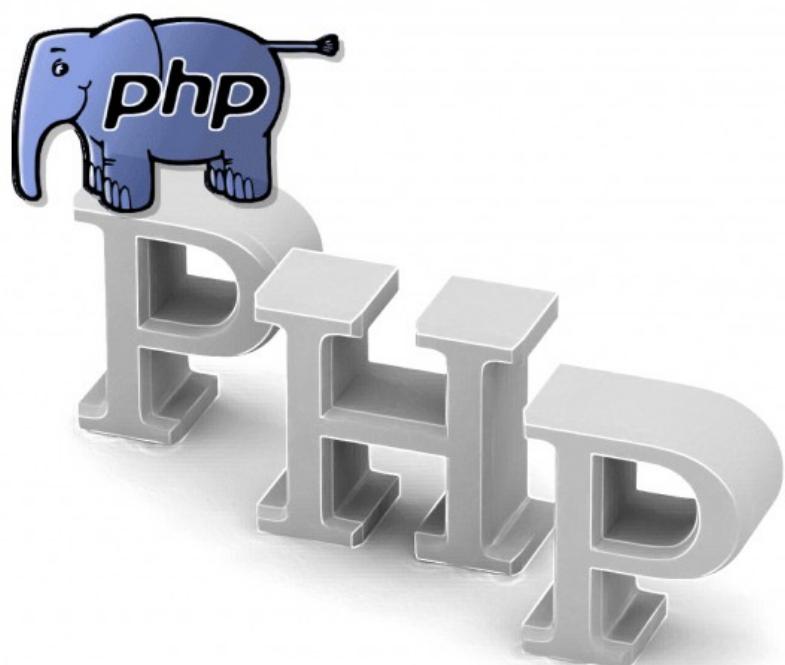


PHP

El nuevo PHP paso a paso.



Vicente Javier Eslava Muñoz

El nuevo PHP paso a paso.

Por Vicente Javier Eslava Muñoz

© Vicente Eslava
© El nuevo PHP paso a paso
ISBN papel : **978-84-686-4109-6**
ISBN Digital: **978-84-686-4110-2**
Impreso en España
Editado por Bubok Publishing S.L

A la princesa de la noche que me tiene de rodillas.
À ma première épousse.

I love you.

El nuevo PHP paso a paso.

Indice de contenidos

Prólogo	8
1 – Introducción	9
2 – El lenguaje PHP, conceptos básicos	27
3 – Estructuras de control	50
4 – Funciones	62
5 – Arrays	77
6 – Cadenas	87
7 – Formularios	96
8 – Cookies	106
9 – Sesiones	110
10 – Ficheros	113
11 – Programación orientada a objetos	122
12 – Publicación	141
ANEXO I: Usabilidad	147
ANEXO II: Accesibilidad	154
ANEXO III: Otras tecnologías	157
ANEXO IV: Solución de ejercicios	159
ANEXO V: Entornos de desarrollo	176
Bibliografía	182

Prólogo

En el tiempo que llevamos relacionados con la informática, hemos sido testigos de cambios importantes, pero también hemos visto algunos momentos en verdad revolucionarios, tanto a nivel hardware como a nivel software. En estos días, los ordenadores han alcanzado una capacidad sorprendente, mientras que el software se vuelve cada vez más completo y más útil. Gran parte del potencial que han demostrado hardware y software se debe gracias a la existencia de Internet, la red de redes, que expande la funcionalidad de cualquier sistema, simplemente "conectándolo" al mundo. Ahora, Internet está a punto de sufrir un cambio radical con la aparición de la quinta versión del HyperText Markup Language y es probable que nosotros cambiemos junto con Internet, porque la red ya no será la misma.

PHP posee la característica de brindar al desarrollador una gran libertad en el momento de desarrollar aplicaciones. Además un punto que destaca respecto a otras alternativas equivalentes es la curva de aprendizaje demandada para aquellos que quieran aprender es muy corta. Esto se debe a su sintaxis intuitiva y eficaz, la cual nos permite en pocas líneas de código aplicar soluciones eficaces.

Una aplicación web basada en PHP puede ejecutarse en cualquier ordenador independientemente del sistema operativo: Lo único que necesita es un servidor de aplicaciones capaz de interpretar su código y devolver el HTML correspondiente, que en este caso se leería utilizando un navegador Web. Ni siquiera sería necesaria una conexión a Internet, porque dichas aplicaciones pueden trabajar de forma local.

Los navegadores Web se han convertido en una pieza central de la actividad de cualquier usuario que se sienta frente a un ordenador. Desde consultar el correo electrónico hasta descargar archivos, buscar información, e incluso leer estas mismas líneas, se hace a través de alguna clase de navegador Web. La conveniencia del navegador Web es innegable, y aunque los sistemas operativos no van a desaparecer, al menos deberán reconocer el protagonismo de los navegadores en los años por venir, especialmente con el poder que les entrega PHP.

La creciente dependencia de la sociedad en el uso de las tecnologías significa que es más importante que nunca entender cómo funcionan los sistemas que el hombre utiliza. Una manera práctica de tomar el control de los sistemas que utilizamos en el día a día es entender cómo funcionan. ¿Cómo se hace esto? Aprendiendo a desarrollarlos.

¿Por qué este libro?

La presente obra pretende dar respuesta a las inquietudes actuales en la materia de programación web y en particular del lenguaje PHP. Es un libro que está dirigido a estudiantes de todas las edades, desde la ESO, pasando por los ciclos formativos específicos, hasta la universidad y a todas aquellas personas que tengan una inquietud en el tema.

PHP es una de las piedras angulares de la programación open source web, un marco diseñado para apoyar la innovación y fomentar el potencial que tiene la web para ofrecer.

¡PHP es el presente y el futuro de la web!

Este libro presume como requisitos el conocimiento básico de HTML, si no se tienen se recomienda la consulta previa de "**HTML, presente y futuro de la web.**" del mismo autor.

1 – Introducción.

1.1 – Lenguajes de programación web.

Tipos de páginas web.

Las primeras páginas que aparecieron en Internet eran muy sencillas y poco interactivas. En ellas sólo se permitía mostrar información, ya se tratara de texto o imágenes, y seguir enlaces a nuevas páginas.

Para compensar esta falta de interactividad aparecieron los lenguajes de programación para el cliente. Es decir, para el navegador de Internet. Estos lenguajes aportaban una interactividad limitada. Dentro de esta categoría podemos destacar el lenguaje Javascript. Un par de ejemplos típicos de programación mediante Javascript podrían ser un menú jerárquico desplegable o una calculadora.

Pero hay multitud de funcionalidades que están fuera del alcance de los lenguajes de programación en el cliente. La interactividad que permiten se reduce prácticamente a mejorar la apariencia de la información en la página web, a añadir efectos visuales que aunque puedan ser curiosos no aportan un valor añadido, y a realizar operaciones matemáticas de mayor o menor dificultad con datos suministrados por el usuario final.

Con la llegada de los lenguajes de servidor aparecieron las primeras páginas dinámicas, cuyos principales avances consiste en la extracción de información de una base de datos, y el almacenamiento de información entre acciones. Hoy en día, la mayor parte de Internet está formada por este último tipo de páginas.

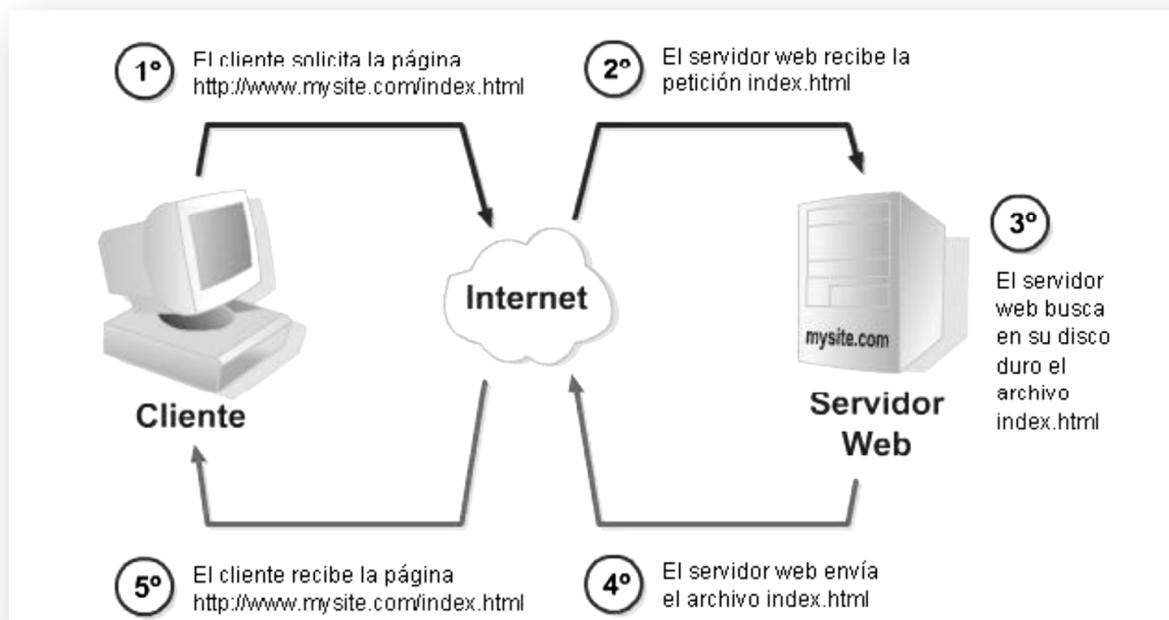
El buscador de Internet que usamos cada vez que queremos encontrar páginas de una temática y las tiendas virtuales que podemos encontrar en Internet son páginas dinámicas.

Generación de páginas web

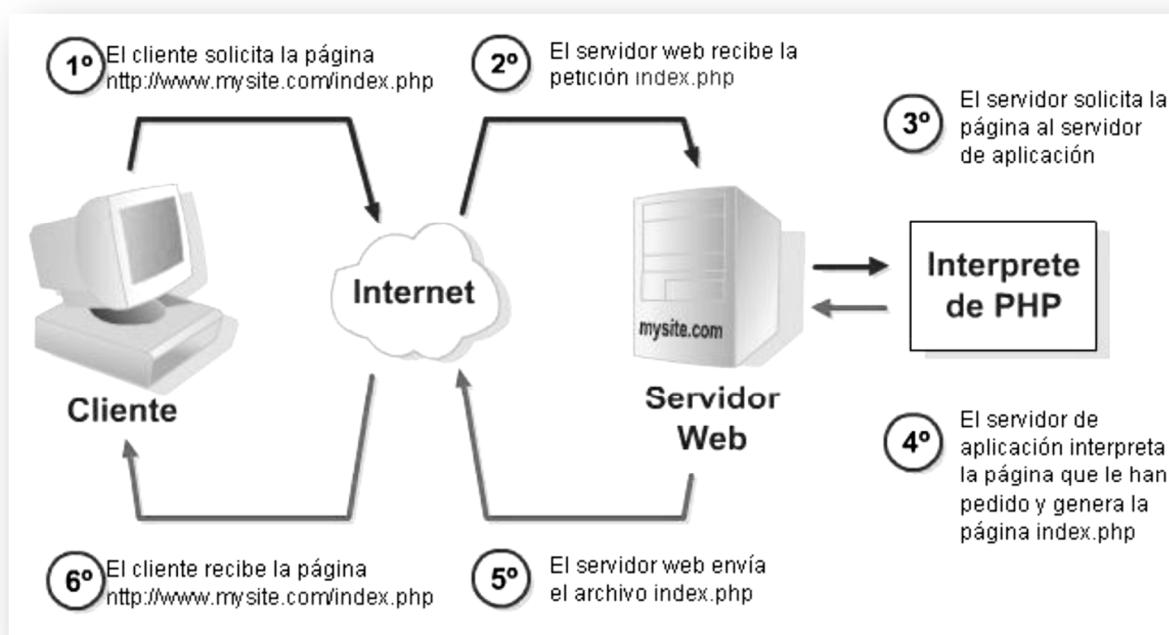
Para poder comprender la diferencia entre una página estática y una página dinámica, hay que tener presente el proceso que sigue una página web desde que la solicitamos, mediante un clic en un enlace del navegador, hasta que la recibimos y se muestra por pantalla.

Página estática. Elementos del proceso:

- a) Navegador: aplicación que permite al usuario recibir y visualizar las páginas web.
- b) Cliente: en el ámbito de Internet, es el ordenador que está conectado a la red. Su función se limita a realizar las peticiones de páginas web y mostrar los resultados en el navegador.
- c) Internet: es el medio físico por el que se transmiten las peticiones web, y sus respuestas.
- d) Servidor web1 (hardware): ordenador conectado a Internet que aloja páginas web.
- e) Servidor web (software): programa que transfiere páginas web a los clientes mediante el protocolo HTTP.



Página dinámica. Ahora disponemos de un nuevo elemento, el servidor de aplicación.



f) Servidor de aplicación: es el programa que colabora con el servidor web para generar páginas dinámicas.

A diferencia de las páginas estáticas, en las que se guardaba en el disco duro el archivo html correspondiente, las páginas dinámicas no se envían tal cual están almacenadas en el disco duro. En su lugar hay un programa, que se ejecuta en el

servidor de aplicaciones, encargado de interpretar y ejecutar determinados ficheros con el fin de crear el archivo que se transferirá al cliente.

Lenguajes de programación web.

Para generar páginas web se han usado a lo largo del tiempo un gran número de tecnologías. En los comienzos no había ningún lenguaje de programación específico, por lo que se utilizaban lenguajes de uso más general como C, C++, Basic o Pascal.

Para comunicar estos programas con el servidor web se hacía uso de la especificación CGI, siglas de Common Gateway Interface, o en castellano Pasarela de Interfaz Común. Hoy en día, aún hay aplicaciones que siguen este enfoque, y comúnmente se las conoce como CGI's.

Pero la utilización de lenguajes de uso general hacía complicado el proceso de construir páginas web dinámicas. El código se escribía de forma casi artesanal, y era difícil de mantener y actualizar. Además, el código que funcionaba en un servidor web podía no funcionar en otro o requerir de un gran esfuerzo para su instalación. Además, los CGI suelen ser poco eficientes.

