEFT" | "RIGHT" | "TOP" | "ABSMIDDLE" | "ABSBOTTOM" | "TEXTTOP" | "MIDDLE" | "BASELINE" | "BOTTOM".
    - HEIGHT="altura".
    - WIDTH="anchura".
    - SIZE="tamaño".
- Intervalo de contenido: <SPAN> ... </SPAN>.
- Posible salto de línea: <WBR>.

A.16. Atributos universales

Los siguientes atributos se pueden emplear con prácticamente todas las etiquetas situadas en el cuerpo de una página <BODY> ... </BODY>.

- Estilo de la clase: CLASS="claseEstilo".
- Idioma: LANG="ISO".
- Nombre de lugar o de estilo: ID="nombreLugarOEstilo".
- Estilo: STYLE="estilo".

Apéndice B

Colores en HTML

El lenguaje HTML posee varias etiquetas con atributos que indican un color. Por ejemplo, la etiqueta <BODY> tiene el atributo BGCOLOR que permite indicar el color de fondo de una página y la etiqueta <FONT> posee el atributo COLOR para cambiar el color del texto. En este apéndice se explica cómo trabajar con los colores en HTML. Además, también se incluyen una serie de consejos sobre el uso de los colores en las páginas web.

Índice General

B.1. Cómo trabajar con las componentes RGB

Las componentes **RGB** permiten expresar cualquier color mediante la combinación de los tres colores básicos (primarios) rojo, verde y azul. Cada componente expresa la intensidad del color básico en la combinación. Así, por ejemplo, el color negro es el resultado de combinar los tres colores básicos con una intensidad nula (0), mientras que el blanco es el resultado de combinar los tres colores con una intensidad máxima (255 cada componente). Este sistema de codificación de los colores permite 16 777 216 colores (256 x 256 x 256 combinaciones posibles).

Cuando se trabaja con los colores en **HTML**, las tres componentes se tienen que expresar en hexadecimal. Como este sistema de numeración es un poco "engorroso", a continuación mostramos cómo se pueden convertir las componentes **RGB** de decimal a hexadecimal mediante las herramientas que posee Microsoft Windows.

B.1.1. Obtener las componentes del color deseado en decimal

Mediante cualquier programa que permita seleccionar colores, podemos elegir el color que queremos emplear en una página **HTML**. Por ejemplo, en el programa Microsoft Paint[1], que viene con los sistemas operativos de MICROSOFT y que se puede encontrar a través de **Inicio** → **Programas** → **Accesorios**, si seleccionamos en el menú **Colores** la opción **Modificar colores...** aparece la paleta de colores de la Figura B.1. Si en esta ventana se pulsa el botón **Definir colores personalizados** », la ventana anterior se amplia y aparece la ventana de la Figura B.2.

En esta ventana se puede elegir de la paleta de colores un color y para el color elegido se puede seleccionar su brillo. En las casillas **Rojo**, **Verde** y **Azul** aparecen las correspondientes componentes en decimal (los valores de cada casilla varían desde 0 hasta 255). Una vez que se tienen las componentes del color deseado, el siguiente paso es convertirlas a hexadecimal.

B.1.2. Transformar las componentes de decimal a hexadecimal

Para pasar un número del sistema decimal al hexadecimal, se tiene que dividir el número entre 16. Si el cociente es mayor que 15, se divide el cociente entre 16 y así sucesivamente hasta lograr un cociente menor que 16. El número en hexadecimal escrito de izquierda a derecha se compone del último cociente y los sucesivos restos obtenidos, desde el último hasta el primero, pero si el último cociente o los restos tienen dos cifras, se tienen que transformar según las equivalencias del Cuadro B.1.

Cociente o resto	Cifra hexadecimal
10	A
11	B
12	C
13	D
14	E
15	F

Cuadro B.1: Equivalencias para pasar del sistema decimal al hexadecimal

[1]Se ha elegido este programa porque se encuentra en la mayoría de los ordenadores con sistema operativo Microsoft Windows. Existen programa de dibujo o retoque fotográfico, como Jasc Paint Shop Pro, que facilitan la tarea, ya que directamente muestran en hexadecimal las componentes **RGB** de un color.

Figura B.1: Ventana para modificar colores en Microsoft Paint

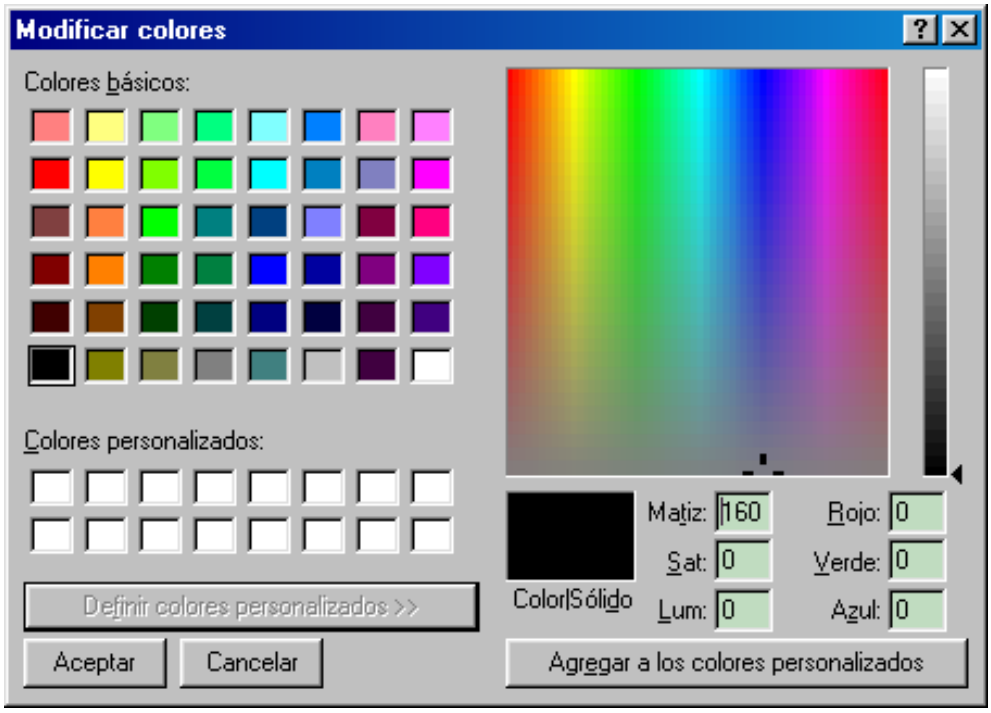


Figura B.2: Ventana para definir colores personalizados en Microsoft Paint

Por ejemplo, el número 241 en decimal equivale a F1 en hexadecimal, ya que el primer (y único) resto que se obtiene es 1 y el último (y único) cociente 15, que equivale a F.

Como el método anterior es tedioso y propenso a errores, se puede realizar la conversión automáticamente mediante el ordenador. Para ello, se puede emplear el programa Calculadora que se incluye en los sistemas operativos de MICROSOFT y que se encuentra otra vez en **Inicio** → **Programas** → **Accesorios**. El programa Calculadora posee dos modos de visualización: estándar y científica. Para realizar la conversión, tiene que estar en el modo calculadora científica, que se selecciona a través del menú **Ver**.

Tal como se ve en la Figura B.3, el programa Calculadora tiene cuatro botones de radio marcados con las etiquetas **Hex** (hexadecimal), **Dec** (decimal), **Oct** (octal) y **Bin** (binario), que permiten convertir un número a los distintos sistemas de numeración. Para pasar de decimal a hexadecimal, simplemente hay que escribir el número cuando la calculadora se encuentra en el sistema **Dec** y pulsar el botón **Hex** para que aparezca en pantalla el número convertido a hexadecimal.

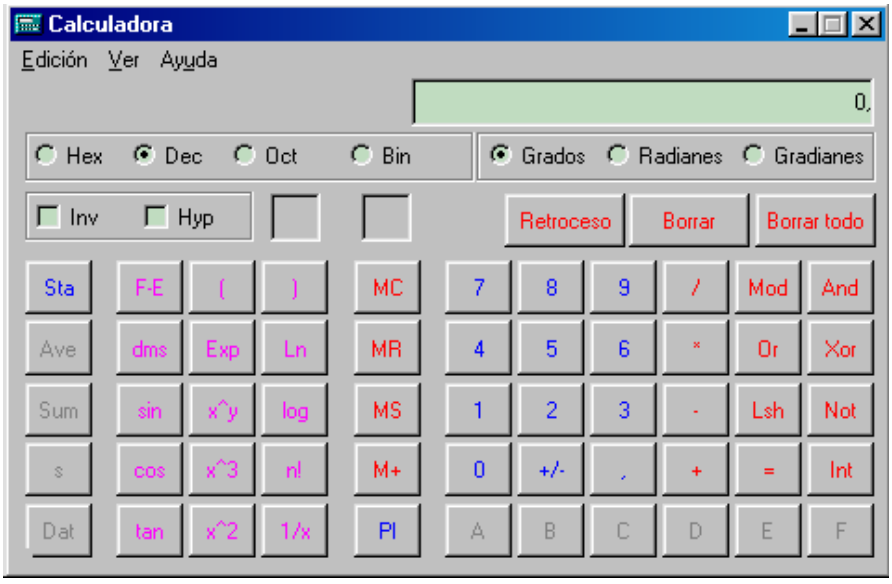


Figura B.3: Calculadora en modo científico

B.2. Tabla de colores

En **HTML** existe una serie de colores predefinidos a los que se les ha asignado un nombre en inglés. La lista de nombres contiene cientos de colores, por lo que aquí solo incluimos el Cuadro B.2 con algunos de ellos, junto con su valor en **RGB**.

Nombre color	Valor RGB
aqua	#00FFFF
black	#000000
blue	#0000FF
fuchsia	#FF00FF
gray	#808080
green	#008000
lime	#00FF00
maroon	#800000
navy	#000080
olive	#808000
purple	#800080
red	#FF0000
silver	#C0C0C0
teal	#008080
yellow	#FFFF00
white	#FFFFFF

Cuadro B.2: Nombres de algunos colores en HTML

B.3. Cambio de colores

Si se quiere cambiar un color, ya sea el color del fondo, el color del texto o el color de los enlaces, es importante establecer un color para todos ellos mediante los atributos `BGCOLOR`, `TEXT`, `LINK`, `VLINK` y `ALINK` de la etiqueta `<BODY> ... </BODY>`, ya que cualquier cambio parcial puede producir una combinación con poco contraste entre los colores que se modifican y los colores que se emplean por defecto (que cambian de un navegador a otro o incluso el usuario los puede modificar a su gusto). Por ejemplo, en la Figura B.4 se puede ver el cuadro de diálogo de Microsoft Internet Explorer 5.5 que permite cambiar los colores de los atributos anteriores y en la Figura B.5 se puede observar el cuadro de diálogo de Netscape Communicator 4.78 que realiza la misma función.

Incluso si se emplea una imagen como fondo de una página, hay que asignar un valor al atributo `BGCOLOR`: si el usuario ha desactivado la carga de imágenes, puede ser que el texto no se vea correctamente sobre el color de fondo por defecto de la página.

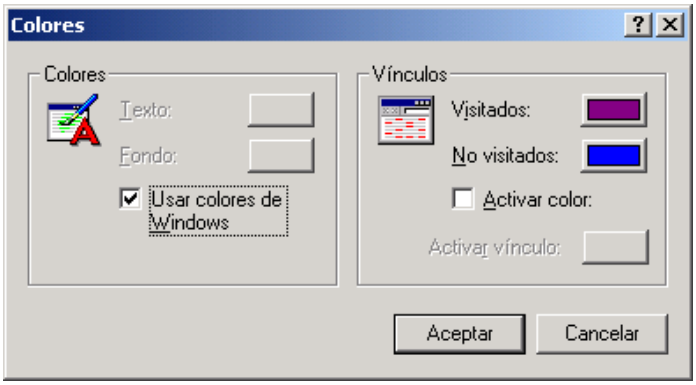


Figura B.4: Cuadro de diálogo para cambiar colores en Microsoft Internet Explorer