Afortunadamente, hoy en día el panorama ha cambiado radicalmente. El éxito de Internet ha ido acompañado de la aparición de lenguajes de programación web que facilitan todo el proceso. Para especializarse en la creación de páginas web dinámicas, estos lenguajes intentan cumplir los siguientes objetivos:

- Ser de fácil uso, para permitir una curva de aprendizaje sencilla.
- Adaptarse eficientemente a la rápida evolución de Internet, ofreciendo de manera continua nuevas características y actualizaciones de seguridad.
- Tener una gran comunidad de usuarios que garanticen el éxito del producto.

Estos lenguajes han proliferado de tal modo, que en la actualidad hay una gran cantidad de tecnologías disponibles, con la intención de acaparar la mayor cuota de mercado posible.

De entre estos lenguajes podemos destacar tres tecnologías mediante las cuales se desarrollan la mayoría de los sitios web: ASP, JSP y PHP.

ASP: Active Server Pages.

Esta es una tecnología propietaria de Microsoft. Actualmente, ésta engloba en su estrategia .NET. Su principal ventaja es su sencillez, además de una muy buena integración con otros productos de Microsoft como su entorno de desarrollo .NET, que permite crear aplicaciones en un tiempo record. Se integra muy bien con el servidor web de Microsoft, IIS (Internet Information Center).

JSP: Java Server Pages.

Esta es una tecnología basada en uno de los lenguajes de uso general más populares, Java. Es quizás el más complejo de los 3 pero a su vez el más potente. La gran comunidad de usuarios garantiza la existencia de numerosas herramientas, librerías de código abierto y bases de conocimiento. Además se integra muy bien en el servidor web mayoritario, Apache, gracias a la extensión Tomcat.

PHP: PHP Hypertext Preprocessor.

A su favor tiene el ser una iniciativa de código abierto, por lo que es mantenido e impulsado por una gran comunidad de programadores. Además, es el más sencillo de los tres, por lo que atrae a usuarios de muy diferentes procedencias: programadores en lenguajes de programación más generalistas, o diseñadores que desean completar sus conocimientos.

1.2 - Historia de PHP.

El lenguaje PHP fue creado por una gran comunidad de personas. El sistema fue desarrollado originalmente en el año 1994 por Rasmus Lerdorf como un CGI escrito en C que permitía la interpretación de un número limitado de comandos. El sistema fue denominado Personal Home Page Tools y adquirió relativo éxito gracias a que otras personas pidieron a Rasmus que les permitiese utilizar sus programas en sus propias páginas. Dada la aceptación del primer PHP y de manera adicional, su creador diseñó un sistema para procesar formularios al que le atribuyó el nombre de FI (Form Interpreter) y el conjunto de estas dos herramientas, sería la primera versión compacta del lenguaje: PHP/FI.

La siguiente gran contribución al lenguaje se realizó a mediados del 97 cuando se volvió a programar el analizador sintáctico, se incluyeron nuevas funcionalidades como el soporte a nuevos protocolos de Internet y el soporte a la gran mayoría de las bases de datos comerciales. Todas estas mejoras sentaron las bases de PHP versión 3. Actualmente PHP se encuentra en su versión 4, que utiliza el motor Zend, desarrollado con mayor meditación para cubrir las necesidades actuales y solucionar algunos inconvenientes de la anterior versión. Algunas mejoras de esta nueva versión son su rapidez -gracias a que primero se compila y luego se ejecuta, mientras que antes se ejecutaba mientras se interpretaba el código-, su mayor independencia del servidor web - creando versiones de PHP nativas para más plataformas- y un API más elaborado y con más funciones. Para *usuarios finales*:

En la siguiente tabla observaremos las versiones de php fecha de creación y cambios entre unas de otras.

Versión	Fecha	Cambios más importantes
PHP 1.0	8 de junio de 1995	Oficialmente llamado "Herramientas personales de trabajo (PHP Tools)". Es el primer uso del nombre "PHP".
PHP Versión 2 (PHP/FI)	16 de abril de 1996	Considerado por el creador como la "más rápida y simple herramienta" para la creación de páginas webs dinámicas.
PHP 3.0	6 de junio de 1998	Desarrollo movido de una persona a muchos desarrolladores. Zeev Suraski y Andi Gutmans reescriben la base para esta versión.
PHP 4.0	22 de mayo de 2000	Se añade un sistema más avanzado de análisis de etiquetas en dos fases análisis/ejecución llamado el motor Zend.
PHP 4.1	10 de diciembre de 2001	Introducidas las variables superglobales (\$_GET, \$_SESSION, etc.).
PHP 4.2	22 de abril de 2002	Se deshabilitan register_globals por defecto.
PHP 4.3	27 de diciembre de 2002	Introducido la CLI, en adición a la CGI.
PHP 4.4	11 de julio de 2005	
PHP 5.0	13 de julio de 2004	Motor Zend II con un nuevo modelo de objetos.

PHP 5.1	25 de noviembre de 2005	
PHP 5.2	2 de noviembre de 2006	Habilitado el filtro de extensiones por defecto.
PHP 5.2.4	30 de agosto de 2007	
PHP 5.2.5	8 de noviembre de 2007	Versión centrada en mejorar la estabilidad (+60 errores solucionados).
PHP 5.2.8	8 de diciembre de 2008	
PHP 5.2.9	26 de febrero de 2009	Diversas mejoras en el ámbito de la seguridad (+50 errores solucionados).
PHP 5.4.0	1 de marzo de 2012	Soporte para Trait y sintaxis abreviada de array. Elementos removidos: register_globals, safe_mode, allow_call_time_pass_reference, session_register(), session_unregister() y session_is_registered(). Servidor web incorporado. 14 Varias mejoras a características existentes y al rendimiento, y requerimientos de memoria menores.
PHP 5.4.1	26 de abril de 2012	Varios bug fixes y mejoras de seguridad.
PHP 5.4.2	3 de mayo de 2012	Parche de seguridad para arreglar vulnerabilidad del parámetro query string de PHP-CGI.
PHP 5.4.3	8 de mayo de 2012	Arreglo de vulnerabilidad para instalaciones basadas en CGI y también para la vulnerabilidad de desbordamiento de búfer para apache_request_headers()
PHP 5.5.0	20 de junio de 2013	Nuevos generadores para bucles, empty() soporta expresiones. Se pierde el soporte para Windows XP y Windows Server 2003.
PHP 6	Sin fecha	Soportar UNICODE PELC Mejoras en orientación a objetos Limpieza de funcionalidades obsoletas

Los **parámetros** se identifican por un término, seguido de un signo de igual y a continuación, entre comillas, el valor que le queramos asignar. Aunque podemos usar comillas simples, normalmente se usan siempre **comillas dobles** para englobar el valor de cada parámetro.

1.3 – Características de PHP.

Toda página HTML está comprendido entre las etiquetas <HTML> y </HTML>:

<HTML> [Todo el documento] </HTML>

Un documento HTML en sí está dividido en dos zonas principales:

- El **encabezamiento**, entre las etiquetas <HEAD> y </HEAD>
- El **cuerpo**, comprendido entre las etiquetas <BODY> y </BODY>

Fácil de usar.

PHP 5 es un lenguaje muy fácil de aprender con respecto a otros lenguajes utilizados para el mismo propósito, como JAVA o ASP. Debido a esto no es necesario hacer un estudio muy concienzudo de sus funciones para realizar programas sencillos que nos resuelvan la mayoría de los problemas diarios.

La mayoría de las funciones más usuales están disponibles por defecto, como la conexión a bases de datos o la utilización de servidores IMAP. Existe una gran cantidad de páginas con documentación y programas hechos por desarrolladores que se pueden leer y modificar libremente.

Embebido en HTML.

Las páginas creadas con PHP son simples páginas en HTML que contienen, además de las etiquetas normales, el programa que queremos ejecutar. Por ejemplo:

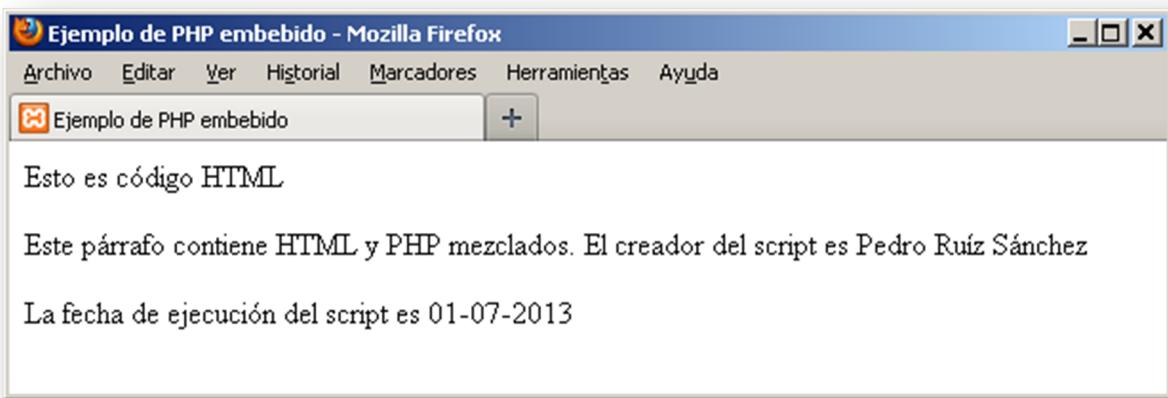
```
<HTML>
<HEAD>
    <TITLE>Ejemplo de PHP embebido</TITLE>
</HEAD>
<BODY>
    <B>Esto es código HTML </B>
    <?php
        //A partir de aquí comienza el código PHP
        $nombre="Pedro";
        $apellidos="Ruiz Sánchez";
        $fecha=date('d-ra-Y1');
    ?>
    <P>Este párrafo contiene HTML y PHP mezclados.
    El creador del script es <?php echo ("$nombre $apellidos"); ?></P>
    <P>La fecha de ejecución del script es <?php echo("$fecha");?></P>
</BODY>
</HTML>
```

Cuando el cliente solicita esta página, el servidor preprocesa los datos y ejecuta las instrucciones de PHP. En este caso, las variables (las palabras que tienen el signo \$ delante) se llenan con los valores que hay a la derecha del signo igual. Una vez resuelto todo el proceso, el servidor le envía al cliente una página sólo con etiquetas de HTML. La siguiente figura muestra este proceso.

Si inspeccionamos ahora el código que tenemos en el navegador nos daremos cuenta de que las etiquetas de PHP han desaparecido.

```
<HTML>
<HEAD>
    <TITLE>Ejemplo de PHP embebido-:</TITLE>
</HEAD>
<BODY>
    <B>Esto es código HTML</B>
    <P>Este párrafo contiene HTML y PHP mezclados.
    El creador del script es Pedro Ruiz Sánchez</P>
    <P>La fecha de ejecución del script es 0 1-07-2013<P>
</BODY>
</HTML>
```

La consecuencia más inmediata es que no es necesario compilar el programa en código binario antes de poder testar si funciona o no. PHP es un lenguaje interpretado como otros muchos en el mercado (ASP, Python o JSP).



Multiplataforma.

PHP se ejecuta en multitud de plataformas, Sistemas Operativos y Servidores existentes. Es compatible con los tres servidores líderes del mercado: Apache, Microsoft Internet Information Server y Netscape Enterprise Server.

Sistema Operativo	Servidores
AIX, A/UX, BSDI, Digital UNIX/Tru64, FreeBSD, HPUX, IRIX, MacOS X, gnuLinux, gnuLinEx, NetBSD, OpenBSD, SCO UnixWare, Solaris, SunOS, Ultrix, Xenix y muchos más Windows 98/Me, Windows NT/2000/XP/2007,2008	Apache, fhttpd, Netscape US, PWS, Netscape, Apache, Omni

Puesto que PHP se ejecuta en todos los Sistemas Operativos indicados en la tabla anterior y en la mayoría de las plataformas hardware existentes (Intel, AMD, PowerPc, SPARC, etcétera), nos será muy sencillo conseguir un laboratorio de pruebas para nuestros script.

Licencia Open Source.

La licencia de Código Abierto implica que el código fuente de PHP es libre de ser descargado e inspeccionado por nosotros. La consecuencia principal es que el coste del producto en la mayoría de los casos es de 0 Euros. Tener el código fuente de PHP sirve, entre otras cosas, para poder hacer nuestro servidor a medida, es decir, podemos compilar el programa con las opciones que realmente utilicemos (base de datos, LDAP).

Si acompañamos Apache, el servidor más popular, a la instalación de PHP y añadimos alguna base de datos Open Source como PostgreSQL, tendremos un sistema completo de script de servidor, cuyo coste es nulo, frente a otras opciones en las que es necesario el uso de licencias.

Multitud de Extensiones.

El lenguaje PHP se desarrolla para dar la mayor versatilidad y flexibilidad a los usuarios que lo utilizan. Es por esto por lo que existen muchas extensiones del lenguaje que permiten utilizar nuevas bases de datos, protocolos, enlaces a librerías, etcétera.

El acceso a bases de datos tiene una gran potencia, implementando soporte nativo para 15 Sistemas Gestores de Bases de Datos muy populares. Encuanto a los protocolos, podemos contar con extensiones que controlan el acceso a LDAP, IMAP o POP3. También se ha cuidado el soporte para crear imágenes en tiempo de ejecución, gracias a la librería GD.

Dada la importancia del desarrollo del lenguaje XML en los últimos años, PHP incorpora tres métodos de acceso a este tipo de archivos, SAX, DOM y simpleXML. Además, se incorpora a PHP 5 la gestión de errores mediante el manejo de excepciones.

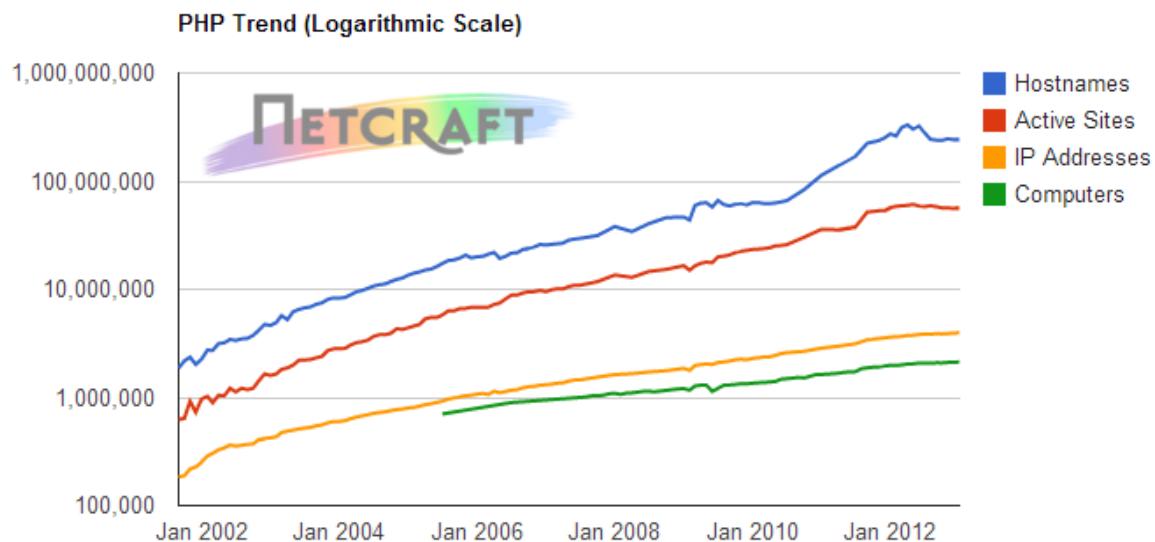
En esta versión se han añadido facilidades para utilizar los repositorios de código de PEAR.

Velocidad e incorporación de objetos.

El nuevo motor Zend 2.0 acelera los procesos de ejecución del código. Además, incorpora un nuevo modelo de objetos que permite crear herencia, mensajes y métodos privados, protegidos y públicos, clases abstractas e interfaces.

Popularidad.

El uso de PHP se ha disparado desde el año 2002 como puede verse en la siguiente figura, teniendo cerca de los 100 millones de sitios activos.



Gran Comunidad de apoyo.

PHP se ha escrito bajo el auspicio del Código Abierto. Por lo tanto, existe una comunidad que apoya su desarrollo en colaboración. La ventaja principal es que existen

multitud de páginas, listas de correo y foros de debate cuyo tema de conversación es el manejo de este lenguaje de programación.

Esta comunidad sirve de apoyo para todos los que necesitamos conocer desde los aspectos más básicos, hasta las implementaciones más complicadas. Tan pronto como hagamos uso de esta ayuda, nos sentiremos obligados a prestar la nuestra a usuarios principiantes y así, la Comunidad se irá ampliando. Si nuestros conocimientos llegan a superar algún día los objetivos de este libro, podemos pensar en contribuir enviando fallos en el lenguaje, respondiendo a mensajes de las listas de correo, participando en foros de debate o escribiendo extensiones en lenguaje C.

1.4 - ¿Qué se puede hacer con PHP?

PHP puede hacer cualquier cosa que se pueda hacer con un script CGI, como procesar la información de formularios, generar páginas con contenidos dinámicos, o enviar y recibir cookies. Y esto no es todo, se puede hacer mucho más.

Existen principalmente tres campos en los que se usan scripts en PHP.

Scripts del lado-servidor. Este es el campo más tradicional y el principal foco de trabajo. Se necesitan tres cosas para que esto funcione. El intérprete PHP (CGI módulo), un servidor web y un navegador. Es necesario hacer funcionar el servidor, con PHP instalado. El resultado del programa PHP se puede obtener a través del navegador, conectándose con el servidor web. Consultar la sección Instrucciones de instalación para más información.

Scripts en la línea de comandos. Puede crear un script PHP y correrlo sin necesidad de un servidor web o navegador. Solamente necesita el intérprete PHP para usarlo de esta manera. Este tipo de uso es ideal para scripts ejecutados regularmente desde cron (en *nix o Linux) o el Planificador de tareas (en Windows). Estos scripts también pueden ser usados para tareas simples de procesamiento de texto. Consultar la sección Usos de PHP en la línea de comandos para más información.

Escribir aplicaciones de interfaz gráfica. Probablemente PHP no sea el lenguaje más apropiado para escribir aplicaciones gráficas, pero si conoce bien PHP, y quisiera utilizar algunas características avanzadas en programas clientes, puede utilizar PHP-GTK para escribir dichos programas. También es posible escribir aplicaciones independientes de una plataforma. PHP-GTK es una extensión de PHP, no disponible en la distribución principal. Si está interesado en PHP-GTK, puedes visitar las » páginas web del proyecto.

PHP puede ser utilizado en cualquiera de los principales sistemas operativos del mercado, incluyendo Linux, muchas variantes Unix (incluyendo HP-UX, Solaris y OpenBSD), Microsoft Windows, Mac OS X, RISC OS y probablemente alguno más. PHP soporta la mayoría de servidores web de hoy en día, incluyendo Apache, IIS, y muchos otros. Esto incluye cualquier servidor web que pueda utilizar el binario PHP de FastCGI, como lighttpd y nginx. PHP funciona ya sea como un módulo, o como un procesador de CGI.

De modo que, con PHP tiene la libertad de elegir el sistema operativo y el servidor web de su gusto. Además, tiene la posibilidad de utilizar programación por procedimientos, programación orientada a objetos (POO), o una mezcla de ambas.

Con PHP no se encuentra limitado a resultados en HTML. Entre las habilidades de PHP se incluyen: creación de imágenes, archivos PDF e incluso películas Flash (usando libswf y Ming) sobre la marcha. También puede presentar otros resultados, como XHTML y cualquier otro tipo de ficheros XML. PHP puede autogenerar éstos archivos y almacenarlos en el sistema de archivos en vez de presentarlos en la pantalla, creando un caché en el lado-servidor para contenido dinámico.

Una de las características más potentes y destacables de PHP es su soporte para una gran cantidad de bases de datos. Escribir una página web con acceso habilitado a una base de datos es increíblemente simple utilizando una de las extensiones específicas (por ejemplo, para mysql), o utilizar una capa de abstracción como PDO, o conectarse a cualquier base de datos que soporte el estándar de Conexión Abierta a Bases de Datos por medio de la extensión ODBC. Otras bases de datos podrían utilizar cURL o sockets, como lo hace CouchDB.

PHP también cuenta con soporte para comunicarse con otros servicios usando protocolos tales como LDAP, IMAP, SNMP, NNTP, POP3, HTTP, COM (en Windows) y muchos otros. También se pueden crear sockets puros e interactuar usando cualquier otro protocolo. PHP soporta WDDX para el intercambio de datos entre lenguajes de programación en web. Y hablando de interconexión, PHP puede utilizar objetos Java de forma transparente como objetos de PHP.

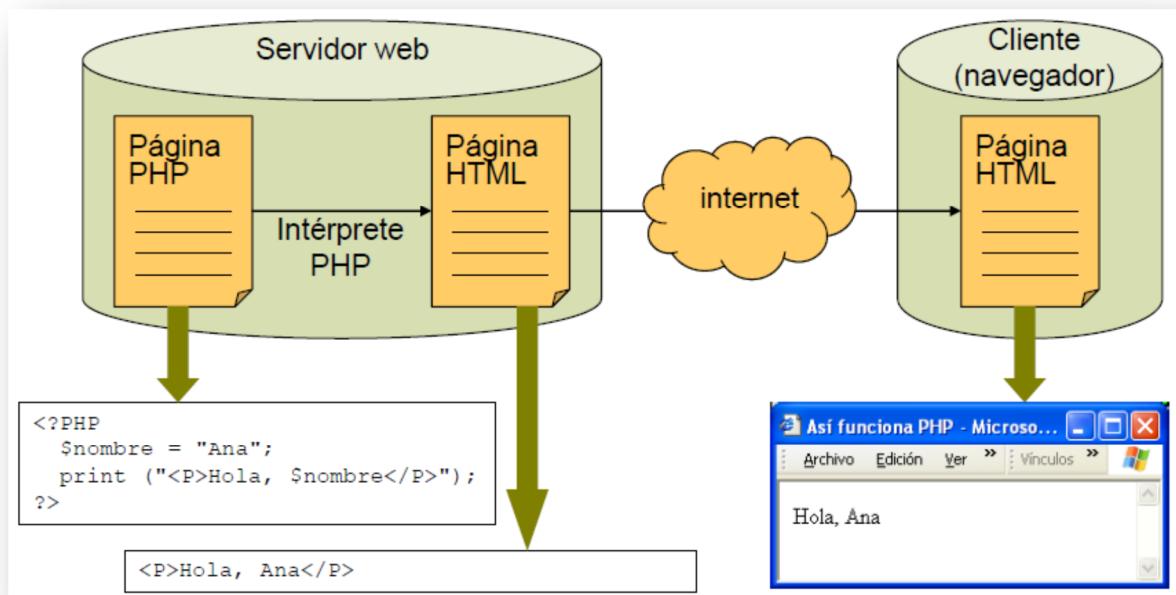
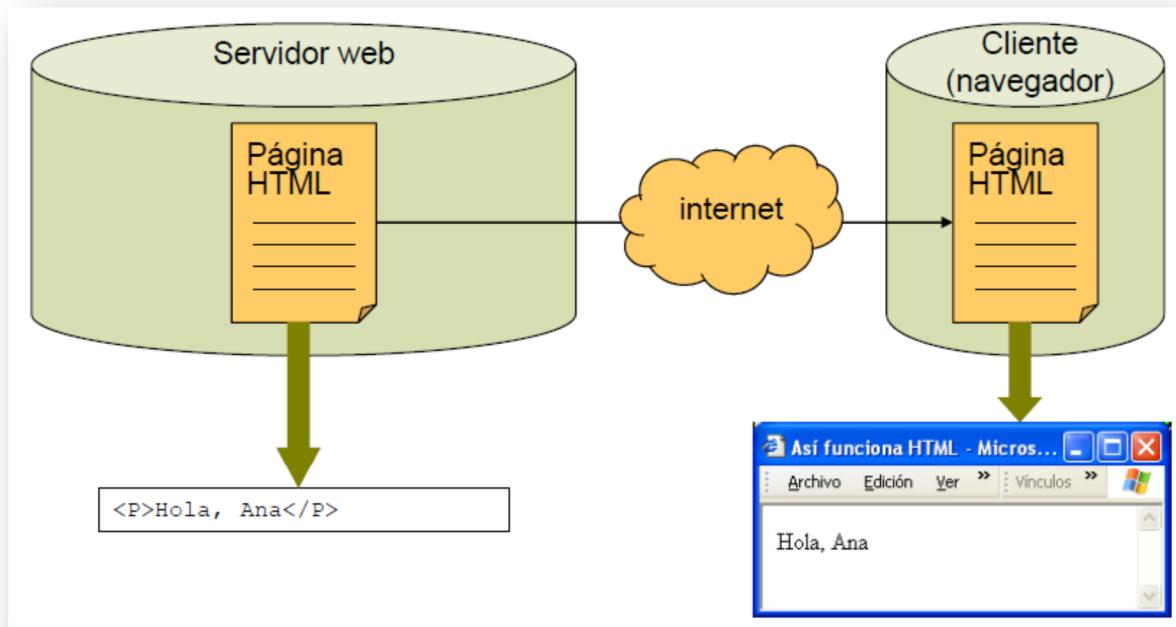
PHP tiene útiles características de procesamiento de texto, las cuales incluyen las Expresiones Regulares Compatibles de Perl (PCRE), muchas extensiones, y herramientas para el acceso y análisis de documentos XML. PHP estandariza todas las extensiones XML sobre el fundamento sólido de libxml2, y extiende este conjunto de características añadiendo soporte para SimpleXML, XMLReader y XMLWriter.

1.5 - Funcionamiento de un servidor web.

Cuando desde nuestro navegador solicitamos una página web cualquiera, por ejemplo <http://www.dominio.com/pagina.php>, lo que hacemos es dirigirnos a un ordenador (servidor) de la empresa que gestiona dominio.com que es donde está almacenada la pagina.php. El servidor procesa, es decir, ejecuta la página y devuelve el resultado en forma de código HTML.



A continuación se muestra la diferencia de proceso cuando solicitamos una página plana en HTML o una página que contiene programación, en este caso PHP.



Cuando se solicita una página php, primero se interpretan los comandos del código y se transforma su resultado en HTML. En ambos casos, el código que llega al cliente es HTML, esto es así para proteger la lógica interna de la programación de la página PHP. Desde el cliente no se puede acceder al código fuente PHP del servidor.

1.6 - Instalación de un servidor web.

Las páginas con código PHP son archivos que usan la extensión “.php”, Para poder visualizarlas, deben estar alojadas en un servidor web remoto capaz de interpretarlas.

Practicar con un servidor web profesional no es factible, ya que tiene las siguientes desventajas:

- Hay que contratar un servicio de alojamiento, lo que puede resultar costoso.
- Hay dependencia de la conexión a Internet.
- Gestionar los archivos que subimos al servidor a través de FTP es algo engorroso y lento, por lo que no se pueden hacer pruebas de código con agilidad.

La solución es sencilla: haremos que nuestro ordenador actúe como servidor web. A esta forma de trabajo se le llama coloquialmente, desarrollo en local.

Para desarrollar las habilidades necesarias para el manejo e instalación de una aplicación como un servidor Apache y su módulo para PHP, se necesitaría un curso completo. Debido a ello, Apache ha sido criticado en este aspecto, por no contar con una instalación y/o configuración gráfica/amigable al usuario, ya sea novel o experto.

Afortunadamente existen soluciones en el mercado que integran todo lo que necesitamos para trabajar, y además disponen de los llamados instaladores automáticos, que nos dejan preparado y, lo más importante, configurado de manera homogénea el entorno de desarrollo que vamos a emplear en todo este curso.

De entre los instaladores disponibles, hemos elegido una solución probada y que cubre varios sistemas operativos (Windows, GNU/Linux y Mac). Esta se llama Xampp en su versión reducida (lite).

En el libro vamos a usar la versión para Windows, que será la referente por facilidad de uso para los usuarios menos avanzados, aunque veremos algunas pantallas de Linux como alternativa (99% idénticas). En Linux y otros sistemas podemos usar Xampp o instalar Apache+PHP+MySQL por nuestra cuenta. Aunque esto último sólo es recomendable para usuarios avanzados, dado que hay que configurar el sistema.



Xampp en la versión 1.8.2 elegida instala las siguientes versiones de Apache, Php y Mysql entre otros:

- Apache 2.4.4
- MySQL 5.5.32
- PHP 5.4.16
- phpMyAdmin 4.0.4

A continuación veremos en detalle la instalación sobre el sistema operativo Windows, en Linux es similar.