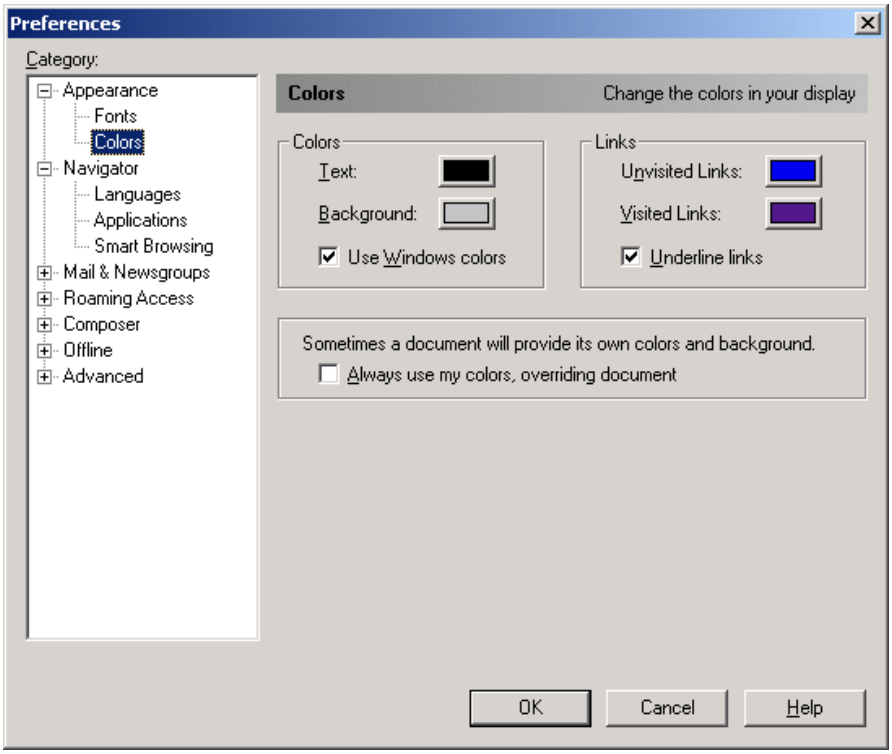


Figura B.5: Cuadro de diálogo para cambiar colores en Netscape Communicator

B.4. Consejos sobre el uso de colores

Los colores son una gran ayuda a la hora de comunicar un mensaje. Una página web en blanco y negro es aburrida y monótona, pero una página con colores divierte y mejora su lectura. Sin embargo, el mal uso de los colores puede destruir el mensaje que se quiere comunicar, ya que el usuario abandona la página web sin consultar su contenido.

Cuando se emplee un color, hay que pensar para qué se está empleando. Un color se puede usar de distintas formas:

- Para etiquetar: en ese caso, el color actúa como un nombre.
- Para medir: el color se puede emplear para representar medidas (escala de colores).
- Para representar o imitar la realidad: el color se puede emplear como una representación de la realidad (metáfora).
- Para decorar: el color se puede emplear para añadir belleza.

Sobre el mal uso de los colores, es adecuado recordar el comentario que hacía Edward R. Tufte en 1990, poco antes del nacimiento de la web (pero que se puede aplicar sin problema a las páginas web):

> Color's multidimensionality can also enliven and inform what users must face at computer terminals, although some color applied to display screens has made what should be a straightforward tool into something that looks like a grim parody of a video game.
>
> *La multidimensionalidad del color también puede alegrar e informar aquello que los usuarios tienen que afrontar delante de un ordenador, aunque algunos colores aplicados en las pantallas de ordenador han transformado aquello que debía ser una simple herramienta en algo que parece una grotesca parodia de un videojuego.*
>
> Edward R. Tufte en *Envisioning Information*, Graphics Press, 1990

Por otro lado, la capacidad humana de distinguir colores tiene un límite. Los expertos en color se cree que pueden distinguir aproximadamente un millón de colores, mientras que la mayoría de la gente no llega a los 20.000. Un límite adecuado para una página web es emplear 20 colores distintos como máximo (sin contar, claro está, los colores que se emplean en las imágenes o fotografías).

Algunos consejos básicos sobre el uso de los colores son los siguientes:

- Un color brillante sobre una gran superficie también brillante suele irritar (por ejemplo, un texto rosa sobre un fondo amarillo). Sin embargo, un color brillante usado sobre un fondo sin brillo (apagado) destaca y proporciona dinamismo.
- Los colores brillantes cercanos entre sí causan un mal efecto.
- Los colores de fondo deben de ser discretos y suaves, de modo que no distraigan la atención del usuario frente a los elementos situados en un primer plano.
- Para que el texto sea claramente legible, tiene que existir un gran contraste entre el color del texto y el color del fondo. La combinación de mayor contraste es el texto en negro sobre un fondo blanco, que es la que se emplea normalmente en los documentos impresos (libros, periódicos, manuales, etc.). Sin embargo, para evitar la fatiga visual que produce una pantalla completamente blanca, se puede emplear justo la combinación contraria: el texto en blanco sobre un fondo negro[2].

[2]Sin embargo, las combinaciones de colores claros para el texto sobre fondo oscuro se imprimen con dificultad.

Apéndice C

Depuración de errores de JavaScript

Como los lenguajes de script que se emplean en las páginas web son interpretados, la depuración de errores es difícil (al no existir la fase de compilación, no se detectan los errores sintácticos o semánticos). En este apéndice se explica como se puede depurar el código de script en cualquier navegador. Además, se comentan algunas herramientas específicas que poseen los navegadores Microsoft Internet Explorer y Netscape Communicator.

Índice General

C.1. Introducción

La depuración de errores de *JavaScript* suele estar desactivada en los navegadores, ya que sólo es útil para aquellos programadores que quieran verificar su código. Vamos a ver como podemos depurar código *JavaScript* de forma general en cualquier

navegador y de forma específica mediante las características que nos ofrecen Netscape Communicator y Microsoft Internet Explorer.

C.2. Depuración en cualquier navegador

Se pueden emplear las ventanas de alerta de *JavaScript* para mostrar la información que deseemos durante la ejecución del código: valor de una variable, situación del punto de ejecución del código, etc. Este método también permite detener momentáneamente la ejecución del código. Las ventajas que ofrece este sistema son:

1. La información que se muestra puede ser tan detallada como nosotros queramos.
2. Su uso no tiene efecto sobre el resto del código.
3. Se pueden emplear tantas ventanas de alerta como se quiera.
4. Se pueden colocar prácticamente en cualquier lugar del código.
5. Funciona con todos los navegadores que soporten código *JavaScript*.