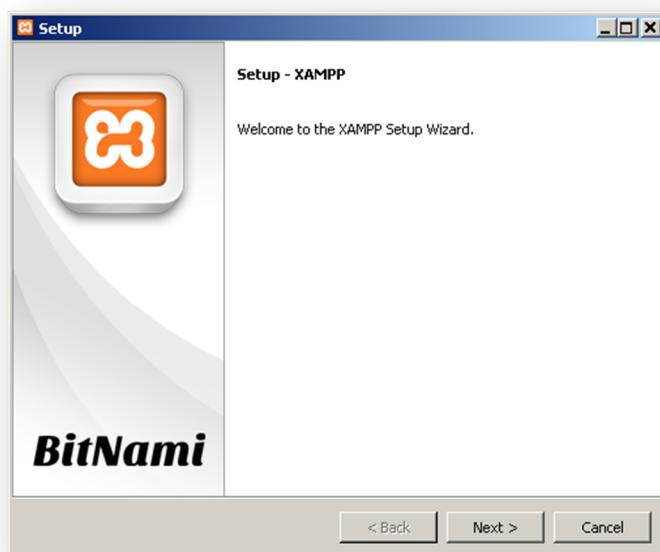
Instalación paso a paso sobre Windows.

1 – Introducción.

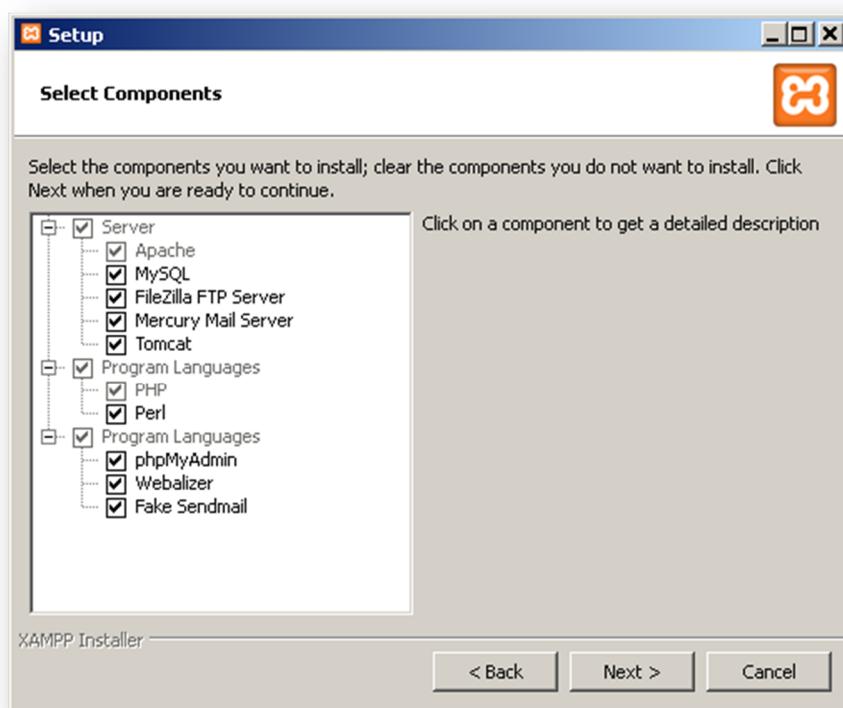
1. Descargamos la versión deseada según el sistema operativo que tengamos desde las direcciones siguientes:

- Windows: <http://www.apachefriends.org/en/xampp-windows.html#641>
- Linux: <http://www.apachefriends.org/en/xampp-linux.html#374>.

2. Ejecutamos el fichero. Al hacerlo nos aparecerá la siguiente ventana de bienvenida al asistente.

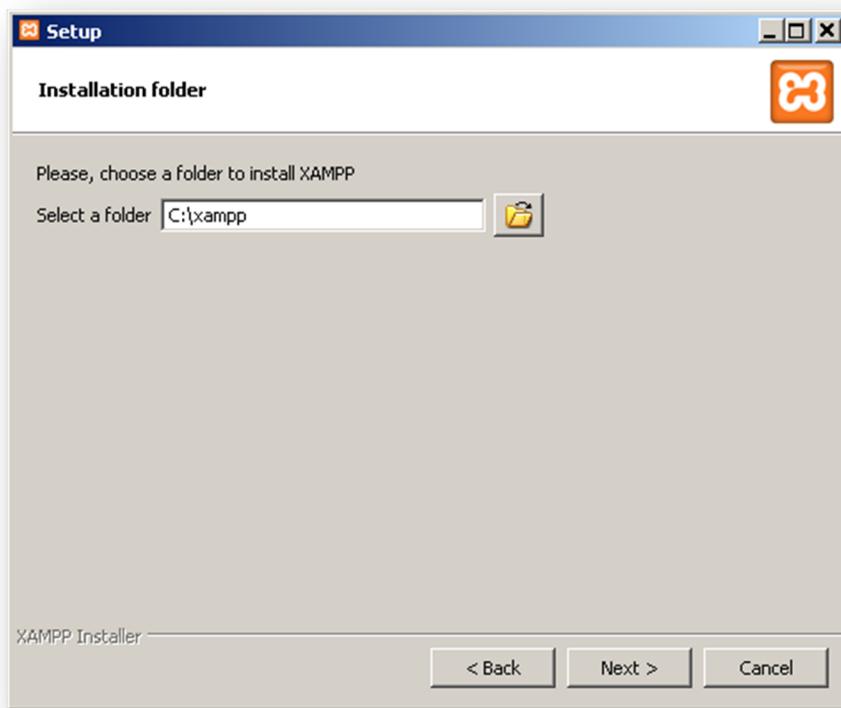


3. A continuación, el instalador nos informará sobre los componentes que podemos instalar, en el servidor, los lenguajes y las aplicaciones que se instalarán. Se recomienda seleccionarlos todos. Hacemos clic en “Next>”.

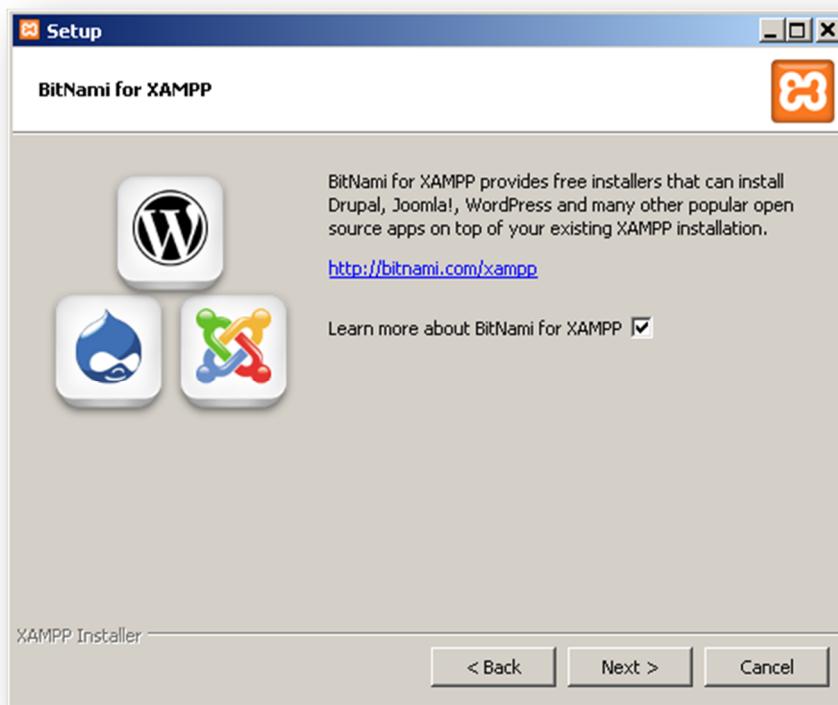


1 – Introducción.

4. A continuación deberemos seleccionar la ruta de instalación, tal y como se muestra en la siguiente pantalla:

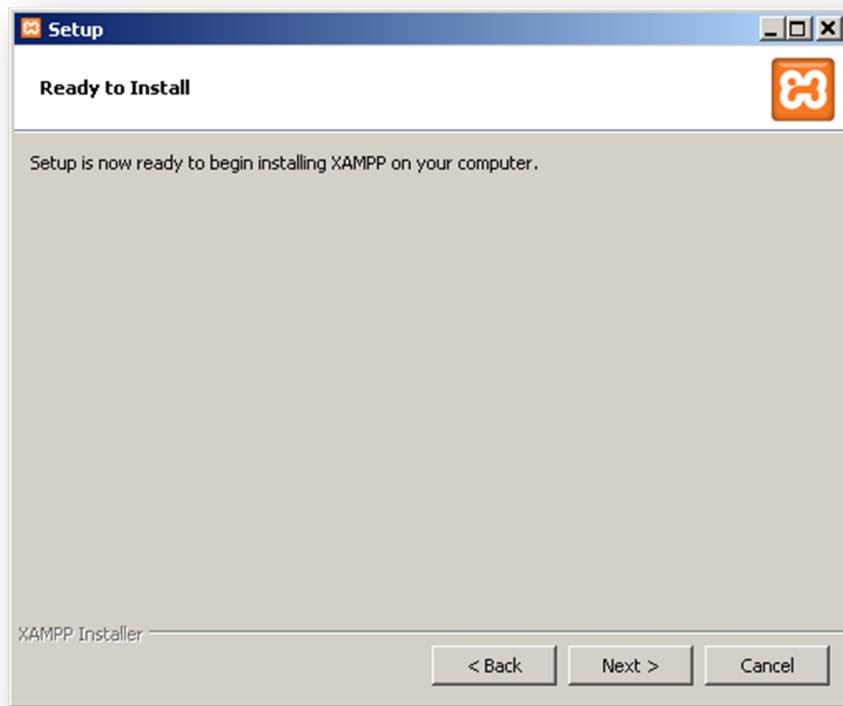


5. En este paso, el asistente nos ofrece la posibilidad de añadir aplicaciones de uso popular realizadas en PHP.



1 – Introducción.

6. Por último se nos informa que el asistente está listo para instalar bajo las premisas seleccionadas. Hacemos clic en “Next>” y esperamos a que termine la instalación.



7. Para comprobar que todo ha ido correctamente y que Apache está funcionando tenemos que ir a la dirección <http://localhost> con nuestro navegador favorito y deberíamos visualizar la siguiente página:

XAMPP
1.8.2
[PHP: 5.3.5]

Bienvenido
Estado
Chequeo de seguridad
Documentación
Componentes

PHP
phpinfo()
Administración de CD
Bioritmo
Instant Art
Agenda de teléfonos

Perl
perlinfo()
Libro de invitados

J2EE
Estado
Tomcat examples

Herramientas
phpMyAdmin
Webalizer
Mercury Mail
FileZilla FTP

©2002-2010
...APACHE
FRIENDS...

English / Deutsch / Français / Nederlands / Polski / Italiano /
Norwegian / Español / 中文 / Português (Brasil) / 日本語

Bienvenido a XAMPP para Windows!

Felicitaciones:
XAMPP se instaló con éxito en su ordenador!

Ahora se puede empezar a trabajar. :) Primero por favor pulse encima de »Estado« en la parte izquierda. De esta manera tendrá una visión de que es lo que funciona ya. Algunas funciones estarán desactivadas. Es intencionado. Son funciones, que no funcionan en todas partes o eventualmente podrían ocasionar problemas.

Atención: XAMPP fue modificado a partir de la versión 1.4.x a una administración de paquete único. Existen los siguientes paquetes/Addons:

- XAMPP paquete básico
- XAMPP Perl addon
- XAMPP Tomcat addon
- XAMPP Cocoon addon
- XAMPP Python addon (developer version)

Y en un futuro:

- XAMPP Utility addon (Accesorio pero aún inactivo)
- XAMPP Server addon (otros servidores aún inactivos)
- XAMPP Other addon (otras cosas útiles aún inactivas)

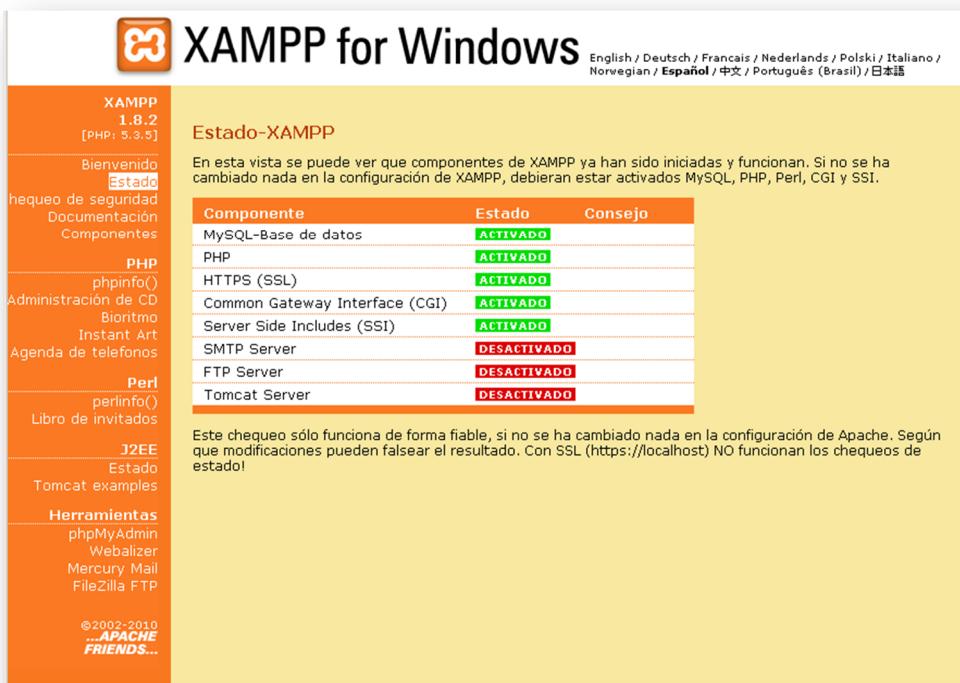
Por favor "Instalad" los paquetes adicionales, que aún necesiteis, simplemente a continuación. Despues de subirlos con éxito, por favor siempre accionar "setup_xampp.bat", para inicializar nuevamente XAMPP. A bueno, las versiones Instalador de los Addons individuales funcionan sólo si el paquete básico XAMPP tambien fue montado a partir de una versión instalador.