Por ejemplo, si queremos comprobar en un punto del código el valor de una variable, sólo hay que incorporar una línea de código similar a la siguiente:

Ejemplo C.1

```
1 alert("El valor de la variable v es " + v);
```

C.3. Netscape Communicator

En versiones anteriores a la 4.06, cada vez que se producía un error de *JavaScript*, se mostraba una ventana de alerta con el error que se había producido. Este sistema era bastante engorroso, ya que si una página contenía múltiples errores, se mostraban múltiples ventanas que el usuario tenía que cerrar una a una.

A partir de la versión 4.06, que incluye *JavaScript* 1.3, se emplea la consola de *JavaScript* (Figura C.1). La consola sustituye a todas las ventanas de alerta: cada vez que se encuentra un error, se escribe un mensaje en la consola. La consola almacena todos los mensajes de error que se producen (Figura C.2). Además, la consola se puede dejar abierta o cerrar y abrir tantas veces como se quiera.

Por defecto, la consola no se muestra: se encuentra oculta y no hay ninguna opción en los menús del navegador que permita mostrarla. Si se desea mostrar la consola, existen cuatro posibilidades:

1. Escribir `javascript:` en la barra de direcciones del navegador y pulsar la tecla `Enter` (`Return`).

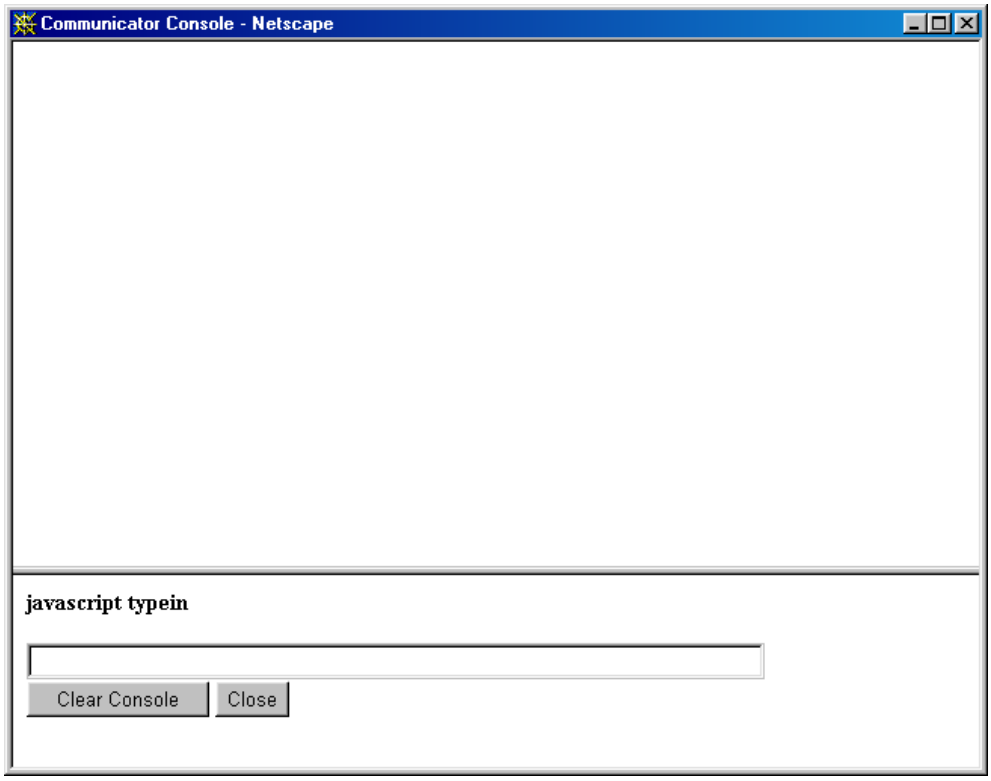


Figura C.1: Consola JavaScript de Netscape Communicator

2. Seleccionar en el menú **File** la opción **Open Page...** y escribir en el campo **URL** `javascript:`.

3. Incorporar en el código **HTML** de la página el siguiente enlace:

Ejemplo C.2

```
1 <A HREF="javascript:">Abrir consola JavaScript</A>.
```

4. Modificar las preferencias del navegador para que se muestre automáticamente cada vez que se produzca un error. Esta opción es la mejor para los usuarios que estén desarrollando código *JavaScript*.

La consola *JavaScript* dispone de dos botones, tal como se ve en la Figura C.1:

- El botón **Clear Console** permite borrar todo los mensajes de error mostrados hasta el momento.
- El botón **Close** cierra la ventana de la consola.

Para cada error que se encuentra, en la consola *JavaScript* se muestra un mensaje con la **URL** del fichero que contiene el error, el número de línea y un comentario que describe el tipo de error. En algunos casos, se incluye también una parte de la línea que contiene el error y se marca el punto exacto donde se encuentra.

C.3.1. Modificar las preferencias

Para algunos propósitos, el tener que abrir de forma manual la consola *JavaScript* puede ser molesto. Por ejemplo, si estamos desarrollando una página, el tener que teclear `javascript:` cada vez que hay un error para abrir la consola y ver el mensaje de error puede ser tedioso.

Se puede especificar que automáticamente se abra la consola cuando se produzca un error de *JavaScript*. Para ello, hay que modificar el fichero de preferencias `prefs.js`, que se encuentra en el directorio personal de cada usuario de Netscape Communicator. Por defecto, suele ser `C:\Archivos de programa\Netscape\Users\-nombreUsuario`.

Para modificar este fichero hay que seguir los siguientes pasos:

1. Asegurarse de que el navegador no está ejecutándose. El navegador puede sobrescribir los cambios que realicemos si se está ejecutando cuando editemos el fichero de preferencias.