Para el soporte OpenSSL utilice por favor el certificado de chequeo con la URL <https://127.0.0.1> ó <https://localhost>

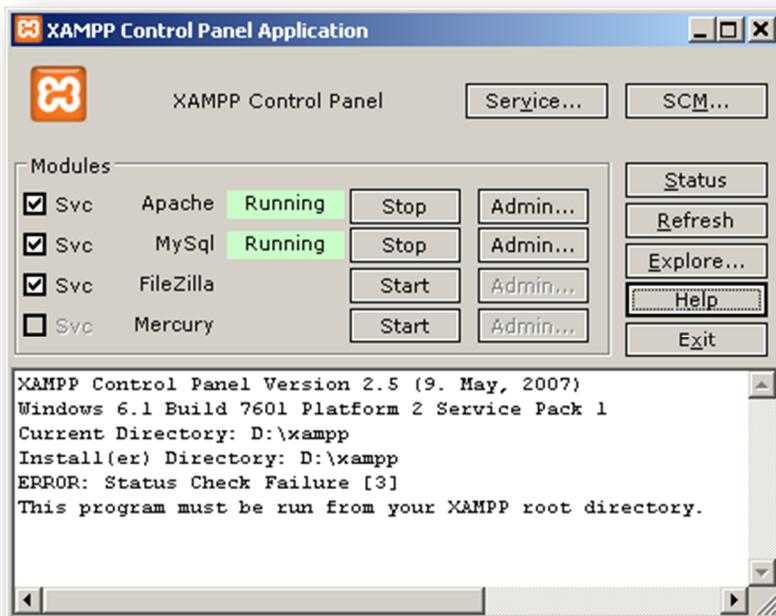
Os deseamos mucha diversión, Kay Vogelgesang + Kai 'Oswald' Seidler

1 – Introducción.

8. Pulsaremos en el enlace “Estado” para probar toda la instalación definitivamente. Para concluir, si todo ha funcionado correctamente deberemos tener activos al menos los componentes que se muestran en la siguiente captura.



9. Xampp viene con un panel de control mediante el cual podemos administrar los diferentes módulos que se han instalado: Apache, MySQL, Filezilla y Mercury y consignarlos como un servicio para que arranquen con el sistema operativo. Esta utilidad nos permite parar y reiniciar los módulos a voluntad.



Actividades.

UD1 – ACTIVIDAD 1: Obteniendo información de PHP.

TIPO	Investigación
OBJETIVOS	Clarificar conceptos fundamentales.
RECURSOS	Editor de texto y navegador web.
ENUNCIADO DE LA ACTIVIDAD	
Crea una página web que muestre por pantalla lo siguiente:	
PREGUNTA	RESPUESTA
¿Quién inventó PHP?	
¿Qué es PHP?	
¿Qué significa PHP?	
¿Cuáles son las diferencias más notables entre JavaScript y PHP?	
¿Qué tipo de licencia tiene PHP?	
¿Dónde podemos encontrar ayuda para php?	
¿Qué necesitamos para trabajar con PHP?	
¿Cómo accedemos a nuestro servidor local?	
¿Dónde se encuentra la página de inicio de XAMPP?	

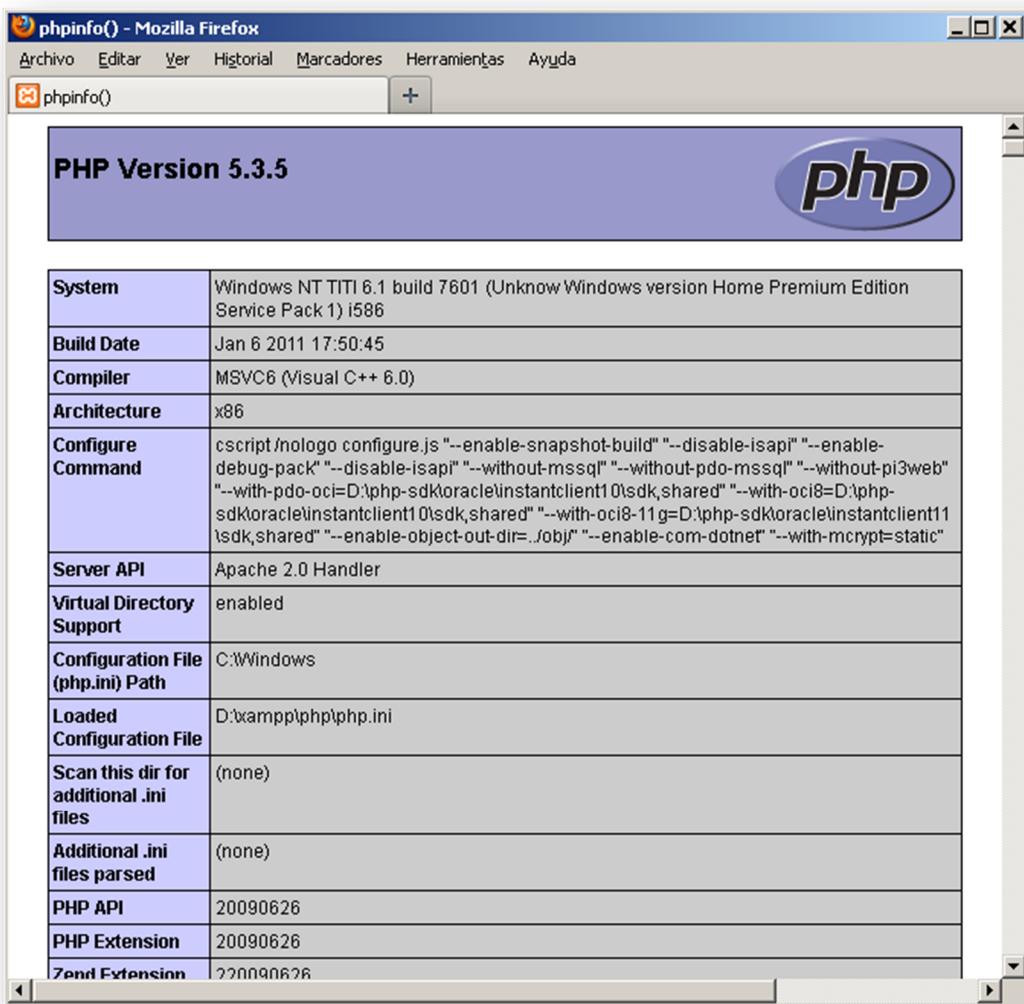
UD1 – ACTIVIDAD 2: Primera página web.

TIPO	Desarrollo
OBJETIVOS	Practicar la creación de páginas php y estudiar la información aportada por <code>phpinfo()</code> .
RECURSOS	Editor de texto y navegador web.

ENUNCIADO DE LA ACTIVIDAD

Crea una primera página php mediante un editor de textos (puedes ver las opciones propuestas en los anexos) y copia el siguiente código para acceder mediante la función `phpinfo()` a la información de nuestra instalación de PHP.

```
<?PHP
    phpinfo();
?>
```



COMENTARIOS

El fichero php deberá situarse en el servidor de aplicaciones, si la instalación se realizó mediante Xampp (dentro de \xampp\htdocs) siguiendo las indicaciones anteriores. Crea una carpeta para las prácticas, una para cada unidad.

Es bueno ir familiarizándose con los diferentes elementos que aparecen con `phpinfo()` porque nos permitirá configurar nuestro sistema.

2 – El lenguaje PHP, conceptos básicos.

En la presente unidad vamos a ver como introducir los conceptos básicos del lenguaje PHP. A pesar de ser el tema más simple, es a su vez el más importante porque el resto se sustentan y recurren a él.

2.1 – Como funciona PHP.

Cuando solicitamos una página HTML con el siguiente código:

```
<HTML>
    <TITLE>Contenido HTML</TITLE>
<BODY>
    Contenido de una página HTML
</BODY>
</HTML>
```

El servidor nos devolverá el contenido del fichero tal cual, es decir, sin realizarle ninguna modificación.

```
<HTML>
    <TITLE>Contenido HTML</TITLE>
<BODY>
    Contenido de una página HTML
</BODY>
</HTML>
```

El resultado gráfico será el siguiente:



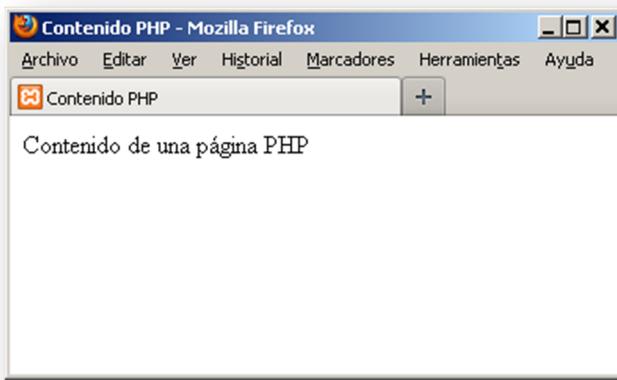
Pero si solicitamos una página PHP con el siguiente código:

```
<HTML>
    <TITLE>Contenido PHP</TITLE>
<BODY>
<?PHP
    print("Contenido de una página PHP");
?>
</BODY></HTML>
```

El servidor nos devolverá el contenido del fichero después de haber interpretado el código PHP.

```
<HTML>
    <TITLE>Contenido PHP</TITLE>
<BODY>
    Contenido de una página PHP
</BODY>
</HTML>
```

El resultado gráfico será el siguiente:



¿Qué ventajas nos aporta esto?

- Podremos hacer páginas dinámicas, en las cuales el contenido variará según nuestras necesidades.
- Siempre enviaremos código HTML al cliente (es decir, al ordenador que ha realizado la petición de la página), de esta forma el explorador del cliente sólo deberá saber interpretar el HTML.

2.2 – Sintaxis básica.

Como ha podido verse anteriormente el código PHP se inserta dentro de una página HTML, pero dicho fichero deberá tener la extensión "PHP" para que el servidor sepa que ha de interpretarla antes de enviarla al cliente.

Podemos indicar que el código que viene a continuación es PHP de varias formas, pero la más común es la siguiente:

```
<?PHP
    código PHP...
?
```

Cualquier cosa fuera del par de etiquetas de apertura y cierre es ignorado por el intérprete de PHP, el cual permite que los ficheros de PHP tengan contenido mixto. Esto permite que PHP sea embebido en documentos HTML para, por ejemplo, crear plantillas.

```
<p>Esto va a ser ignorado por PHP y mostrado por el navegador.</p>
    <?php echo 'Mientras que esto va a ser interpretado.'; ?>
<p>Esto también será ignorado por PHP y mostrado por el navegador.</p>
```

Esto funciona como se espera, porque cuando PHP intercepta las etiquetas de cierre ?, simplemente comienza a imprimir cualquier cosa que encuentre (a excepción

de un una nueva línea inmediatamente después - véase separación de instrucciones) hasta que dé con otra etiqueta de apertura a menos que se encuentre en mitad de una sentencia condicional, en cuyo caso el intérprete determinará el resultado de la condición antes de tomar una decisión de qué es lo que tiene que saltar. A continuación se muestran otras formas de realizar la apertura y cierre de código php.

1. <?php echo 'si se quiere mostrar documentos XHTML o XML, debe hacerse así'; ?>
2. <script language="php">
echo 'algunos editores (como FrontPage) no les gusta
las instrucciones de proceso';
</script>
3. <? echo 'esta es la forma más simple, una instrucción de procesado SGML'; ?>
<?= expresión ?> Esto es una forma abreviada de "<? echo expresión ?>"
4. <% echo 'Quizá use de forma opcional etiquetas de estilo ASP'; %>
<%= \$variable; # Esto es una forma abreviada de "<% echo . . ." %>

Separación de instrucciones.

Como en C o en Perl, PHP requiere que las instrucciones terminen en punto y coma al final de cada sentencia. La etiqueta de cierre de un bloque de código de PHP automáticamente implica un punto y coma; no es necesario usar un punto y coma para cerrar la última línea de un bloque de PHP. La etiqueta de cierre del bloque incluirá la nueva línea final inmediata si está presente.

```
<?php  
    echo 'Esto es una prueba';  
?  
  
<?php echo 'Esto es una prueba' ?>
```

Comentarios.

PHP soporta comentarios 'C', 'C++' y estilo consola Unix (estilo Perl). Por ejemplo:

```
<?php  
    echo 'Esto es una prueba'; // Esto es un comentario estilo c++ de una sola línea  
    /* Esto es un comentario multi-línea  
       y otra línea de comentarios */  
    echo 'Esto es otra prueba';  
    echo 'Una prueba final'; # Esto es un comentario estilo consola de una sola línea  
?>
```

Los comentarios del estilo "una sola línea" solo comentan hasta el final de la línea o del bloque actual de código de PHP, lo primero que suceda. Esto implica que el código HTML después de // ... ?> o # ... ?> SERÁ impreso: ?> sale del modo PHP y vuelve al modo HTML, por lo que // o # no pueden influir en eso. Si la directiva de configuración `asp_tags` está activada, actúa igual que // %> y # %>. Sin embargo, la etiqueta </script> no sale del modo PHP en un comentario de una sola línea.

```
<h1>Esto es un <?php # echo 'simple';?> ejemplo</h1>  
<p>El encabezado anterior dirá 'Esto es un ejemplo'.</p>
```

Mostrando información.

Ya se ha comentado que el PHP es un lenguaje interpretado, que el servidor transforma antes de enviar al ordenador cliente que realizó la petición, por lo que vamos

a ver ahora una de las funciones más importantes que nos permitirá realizar esa transformación.

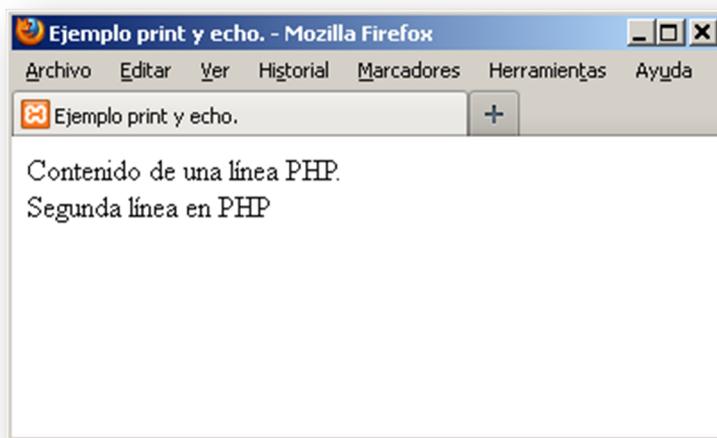
Para ello vamos a usar las órdenes "**echo**" y "**print**" que nos convertirán lo que le indiquemos en cadenas de texto HTML. Obsérvese que se escriben en minúsculas. Veamos el siguiente ejemplo:

```
<HTML>
    <TITLE>Ejemplo print y echo.</TITLE>
<BODY>
<?PHP
    // Ejemplo de la orden "print" y "echo"
    print ("Contenido de una línea PHP. <BR />");
    echo "Segunda línea en PHP.";
?>
</BODY></HTML>
```

Nos devuelve el código siguiente:

```
<HTML>
    <TITLE>Ejemplo print y echo.</TITLE>
<BODY>
    Contenido de una línea PHP. <BR />Segunda línea en PHP.
</BODY></HTML>
```

Cuyo resultado gráfico es:



Si quisiéramos que las líneas del código apareciesen separadas cada una en una línea independiente (con el objetivo de poder leer mejor el código fuente), tendremos que hacer uso del carácter de escape **\n**. Esto hace que termine la línea. El código quedaría como sigue:

```
<HTML>
    <TITLE>Ejemplo print y echo.</TITLE>
<BODY>
<?PHP
    // Ejemplo de la orden "print" y "echo"
    print ("Contenido de una línea PHP. <BR /> \n");
    echo "Segunda línea en PHP.";
?>
</BODY>
</HTML>
```

Pese a que el resultado gráfico sería el mismo, el código fuente queda más claro.

```
<HTML>
    <TITLE>Ejemplo print y echo.</TITLE>
<BODY>
```

```
Contenido de una línea PHP. <BR />
Segunda línea en PHP.
</BODY>
</HTML>
```

Gestión de ficheros.

Cuando tenemos ficheros fuente muy grandes PHP nos ofrece la posibilidad de dividirlos con el objetivo de organizar mejor la información y estructurar nuestro proyecto de forma optima (por ejemplo evitando repetir trozos de código redundantes en nuestro sitio web).

A tal fin, PHP pone a nuestra disposición dos instrucciones capaces de incluir código de ficheros externos en nuestra página actual. Estas son **include** y **require**. La diferencia entra ambas estriba en que si el fichero externo a incluir no se encuentra, include muestra un aviso (warning) pero continua, pero require nos mostrará un error y parará la carga de la página.

La sintaxis para ambas es la misma:

```
include("fichero");
require("fichero");
```

Veamos un ejemplo:

```
<HTML>
<BODY>
<?PHP
    include("cabecera.txt");
?>
</BODY>
</HTML>
```

y el fichero "cabecera.txt" tiene el contenido:

```
echo("Mi sitio web personal");
```

es lo mismo que si tuviéramos la siguiente página PHP:

```
<HTML>
<BODY>
<?PHP
    echo("Mi sitio web personal");
?>
</BODY>
</HTML>
```

2.3 – Tipos de datos.

PHP tiene los siguientes tipos de datos:

- **integer**: Usado para los números enteros. Se puede utilizar la notación decimal (83), octal (0123) o hexadecimal (0x12).
- **float**: Usado para números reales o decimales. El separador decimal es un punto y no una coma.
- **boolean**: Tiene valores lógicos: *true* y *false* (verdadero y falso). El cero es el valor falso, y cualquier otro valor será verdadero. (Internamente los booleanos se almacenan como un entero).
- **NULL**: Es un tipo especial que solo contiene un valor: NULL.
- **resource**: Contiene una referencia a un recurso externo
- **string**: Contienen cadenas de caracteres. Podremos usar comillas simples o comillas dobles para delimitar la cadena, es decir, para decir dónde comienza y dónde termina la cadena. Tienen algunas características interesantes como son:

- Si la cadena está delimitada por comillas dobles, cualquier variable incluida dentro de ella será sustituida por su valor.
- Si la cadena está delimitada por comillas simples, cualquier variable incluida dentro de ella NO será sustituida por su valor.
- Si queremos poner el carácter de la comilla simple dentro de una cadena que está delimitada por la comilla simple deberemos usar el carácter escape, para ello pondremos \`.
- Si queremos poner el carácter escape dentro de una cadena deberemos ponerlo dos veces, indicando la primera vez que aparece que el siguiente carácter (el de escape) se ha de poner tal cual: \\
- Las cadenas con comillas dobles admiten otros caracteres de escape:
 - \n → Nueva línea.
 - \r → Retorno de carro.
 - \t → Tabulación horizontal.
 - \\ → Barra invertida
 - \\$ → Signo del dólar.
 - \" → Comillas dobles.
- **array**: no contiene un único valor, sino un conjunto de valores referenciados con un índice. Podemos crear arrays de dos formas:
 - Estáticamente: creamos el array y le damos valor a cada elemento. Su sintaxis es:


```
$miArray=array([índice1]=>[valor1], [índice2]=>[valor2], ...);
```
 - Dinámicamente: simplemente asignamos valores en las posiciones que queramos y éstas se definen a la vez. Por ejemplo:


```
$otroArray[6]="valor de tipo string en la posición 7ª";
```

 Como puede observarse en el ejemplo anterior, aunque el índice tiene el valor "6" esa posición es la 7ª por que los arrays comienzan en la posición cero.
- También existen los llamados arrays asociativos, en los que el índice no es un número sino una cadena de texto:


```
$semana["primero"]="lunes";
```
- Y por último están los arrays multidimensionales o de varias dimensiones. Supongamos un array en dos dimensiones en el que la primera dimensión nos indicará el código del producto y el la segunda dimensión tendremos sus detalles, como pueden ser la descripción, el precio, etc.


```
$producto[1]["precio"]=23.5;
```
- **object**: Para objetos, formado por un conjunto de datos y funciones independientes.

Gestión de tipos de datos.

El tipo de una variable usualmente no es declarado por el programador; en cambio, es decidido en tiempo de ejecución por PHP dependiendo del contexto en el que es usada la variable.

Para obtener una representación legible para humanos del tipo para propósitos de depuración, use la función [gettype\(\)](#). Para comprobar un cierto tipo, *no use gettype()*, si no las funciones [is_tipo](#). Algunos ejemplos:

```
<?php
$un_bool = TRUE; // un valor booleano
$un_str = "foo"; // una cadena
$un_str2 = 'foo'; // una cadena
$un_int = 12; // un entero

echo gettype($un_bool); // imprime: boolean
echo gettype($un_str); // imprime: string

// Si este valor es un entero, incrementarlo en cuatro
if (is_int($un_int)) {
```

```

$un_int += 4;
}

// Si $a_bool es una cadena, imprimirla
// (no imprime nada)
if (is_string($un_bool)) {
    echo "Cadena: $un_bool";
}
?>
```

Para forzar la conversión de una variable a cierto tipo, puede [moldear](#) la variable o usar la función [settype\(\)](#) sobre ella.

Manipulación de tipos.

PHP no requiere (ni soporta) la definición explícita de tipos en la declaración de variables; el tipo de la variable se determina por el contexto en el cual se emplea la variable. Es decir, si se asigna un valor string a una variable \$var, entonces \$var se convierte en un string. Si un valor integer es entonces asignado a la misma variable \$var, ésta se convierte en integer.

Un ejemplo de la conversión de tipos automática de PHP es el operador suma '+'. Si ambos operandos son float, entonces ambos operandos son evaluados como floats y el resultado será un float. De otra manera, los operandos serán interpretados como integers, y el resultado será entonces integer. Tenga en cuenta que esto no implica que se cambien los tipos de los propios operandos; el único cambio es en como se evalúan los operandos y en el tipo de expresión en sí mismo.

```
<?php
    $foo = "0"; // $foo es string (ASCII 48)
    $foo += 2; // $foo es ahora un integer (2)
    $foo = $foo + 1.3; // $foo es ahora un float (3.3)
    $foo = 5 + "10 Cerditos pequeñitos"; // $foo es integer (15)
    $foo = 5 + "10 Cerdos pequeños"; // $foo es integer (15)
?>
```

Forzado de tipos.

El forzado de tipos en PHP funciona de la misma manera que en C:, donde el nombre del tipo deseado se escribe entre paréntesis antes de la variable que se quiera forzar.

```
<?php
    $foo = 10; // $foo es un integer
    $bar = (boolean) $foo; // $bar es un boolean
?>
```

Los siguientes forzados de tipos están permitidos:

- (int), (integer) - forzado a integer
- (bool), (boolean) - forzado a boolean
- (float), (double), (real) - forzado a float
- (string) - forzado a string
- (array) - forzado a array
- (object) - forzado a object
- (unset) - forzado a NULL (PHP 5)

2.4 – Variables.

Como en todos los lenguajes de programación, PHP permite almacenar datos de distintos tipos en memoria. Estas zonas de memoria se llaman variables. Las variables

comienzan por el símbolo de dólar (\$) y no necesitan ser declaradas antes de comenzar el programa, como en otros lenguajes.

Los nombres de variables siguen las mismas reglas que otras etiquetas en PHP. Un nombre de variable válido tiene que empezar con una letra o un carácter de subrayado (underscore), seguido de cualquier número de letras, números y caracteres de subrayado. Como expresión regular se podría expresar como: '[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*'

El nombre de la variable es sensible a minúsculas y mayúsculas por lo que las siguientes variables son distintas: \$hola ≠ \$Hola ≠ \$HOLA.

De forma predeterminada, las variables siempre se asignan **por valor** (mediante el símbolo “=”). Esto significa que cuando se asigna una expresión a una variable, el valor completo de la expresión original se copia en la variable de destino. Esto quiere decir que, por ejemplo, después de asignar el valor de una variable a otra, los cambios que se efectúen a una de esas variables no afectará a la otra.

```
<?php
    $nombre = 'Roberto';
    $apellidos = 'Martínez Rubio';
    echo "$nombre $apellidos";           // imprime "Roberto Martínez Rubio"
?>
```

PHP también ofrece otra forma de asignar valores a las variables: asignar **por referencia**. Esto significa que la nueva variable simplemente referencia (en otras palabras, "se convierte en un alias de" ó "apunta a") la variable original. Los cambios a la nueva variable afectan a la original, y viceversa.

Para asignar por referencia, simplemente se antepone un signo ampersand (&) al comienzo de la variable cuyo valor se está asignando (la variable fuente). Por ejemplo, el siguiente segmento de código produce la salida '*Mi nombre es Bob*' dos veces:

```
<?php
    $texto = 'Roberto';                // Asigna el valor Roberto
    $nombre = &$texto;                 // referenciar a $texto vía $nombre
    echo "$nombre ";                  // imprime "Roberto"
    $texto = "Luís";                  // Se modifica el valor de $texto por "Luís"
    echo "$nombre ";                  // imprime "Luís" porque $nombre apunta al valor de $texto
?>
```

Para terminar con las variables diremos que podemos crear "**variables variables**", es decir, variables cuyo nombre es variable: la variable tomará su nombre de otra variable que haya sido declarada previamente. Veámoslo con un ejemplo:

```
<?PHP
    $a="buenos";
    $$a="dias";
    echo("$a $buenos <BR>");
    echo("$a ${$a}");
?>
```

cuya salida será la siguiente:

```
buenos dias<BR>buenos dias
```

y el explorador nos mostrará:

```
buenos dias
buenos dias
```

Manejando variables.

PHP brinda al programador una serie de funciones para el manejo de variables.

isset() → Con esta función podemos averiguar si una función existe dentro de nuestro programa. Si existe devuelve *true* y si no existe *false*.

```
<?php
$DNI = "23211133-G";
if (isset($DNI)) {
    echo ("La variable DNI existe!!!");
}
?>
```

unset() → Libera la memoria ocupada por una variable, destruyendo su nombre y su contenido. Después de usar *unset ()*, la variable destruida aparecerá como *false* al utilizar la función *isset()*.

```
<?php
$Nombre = "María Trias";
if (isset($Nombre)) {
    echo ("El nombre existe!!!");
}
//Podemos comprobar qué pasa si liberamos la variable $Nombre
unset($Nombre);
if (isset($Nombre)) {
    echo ("El nombre existe!!!");
}
else {
    echo ("El nombre ya no existe!!!");
}
?>
```

El resultado es el siguiente:

```
El nombre existe!!!
El nombre ya no existe!!!
```

empty() → Comprueba si una variable está vacía, no existe, o su valor es 0.

```
<?php
$correo = "prueba@gmail.com";
if (empty($correo)) {
    echo ("La variable correo no existe");
}
$numero_entero = 0 ;
if (empty($numero_entero)) {
    echo ("La variable numero_entero no existe o tiene el valor 0");
}
?>
```

Variables predefinidas.

PHP proporciona una gran cantidad de variables predefinidas para todos los scripts. Las variables representan de todo, desde **variables externas** hasta variables de entorno incorporadas, desde los últimos mensajes de error hasta los últimos

encabezados recuperados. A continuación se muestra un listado de variables predefinidas que se explicarán más adelante.

- **Superglobals** — Superglobals son variables internas que están disponibles siempre en todos los ámbitos
- **\$GLOBALS** — Hace referencia a todas las variables disponibles en el ámbito global
- **\$_SERVER** — Información del entorno del servidor y de ejecución
- **\$_GET** — Variables HTTP GET
- **\$_POST** — Variables HTTP POST
- **\$_FILES** — Variables de Carga de Archivos HTTP
- **\$_REQUEST** — Variables HTTP Request
- **\$_SESSION** — Variables de sesión
- **\$_ENV** — Variables de entorno
- **\$_COOKIE** — Cookies HTTP
- **\$php_errormsg** — El último mensaje de error
- **\$HTTP_RAW_POST_DATA** — Datos POST sin tratar
- **\$http_response_header** — Encabezados de respuesta HTTP
- **\$argc** — El número de argumentos pasados a un script
- **\$argv** — Array de argumentos pasados a un script

2.5 – Constantes.

Una constante es un identificador (nombre) para expresar un valor simple. Como el nombre sugiere, este valor no puede variar durante la ejecución del script. (A excepción de las **constantes predefinidas**, que en realidad no son constantes). Una constante es sensible a mayúsculas por defecto. Por convención, los identificadores de constantes siempre suelen declararse en mayúsculas.

El nombre de una constante sigue las mismas reglas que cualquier otra etiqueta de PHP. Un nombre de constante válido empieza por una letra o subguion, seguido por cualquier número o letras, números o subguiones.

```
<?php
// Nombre de constantes correctos
define("FOO", "something");
define("FOO2", "something else");
define("FOO_BAR", "something more");

// Nombres de constantes incorrectos
define("2FOO", "something");

// Esto es válido, pero debería ser evitado:
// Ya que quizás algún día PHP crea una constante mágica
// con el mismo nombre y en ese caso provocaría un error en tu script
define("__FOO__", "something");
?
```

Constantes predefinidas.

PHP ofrece un largo número de **constantes predefinidas** a cualquier script en ejecución. Muchas de estas constantes, sin embargo, son creadas por diferentes extensiones, y sólo estarán presentes si dichas extensiones están disponibles, bien por carga dinámica o porque han sido compiladas.