2. Abrir el fichero de preferencias `prefs.js`. El contenido del fichero es similar a (sólo se muestran las primeras líneas):

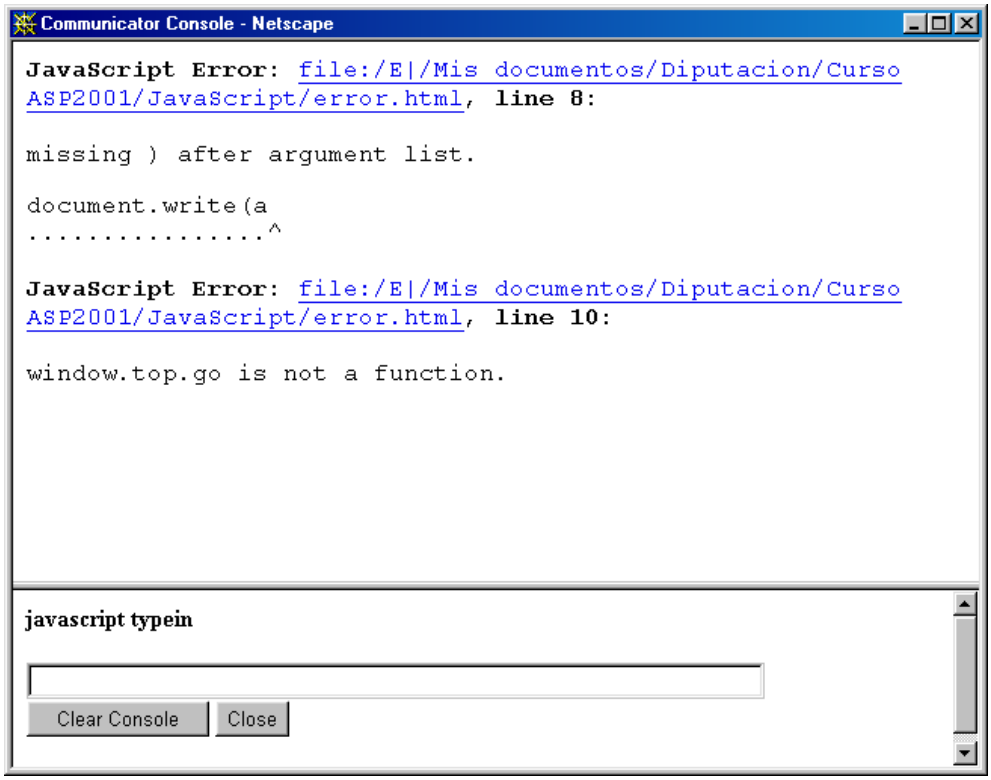


Figura C.2: Consola JavaScript con mensajes de error

Ejemplo C.3

```
// Netscape User Preferences
// This is a generated file!  Do not edit.

user_pref("autoupdate.enabled", false);
user_pref("browser.bookmark_window_showwindow", 3);
user_pref("browser.cache.disk_cache_size", 20480);
user_pref("browser.cache.memory_cache_size", 2048);
user_pref("browser.download_directory", "D:\\TV\\");
```

3. Añadir una de las siguientes líneas al final del fichero:

   - Si se quiere que se abra automáticamente la consola cada vez que se produce un error de *JavaScript*:

     Ejemplo C.4

     ```
     user_pref("javascript.console.open_on_error", true);
     ```

   - Si se quiere que se abra una ventana de alerta cada vez que se produce un error de *JavaScript* (como el sistema empleado en las versiones anteriores):

     Ejemplo C.5

     ```
     user_pref("javascript.classic.error_alerts", true);
     ```

     Nota: aunque este sistema figura en la documentación de Netscape Communicator, se ha probado en la versión 4.7 y no funciona.

4. Grabar y cerrar el fichero `prefs.js`.

C.3.2. Evaluación de expresiones con la consola

La consola de *JavaScript* es un ventana compuesta de dos marcos (Figura C.1). El marco inferior contiene un cuadro de texto etiquetado `javascript typein` en el que se pueden escribir expresiones de una sola línea. Se puede emplear este cuadro de texto para asignar valores a variables, realizar operaciones matemáticas o verificar el resultado devuelto por los operadores de comparación.

Para evaluar una expresión simplemente hay que escribirla en el cuadro de texto y pulsar la tecla `Enter` (`Return`). El resultado se muestra en el marco superior. Por ejemplo, al evaluar las siguiente expresiones se obtienen los resultados citados en los comentarios y mostrados en la Figura C.3:

Ejemplo C.6

```
alert("Hola a todos");   // Muestra una ventana de alerta
5-2;                     // Muestra 3
```

```
3 Math.sqrt(49);            // Muestra 7
4 var a=100; var b=45;      // Crea dos variables
5 a-b;                      // Muestra 55
```

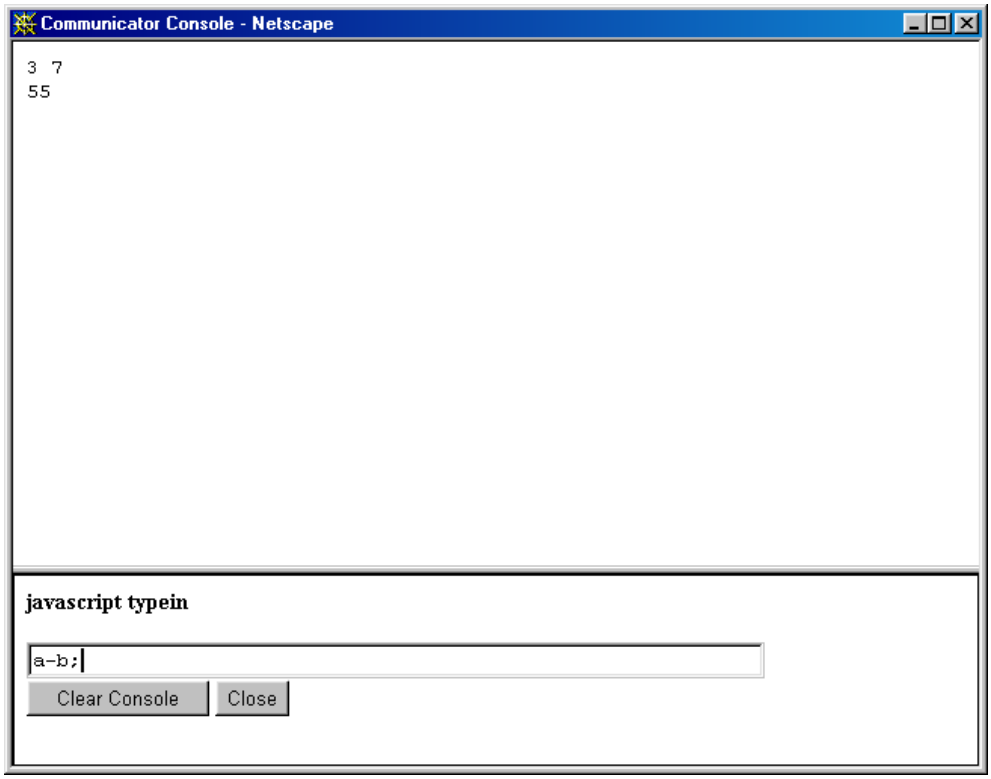


Figura C.3: Evaluación de expresiones

C.3.3. Netscape JavaScript Debugger

Existe una posibilidad más con el navegador de NETSCAPE: Netscape JavaScript Debugger (Figura C.4). Se trata de un *applet* (hecho en *Java*) que permite realizar una depuración avanzada del código *JavaScript*: visor de objetos (*object inspector*), puntos de parada (*breakpoints*), visores de variables (*watches*), ejecución paso a paso (*step running*), visor de la pila de ejecución (*call stack window*), etc.