Hay ocho constantes predefinidas que cambian dependiendo de donde son usadas. Por ejemplo el valor de `__LINE__` depende en la línea que se use en el script. Estas constantes especiales son sensibles a mayúsculas y son las siguientes:

Nombre	Descripción
<code>__LINE__</code>	Línea actual en el fichero.
<code>__FILE__</code>	Ruta completa y nombre del fichero. Si se usa dentro de un include, devolverá el nombre del fichero del include. Desde PHP 4.0.2, <code>__FILE__</code> siempre contiene la ruta absoluta con symlinks resueltos, en otras versiones contenía la ruta relativa según las circunstancias.
<code>__DIR__</code>	Directorio del fichero. Si se utiliza dentro de un include, devolverá el directorio del fichero incluído. Esta constante es igual que <code>dirname(__FILE__)</code> . El nombre del directorio no lleva la barra inicial a no ser que esté en el directorio root. (Fue añadida en PHP 5.3.0)
<code>__FUNCTION__</code>	Nombre de la función. Desde PHP 5 esta constante devuelve el nombre de la función donde fue declarada (sensible a mayúsculas). En PHP 4 su valor siempre es en minúsculas.
<code>__CLASS__</code>	Nombre de la clase. (Añadida en PHP 4.3.0) Desde PHP 5 esta constante devuelve el nombre de la clase donde fue declarada (sensible a mayúsculas). En PHP 4 su valor siempre es en minúsculas. El nombre de la clase incluye el namespace declarado en (p.e.j. <code>Foo\Bar</code>). Tenga en cuenta que a partir de PHP 5.4 <code>__CLASS__</code> también funciona con traits. Cuando es usado en un método trait, <code>__CLASS__</code> es el nombre de la clase del trait que está siendo utilizado.
<code>__TRAIT__</code>	El nombre del trait. (Añadido en PHP 5.4.0) A partir de PHP 5.4 esta constante devuelve el trait que fué declarado (sensible a mayúsculas y minúsculas). El nombre de el trait incluye el namespace si alguno fué declarado en (p.e.j. <code>Foo\Bar</code>).
<code>__METHOD__</code>	Nombre del método de la clase. (Añadida en PHP 5.0.0.) Nombre del método devuelto donde fue declarada. (sensible a mayúsculas).
<code>__NAMESPACE__</code>	Nombre del espacio de nombres actual (sensible a mayúsculas). Esta constante se define en tiempo de compilación (Añadida en PHP 5.3.0) El nombre del namespace actual (sensible a mayúsculas).

2.6 – Expresiones y operadores.

En PHP una **expresión** es cualquier cosa que pueda contener un valor. Las expresiones más simples son las variables y las constantes y otras más complicadas serán las funciones, puesto que cada función devuelve un valor al ser invocada, es decir, contiene un valor, por lo tanto, es una expresión.

Un **operador** es algo que toma uno más valores (o expresiones, en jerga de programación) y produce otro valor (de modo que la construcción en si misma se convierte en una expresión).

Los operadores se pueden agrupar de acuerdo con el número de valores que toman. Los operadores unarios toman sólo un valor, por ejemplo `!` (el **operador lógico de negación**) o `++` (el **operador de incremento**). Los operadores binarios toman dos valores, como los familiares **operadores aritméticos** `+` (suma) y `-` (resta), y la mayoría de los operadores de PHP entran en esta categoría. Finalmente, hay sólo un **operador ternario**, `? :`, el cual toma tres valores; usualmente a este se le refiere simplemente como

"el operador ternario" (aunque podría tal vez llamarse más correctamente como el operador condicional).

Una lista completa de operadores de PHP sigue en la sección [Precedencia de Operadores](#). La sección también explica la precedencia y asociatividad de los operadores, las cuales gobiernan exactamente cómo son evaluadas expresiones que contienen varios diferentes operadores.

Operador de asignación.

El más básico es el símbolo de asignación (=), utilizado para dar valores a las variables que usamos en nuestro código. Las variables que están a la izquierda del operador toman el valor que se encuentra en la expresión de la derecha.

```
<?php
$variable = 34;
$variable2 = "Asignación de valores";
?>
```

Operador unario.

El signo menos (-) se utiliza delante de un número o variable numérica. Este operador tiene la propiedad de hacer a los números, negativos o positivos, dependiendo del signo actual.

```
<?php
$entero = 23;
$entero_negativo = -$entero; // El valor es ahora -23
entero2 = -$entero__negativo; // El valor cambia ahora a 23
?>
```

Operadores aritméticos.

Este tipo de operadores forman parte de la aritmética básica. Nos resultará familiar porque son símbolos muy utilizados en el aprendizaje de las matemáticas.

Operador	Sintaxis	A	B	Resultado
Suma (+)	$\$a + \b	12	-7.3	4.7
Diferencia (-)	$\$a - \b	12	-7.3	19.3
Producto (*)	$\$a * \b	12	-7.3	-87.6
Cociente (/)	$\$a / \b	12	-7.3	-1.6438356164384
Módulo (%)	$\$a \% \b	12	5	2
Negación (-)	$-\$a$	4		-4

Operadores de comparación.

En algunos ejemplos del capítulo anterior puede ver que se utiliza la estructura de control if else. Como veremos más adelante, esta estructura compara dos valores y elige el camino a seguir. El valor de la comparación siempre es **true** o **false**.

PHP dispone de los siguientes operadores de comparación:

\$A == \$B El operador **==** **compara** los valores de dos variables y devuelve **1(CIERTO)** en el caso de que sean iguales y el valor **NUL** –carácter ASCII 0– (**FALSO**) cuando son distintas. Mediante este operador se pueden comparar variables de distinto tipo.

Para comparar una *cadena* con un *número* se extrae el *valor entero de la cadena* (si lleva dígitos al comienzo los extrae y en caso contrario le asigna el valor cero) y utiliza ese valor para hacer la comparación.

Cuando se comparan cadenas *discrimina* entre *mayúsculas y minúsculas* ya que utiliza los códigos ASCII de cada uno de los caracteres para hacer la comparación. La comparación se hace de izquierda a derecha y devuelve **1 (CIERTO)** sólo en el caso que coincidan exactamente los contenidos de ambas cadenas.

\$A === \$B El operador **==** es similar al anterior, pero realiza la comparación en sentido **estricto**. Para que devuelva **1** es necesario que sean iguales *los valores de las variables y también su tipo*.

\$A != \$B El operador **!=** devuelve **1** cuando los valores de las variables **son distintos** (en general **!** indica negación, en este caso podríamos leer «*no igual*») y devuelve **NUL** cuando son iguales. Este operador **no compara** en sentido **estricto**, por lo que puede considerar iguales los valores de dos variables de distinto tipo.

\$A < \$B El operador **<** devuelve **1** cuando los valores de **\$A** son **menores** que los de **\$B**. Los criterios de comparación son los siguientes:

- Los *valores numéricos* siguen el criterio matemático.
- Cuando se trata de un *número* y una *cadena* extrae el valor numérico de ésta (es cero si no hay ningún dígito al principio de la misma) y hace una comparación matemática.
- En el supuesto de dos cadenas, compara **uno a uno** –de izquierda a derecha– los códigos ASCII de cada uno de los caracteres (primero con primero, segundo con segundo, etcétera). Si al hacer esta comprobación encuentra –en la primera cadena– un carácter cuyo código ASCII es mayor que el correspondiente de la segunda cadena, o encuentra que todos son iguales en ambas cadenas devuelve NUL. Solo en el caso de no existir ninguno mayor y sí haber al menos uno menor devolverá UNO.
- Cuando las cadenas tengan distinta longitud, considerará (a efectos de la comparación) que los caracteres *que faltan* en la cadena más corta son NUL (ASCII 0).

\$A <= \$B Se comporta de forma idéntica al anterior. La única diferencia es que ahora aceptará como *ciertos* los casos de *igualdad* tanto en el caso de números como en el de códigos ASCII.

\$A > \$B Es idéntico –en el modo de funcionamiento– a **\$A < \$B**. Solo difiere de éste en el *criterio de comparación* que ahora requerirá que los valores de **\$A** sean *mayores* que los de la variable **\$B**.

\$A >= \$B Añade al anterior la posibilidad de certeza en caso de igualdad.

Operador	Sintaxis	A	B	Resultado
Igualdad (==)	\$a==\$b	12	-7.3	false
Identidad (==)	\$a===\$b	12	4	false
Distinto (<>)	\$a<>\$b	12	-7.3	true
Distinto (!=)	\$a!=\$b	12	12	false

Menor que (<)	\$a < \$b	12	5	false
Menor o igual que (<=)	\$a <= \$b	4	4	true
Mayor que (>)	\$a > \$b	12	5	true
Mayor o igual que (>=)	\$a >= \$b	11	12	false

```
<?php
$a = 23; // Asignación de los valores
$b = 75;
if ($a >= $b) { //La condición no se cumple. El resultado es false
    echo "Esta parte no se ejecuta";
} else {
    echo "La comparación es false porque $a es menor que $b";
}
?>
```

Operadores lógicos

Durante el desarrollo de su proyecto, puede encontrarse con situaciones en las que necesite hacer varias comparaciones seguidas para que se cumpla una determinada condición. PHP permite unir todas las comparaciones en una mediante el uso de los operadores lógicos.

Mediante operadores lógicos es posible *evaluar* un conjunto de variables lógicas, es decir, aquellas cuyos valores sean únicamente: VERDADERO o FALSO (**1** ó **NUL**). El resultado de esa evaluación será siempre **1** ó **NUL**.

\$A AND \$B El operador **AND** devuelve VERDADERO (**1**) en el caso de que **todas** las variables lógicas comparadas sean verdaderas, y FALSO (**NUL**) cuando **alguna** de ellas sea *falsa*.

\$A && \$B El operador **&&** se comporta de forma idéntica al operador **AND**. La única diferencia entre ambos es que *operan con distinta precedencia*.

\$A OR \$B Para que el operador **OR** devuelva VERDADERO (**1**) es suficiente que **una sola** de las variables lógicas comparadas sea **verdadera**. Únicamente devolverá FALSO (**NUL**) cuando **todas** ellas sean FALSAS.

\$A || \$B El operador **||** se comporta de forma idéntica al operador **OR**. Su única diferencia es el orden de precedencia con el que opera.

\$A XOR \$B El operador **XOR** devuelve VERDADERO (**1**) sólo en el caso de que sea *cierta una sola* de las variables, y FALSO (**NUL**) cuando ambas sean ciertas o ambas sean falsas.

! \$A Este operador NOT (negación) devuelve VERDADERO (**1**) si la variable lógica **\$A** es FALSA y devuelve FALSO (**NUL**) si el valor de esa variable **\$A** es VERDADERO.

Sintaxis alternativa

Tal como hemos descrito los distintos operadores lógicos sería necesario que **\$A** y **\$B** contuvieran valores lógicos, y eso requeriría un paso previo para asignarles valores de ese tipo. Habría que recurrir a procesos de este tipo:

```
$A = $x>$y;
$B= $x >=$z;
$A && $B;
```

pero se obtendría el mismo resultado escribiendo: `$x>$y && $x >=$z`; que, aparte de ser la forma habitual de hacerlo, *nos evita dos líneas de instrucciones*.

Aunque el propio ejemplo se auto comenta, digamos que al utilizar operadores lógicos se pueden sustituir las variables lógicas por expresiones que den como resultado ese tipo de valores.

Operador	Sintaxis	A	B	C	Resultado
Y (and)	<code>\$a>\$b and \$c>\$b</code>	12	-7.3	13	true.
O (or)	<code>\$a>\$b or \$b>\$c</code>	12	-7.3	2	true
O exclusivo (xor)	<code>\$a>\$b xor \$c>\$b</code>	12	-7.3	13	false
Negación (!)	<code>!\$a</code>	true			false
Y (&&)	<code>\$a>\$b and \$c>\$b</code>	12	-7.3	13	true
O ()	<code>\$a>\$b or \$b>\$c</code>	12	-7.3	2	true

```
< ?php
    $a = 2 3;
    $b = 75;
    $c = true;
    if ($a < $b and $c) {
        echo "Se cumplen las dos condiciones";
    }
?>
```

Operador ternario

Los operadores que hemos visto hasta ahora son capaces de manejar un operando (Unarios) o dos operandos (binarios). El operador ternario, o de comparación, evalúa un operando y, dependiendo de si es falso o verdadero, evalúa el segundo operando o el tercero. La expresión que se quiere evaluar se escribe delante de un símbolo (?), después la expresión que tiene que ejecutarse si la evaluación anterior es true, seguida del símbolo (:) con la expresión que debe ejecutarse si es false.

```
<?php
    $valor = false;
    $valor == true ? $resultado = "OK" : $resultado = "FALLO";
    // Si $value es true $resultado será OK
    // Si es false $re sultado será FALLO
    echo $resultado;
?>
```

Operadores bit a bit

Estos operadores son complicados de entender si no conoce la lógica binaria. Afortunadamente, se utilizan en muy pocas ocasiones. Los operadores de bit utilizan las variables a nivel bajo, tal y como se almacenan en memoria física y comparan bit a bit los valores. Lo mejor es verlo con un ejemplo.

```
<?php
$a = 4 ; // Valor binario 100
$b = 5; // Valor binario 101
$c = $a & $b;
echo $c; // El valor de c es 100
?>
```

El operador binario Y (símbolo `&`) compara bit a bit las variables `$a` y `$b`. Si los bits de una misma posición son true (tienen el valor 1), el bit resultado es 1. En este caso, sólo existe una pareja de bits que es igual a true (sus dos valores son 1), por lo tanto el valor de la variable `$c` es 4 (en binario 100).

Operador	Sintaxis	Resultado
Y (<code>&</code>)	<code>\$a&\$b</code>	Si las parejas de bits son verdaderas el resultado es verdadero.
O (<code> </code>)	<code>\$a \$b</code>	Si algún bit de la pareja es verdadero el resultado es verdadero.
O exclusiva (<code>^</code>)	<code>\$a^\$b</code>	Si un bit de la pareja es true y el otro false el resultado es verdadero.
NO (<code>~</code>)	<code>\$a~\$b</code>	Los bits 1 se vuelven 0 y viceversa. También el bit que se refiere al signo + o -.
Desplazamiento a la izquierda de bits (<code><<</code>)	<code>\$a<<\$b</code>	Desplaza a la izquierda los bits de la variable <code>\$a</code> tantos bits como indique la variable <code>\$b</code> .
Desplazamiento a la derecha de bits (<code>>></code>)	<code>\$a>>\$b</code>	Desplaza a la derecha los bits de la variable <code>\$a</code> tantos bits como indique la variable <code>\$b</code> .