Esta herramienta se puede descargar en las siguientes direcciones:

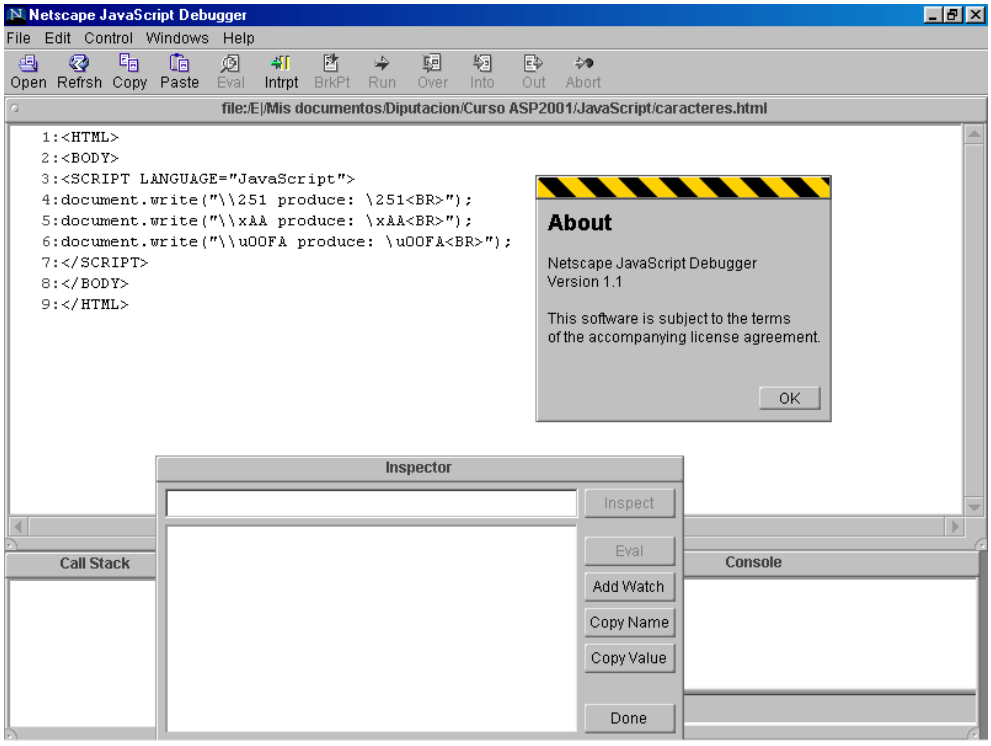


Figura C.4: Netscape JavaScript Debugger

- `http://developer.netscape.com/software/jsdebug.html`.

- `http://home.netscape.com/eng/Tools/JSDebugger/relnotes/relnotes-11.html`.

Para poderlo descargar e instalar, es necesario tener activada la opción **SmartUpdate**. Se accede a ella a través del menú **Edit** → **Preferences** → **Advanced** → **SmartUpdate** y se tiene que activar la casilla **Enable SmartUpdate** (Figura C.5). El sistema de descarga e instalación de este *applet* es automático.

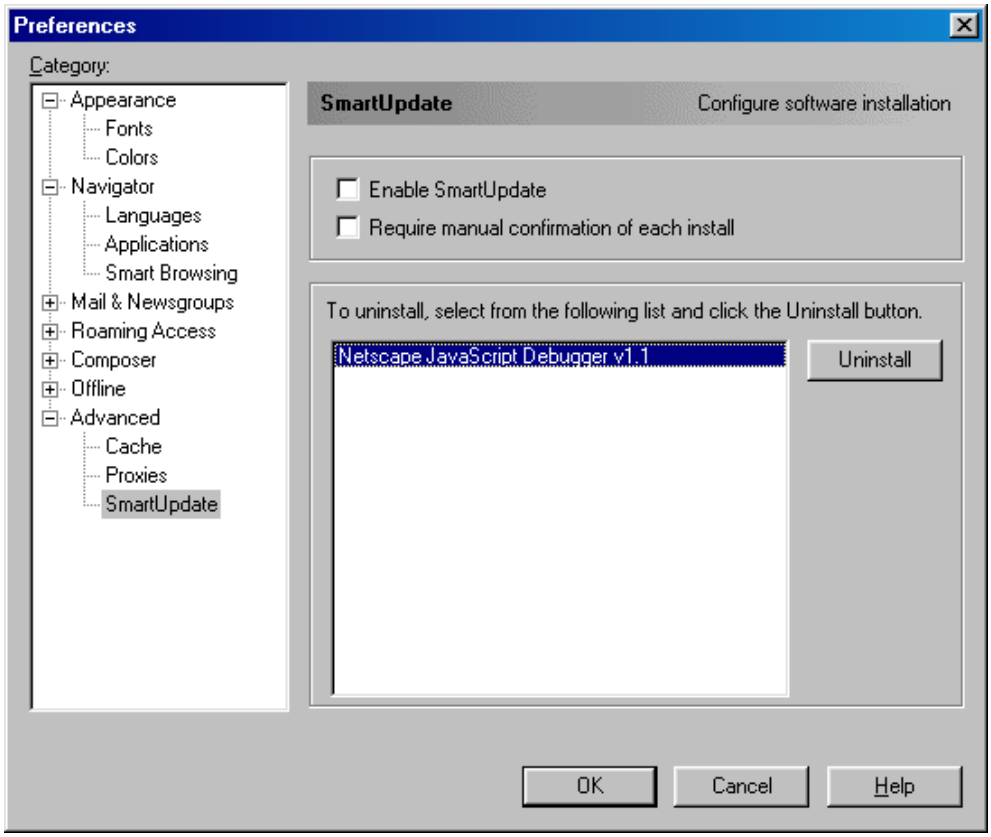


Figura C.5: SmartUpdate en Netscape Communicator

C.4. Microsoft Internet Explorer

En las versiones 5.0 o superiores de este navegador, se muestra automáticamente una ventana de alerta (Figura C.6) por cada uno de los errores de *JavaScript* que se produzca en una página. Estas ventanas de alerta se pueden desactivar de dos formas:

1. Desactivando la casilla **Mostrar siempre este mensaje cuando una página contenga errores** en la propia ventana de alerta (Figura C.6). La próxima vez que se localice un error de *JavaScript*, no se mostrará la ventana de alerta.

2. Desactivando la opción **Mostrar una notificación sobre cada error de secuencia de comandos** en el menú **Herramientas** → **Opciones de Internet...** → **Avanzadas** (Figura C.8). A través de esta opción se puede activar o desactivar tantas veces como se quiera.

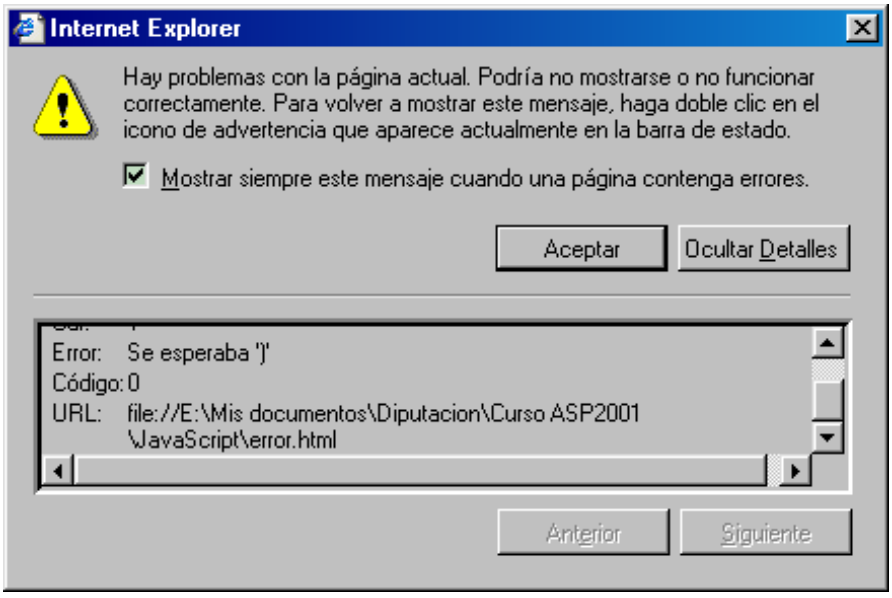


Figura C.6: Mensaje de alerta de Microsoft Internet Explorer

En cualquier caso (esté activado o desactivado), si se encuentra un error, en la barra de estado del navegador se muestra un mensaje (Figura C.7) cuando se localiza un error. Pulsando sobre el icono (triángulo amarillo), se abre la ventana de alerta.

En la ventana de alerta (Figura C.6) se muestra un mensaje con información sobre el error: línea, carácter, error, código y **URL** (fichero que contiene el error). Estos

Figura C.7: Mensaje de alerta en la barra de estado de Microsoft Internet Explorer

mensajes suelen ayudar bastante poco a localizar el error, ya que a veces no señalan el sitio exacto del error.

Si se tiene instalado algún entorno de programación de MICROSOFT, como *Visual Basic* o *Visual C++*, a Microsoft Internet Explorer se incorpora una opción de depuración de secuencia de comandos (código de *script*), tal como se ve en la Figura C.8 y Figura C.11. Esta opción permite depurar el código de *JavaScript* mediante el depurador empleado en los entornos de programación de MICROSOFT. El depurador (Figura C.12) permite realizar un depuración del código mucho más precisa, ya que incorpora inspección de variables, ejecución paso a paso, puntos de interrupción, visor de objetos, etc.

Para que funcione el depurador, tiene que estar desactivada la opción **Deshabilitar depuración de secuencias de comandos** en el menú **Herramientas** → **Opciones de Internet...** → **Avanzadas** (Figura C.8).

Cuando se tiene activada la opción de depurador de secuencias, las ventanas de alerta cambian a otras que muestran la línea donde se produce el error, el error producido y la posibilidad de abrir el depurador de secuencias para depurar el código de *JavaScript* (Figura C.9 y Figura C.10).

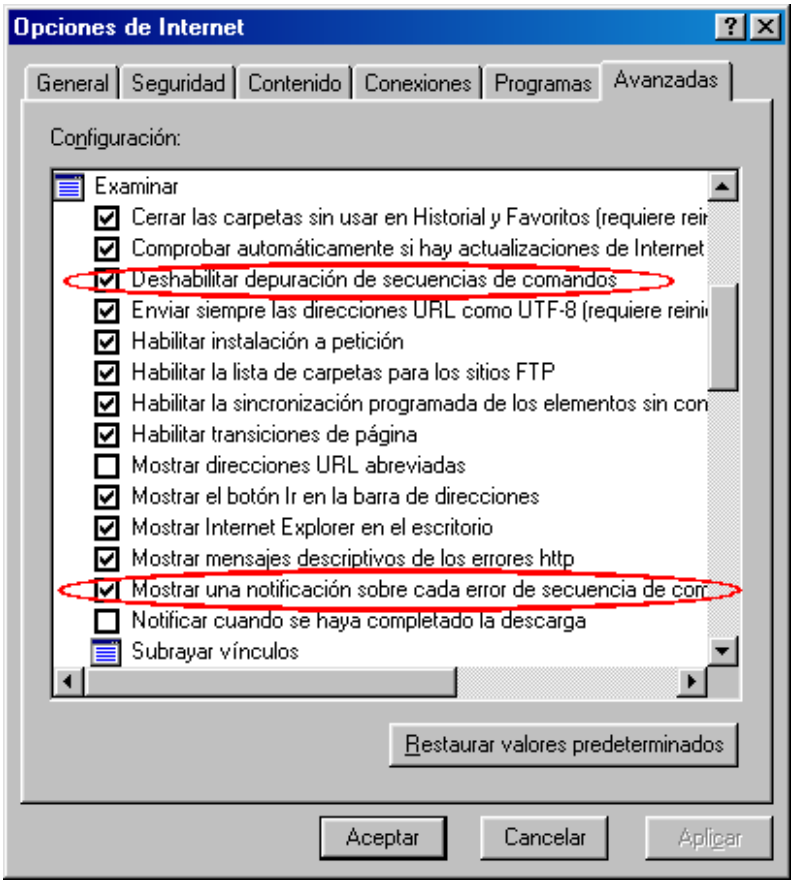


Figura C.8: Opciones de Microsoft Internet Explorer

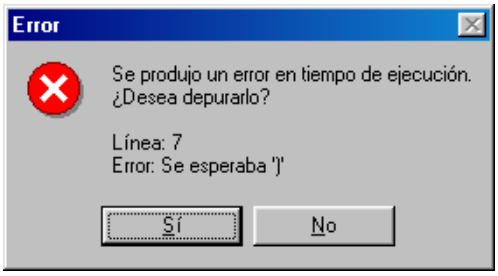


Figura C.9: Mensaje de error en Microsoft Internet Explorer

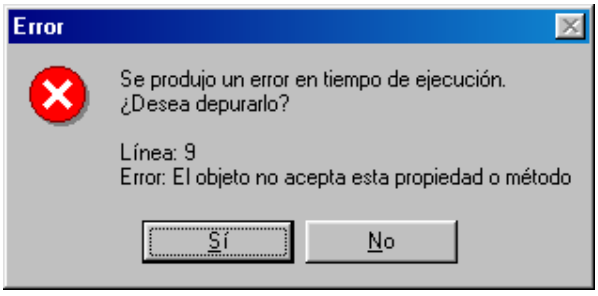


Figura C.10: Mensaje de error en Microsoft Internet Explorer

Figura C.11: Nuevas opciones de Microsoft Internet Explorer

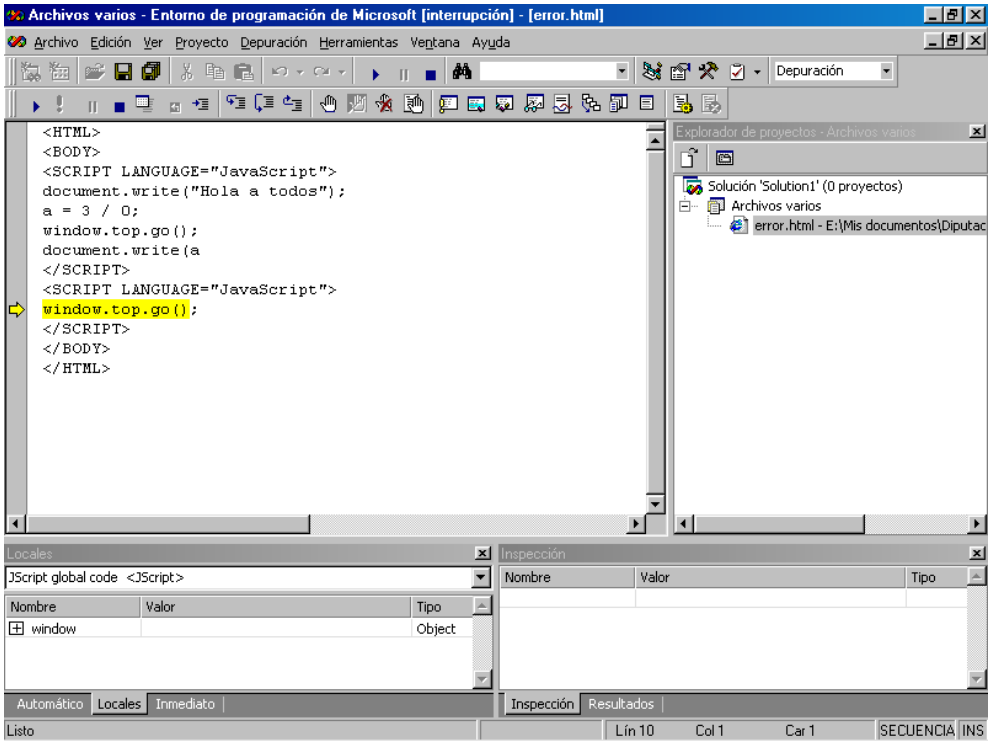


Figura C.12: Depurador de Microsoft

Bibliografía recomendada

A continuación se incluye una serie de libros recomendados agrupados en cuatro categorías: General, HTML, Diseño web y JavaScript.

General

- Eduardo Parra Murga. *Diccionario de Internet.* Nóesis, Madrid, 1996
- Jesús Bobadilla Sancho. *Superutilidades para Webmasters.* Osborne McGraw-Hill, Madrid, 1999
- H. M. Deitel, P. J. Deitel, T. R. Nieto. *Internet and World Wide Web. How to program.* Prentice Hall, New Jersey, 2000
- Chris Bates. *Web Programming: Building Internet Applications.* John Wiley & Sons, New York, 2002

HTML

- Sergio Ríos Aguilar. *Lenguajes HTML, Java y CGI. El diseño de páginas Web para Internet a su alcance.* Abeto Editorial, Madrid, 1996
- Ian S. Graham. *HTML Sourcebook: A Complete Guide to HTML 3.2 and HTML Extensions.* John Wiley & Sons, New York, 1997
- Molly Holzschlag. *Special Edition Using HTML 4.* Que, Macmillan Computer Publishing, 1999

Diseño web

- Sarah Horton Patrick J. Lynch. *Principios de diseño básicos para la creación de sitios web.* Ediciones G. Gili, México, 2000
- Jakob Nielsen. *Usabilidad: diseño de sitios Web.* Prentice Hall, Madrid, 2000

- Steve Krug. *No me hagas pensar: una aproximación a la usabilidad en la Web.* Prentice Hall, Madrid, 2001

JavaScript

- Danny Goodman. *Programación en JavaScript.* Vía@Internet, Anaya Multimedia, Madrid, 1996
- Oscar González Moreno. *Programación en JavaScript.* Guías Prácticas, Anaya Multimedia, Madrid, 1998
- José Manuel Alarcón. *Programación en JavaScript (actualizada hasta JavaScript 1.3 y JScript 5).* Guías Prácticas, Anaya Multimedia, Madrid, 2000

Índice alfabético

A

B

C

D

E

F

G

H

I

J

L

M

N

U

V

W

X

Y