Operadores de asignación combinados.

En numerosas ocasiones se nos presentan situaciones en las que una variable debe incrementar o disminuir su valor en 1. PHP provee operadores combinados que permiten asignar rápidamente incrementos de valor, concatenaciones de caracteres, etcétera.

Ejemplo	Nombre	Equivalencia
<code>\$a++</code>	Incremento	<code>\$a = \$a + 1</code>
<code>\$a~</code>	Decremento	<code>\$a = \$a - 1</code>
<code>+\$a</code>	Incremento	<code>\$a = \$a + 1</code>
<code>-\$a</code>	Decremento	<code>\$a = \$a - 1</code>
<code>\$a += \$b</code>	Suma	<code>\$a = \$a + \$b</code>
<code>\$a -= \$b</code>	Resta	<code>\$a = \$a - \$b</code>
<code>\$a *= \$b</code>	Multiplicación	<code>\$a = \$a * \$b</code>
<code>\$a /= \$b</code>	División	<code>\$a = \$a / \$b</code>
<code>\$a %= \$b</code>	Módulo	<code>\$a = \$a % \$b</code>
<code>\$a &= \$b</code>	Y	<code>\$a = \$a & \$b</code>
<code>\$a = \$b</code>	O	<code>\$a = \$a \$b</code>
<code>\$a ^= \$b</code>	O exclusiva	<code>\$a = \$a ^ \$b</code>
<code>\$a .= \$b</code>	Concatenación	<code>\$a = \$a . \$b</code>
<code>\$a >>= \$b</code>	Desplazamiento a la derecha	<code>\$a = \$a >> \$b</code>
<code>\$a <<= \$b</code>	Desplazamiento a la izquierda	<code>\$a = \$a << \$b</code>

Operador de ejecución.

Si tiene experiencia con la programación en shell de Unix o gnuLinux, sabrá que el apostrofe invertido sirve para ejecutar comandos del sistema. PHP ha adoptado esta nomenclatura y funciona exactamente igual.

```
<?php
    $listado_archivos = "ls -la~; //Hacemos un listado de los
    ficheros del directorio actual
    echo $listado_archivos; // y lo sacamos por pantalla
?>
```

Operador de supresión de errores.

La mayoría de las funciones que se utilizan en PHP muestran errores en el navegador cuando algo falla. Si intenta abrir un fichero que no existe, PHP mostrará un mensaje de error y continuará la ejecución del programa. El operador (@), colocado delante de una función, evitará que se muestre el error.

```
<?php
    $fichero = fopen("prueba.txt", "r"); //Muestra un error en el navegador
    echo ("El programa sigue ejecutándose");
?>
```

El código siguiente oculta la salida de error por pantalla:

```
<?php
    $fichero = @fopen("prueba.txt", "r");
    echo ("El programa sigue ejecutándose");
?>
```

Precedencia de operadores.

La precedencia de un operador indica qué tan "estrechamente" se unen dos expresiones juntas. Por ejemplo, en la expresión $1 + 5 * 3$, la respuesta es 16 y no 18 porque el operador de multiplicación ("*") tiene una precedencia mayor que el operador de adición ("+"). Los paréntesis pueden ser usados para forzar la precedencia, si es necesario. Por ejemplo: $(1 + 5) * 3$ se evalúa como 18.

Cuando los operadores tienen la misma precedencia, su asociatividad decide si se evalúan a partir de la derecha o desde la izquierda - ver más abajo los ejemplos.

La siguiente tabla lista en orden la precedencia de los operadores, con los operadores de mayor precedencia en la parte superior. Los operadores en la misma línea tienen la misma precedencia, en cuyo caso su asociatividad decide cuál es el orden de evaluación.

Asociatividad	Operadores	Información adicional
no asociativo	clone new	clone and new
izquierda	[array()
derecha	++ -- ~ (int) (float) (string) (array) (object) (bool) @	tipos e incremento/decremento
no asociativo	instanceof	tipos

derecha	!	lógico
izquierda	* / %	aritmética
izquierda	+ - .	aritmética y string
izquierda	<< >>	bit a bit
no asociativo	< <= > >=	comparación
no asociativo	== != === !== <>	comparación
izquierda	&	bit a bit y referencias
izquierda	^	bit a bit
izquierda		bit a bit
izquierda	&&	lógico
izquierda		lógico
izquierda	? :	ternario
derecha	= += -= *= /= .= %= &= = ^= <<= >>= ==>	asignación
izquierda	and	lógico
izquierda	xor	lógico
izquierda	or	lógico
izquierda	,	muchos usos

Para operadores de igual precedencia, asociatividad izquierda significa que la evaluación procede de la izquierda a la derecha y asociatividad derecha significa lo opuesto. Los operadores de igual precedencia que no son asociativos podrían no asociarse con sí mismos. Por ejemplo, la sentencia `1 < 2 > 1`, es ilegal en PHP; mientras que la sentencia `1 <= 1 == 1` no lo es, ya que el operador T_IS_EQUAL == tiene menos precedencia que el operador T_IS_SMALLER_OR_EQUAL <=.

```
<?php
    $a = 3 * 3 % 5; // (3 * 3) % 5 = 4
    $a = true ? 0 : true ? 1 : 2; // (true ? 0 : true) ? 1 : 2 = 2

    $a = 1;
    $b = 2;
    $a = $b += 3; // $a = ($b += 3) -> $a = 5, $b = 5

    // mezclar ++ y + produce un comportamiento indefinido
    $a = 1;
    echo ++$a + $a++; // puede mostrar 4 o 5
?>
```

El uso de paréntesis, incluso cuando no es estrictamente necesario, a menudo puede mejorar la legibilidad del código.

2.7 – Expresiones y operadores.

Ya conocemos algunas de las funciones que PHP utiliza para mostrar información –salidas– en la ventana del navegador del cliente. Nos referimos a las funciones **print** y **echo**.

Salidas con formato.

Ni la función echo, ni tampoco print permiten establecer una presentación (formato) en sus salidas, excepto que alguna de las variables que se use contenga el resultado de una función number_format.

La función printf() ofrece un gran número de posibilidades en este sentido. Tanto la sintaxis como los valores de los diferentes parámetros –cuando se trate de presentar números– las tienes resumidas aquí a la derecha.

En la página siguiente veremos el uso de printf() para el tratamiento de variables tipo cadena.

printf (cadena de formato,variable1,variable2,..)				
Cadena de formato "%[relleno][alineación][ancho][precisión][tipo]"				
	Carácter	Valor	Sintaxis	Resultado
Relleno	0	0	printf("%020d",32)	0000000000000000000032
	*	*	printf("%*20d",32)	*****32
	espacio ¹⁾	'	printf("%' 20d",32)	32
	-	'-	printf("%'-20d",32)	-----32
En este apartado prestaremos atención únicamente a los caracteres marcados en rojo, que son los que corresponden a las diferentes formas de relleno. Los demás parámetros los iremos tratando uno en los apartados siguientes. Cuando se pretende rellenar con ceros –a la izquierda– basta escribir el 0 inmediatamente detrás del signo %. Si se trata de llenar con un carácter distinto de cero debe escribirse inmediatamente después de una comilla simple ' seguida del carácter de relleno. Si se pretende llenar con espacios forzados se puede escribir la comilla simple 'inmediatamente después teclear la combinación ALT+0160 (carácter ASCII 160) usando el teclado numérico. Aunque obviamente no se visualiza el espacio si se conserva tal como puede verse en el ejemplo ¹⁾ . Obsérvese que como la tipografía es de ancho variable y que según el carácter que se use como relleno se modifica el ancho de la presentación. Quizá convenga recordar que 32 es en este caso la variable a la que pretendemos dar formato y que ese valor podría ser sustituido por el nombre de una variable que contenga valores numéricos.				
Alineación	Carácter	Valor	Sintaxis	Resultado
	Ninguno	Dcha	printf("%020d",32)	0000000000000000000032
	-	Izda	printf("%-20d",32)	32
	Ninguno	Dcha	printf("%*20d",32)	*****32
	-	Izda	printf("%'*-20d",32)	32*****
	Ninguno	Dcha	printf("%020s",32)	0000000000000000000032
	-	Izda	printf("%-20s",32)	32000000000000000000
	Ninguno	Dcha	printf("%*20s",32)	*****32
Ancho	-	Izda	printf("%'*-20s",32)	32*****
	En los casos en que figura Ninguno en la columna Carácter tratamos de indicar que no es necesario escribir nada en la cadena de formato. Cuando aparece un signo (-) estamos indicando que debe insertarse un signo menos. Fíjate que en los cuatro primeros supuestos el identificador de tipo es d, lo cual hace que considere la variable como numérica, mientras que en los cuatro últimos ese valor es s, con lo cual considera la variable como tipo cadena. Cuando tratamos de llenar una variable numérica con ceros por la derecha PHP los omite para no alterar el valor numérico en la presentación. Con cualquier otro carácter de relleno (incluidos los caracteres numéricos con ' delante) si efectúa el relleno.			
	Carácter	Valor	Sintaxis	Resultado
	Entero	14	printf("%*14d",32)	*****32
	Entero	17	printf("%*17d",32)	32*****
	Decimal	14.5	printf("%*14.5d",32)	*****32
	Decimal	17.8	printf("%*17.8d",32)	32*****
Tipo	Decimal	14.5	printf("%*14.5f",32)	*****32.00000
	Decimal	11.8	printf("%*11.8f",32)	32.00000000
El ancho (nº de caracteres totales) puede especificarse mediante un número entero para todo tipo de variables. Si se expresa mediante un número decimal y la variable es tipo coma flotante la parte decimal indica la precisión (nº de cifras decimales) y la parte entera el ancho como número de caracteres de la parte entera o de la parte decimal, según se rellene a la derecha o a la izquierda.				
Tipo	Tipo	Valor	Sintaxis	Resultado

Presentación en forma binaria	b	printf("%*14b",17)	*****10001
Carácter en código ASCII	c	printf("%*14c",97)	a
Número como entero	d	printf("%*14d",17.83)	*****17
Número con decimales	f	printf("%*14f",17.45)	****17.450000
Presentación en forma octal	o	printf("%*14o",17)	*****21
Presentación en hexadecimal	x	printf("%*14x",170)	*****aa
Presentación en hexadecimal	X	printf("%*14X",170)	*****AA
Presentación como >cadena	s	printf("%*14s",170)	*****170

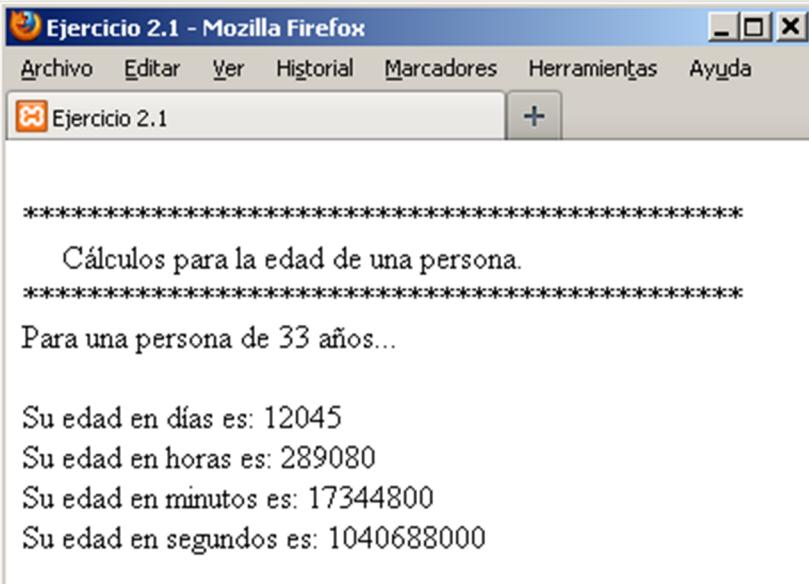
Formatos en cadenas.

printf(cadena de formato,variable1,variable2,...)
<i>Cadena de formato</i>
Dentro de la cadena de formatos deben repetirse tantos formatos como variables se pretenda manejar <code>"%[rell1][alin1][anc1][prec1][tipo1][sepa1] %[rell1][alin1][anc1][prec1][tipo1][sepa1]"</code>
Hemos de mencionar aquí los separadores ya que no fueron mencionados en la página anterior Se puede introducir una cadena de separación al final de una cadena de formato que puede hacer, entre otras, función de separación entre dos cadenas. Por ejemplo, <code>printf("%*15.2f Euros",1475.875)</code> nos devolvería: <code>*****1475.88 Euros</code> La función printf() permite presentar varios valores o variables con distintos formatos utilizando la sintaxis que se indica más arriba. Este ejemplo : <code>printf("%*15.2f Euros=%*18.0f Pesetas",1475.875,1475.875*166.386)</code> devuelve como resultado: <code>*****1475.88 Euros=*****245565 Pesetas</code>

Existe otra función PHP con características muy similares a la anterior. Se trata de **sprintf()**. La sintaxis es idéntica `sprintf (cadena de formato, variable1,variable2, ...)` y su única diferencia con `printf` es que, mientras que `printf()` imprime las variables utilizando el formato indicado, `sprintf()` puede guardar en una nueva variable la cadena resultante de la aplicación del formato.

Actividades.

UD2 – ACTIVIDAD 1: Cálculo de la edad en días, horas, minutos y segundos.

TIPO	Desarrollo
OBJETIVOS	Ejercitarse con los elementos básicos de PHP. Declaración y uso de variables, definición de constantes. Salida de datos formateada.
ENUNCIADO DE LA ACTIVIDAD	
<p>Diseñar un script php que tenga en una variable la edad de una persona y calcule su equivalente en días, horas, minutos y segundos. El resultado obtenido deberá mostrarlo por pantalla debidamente formateado. Una vez ejecutado debe aparecer algo parecido a lo siguiente:</p>  <pre> ***** Cálculos para la edad de una persona. ***** Para una persona de 33 años... Su edad en días es: 12045 Su edad en horas es: 289080 Su edad en minutos es: 17344800 Su edad en segundos es: 1040688000 </pre>	

COMENTARIOS

Es un ejemplo muy sencillo de un programa en PHP. Se recomienda el uso de constantes para definir cuantos días tiene un año, horas un dia, etc.

UD2 – ACTIVIDAD 2: Gestión de tipos.

TIPO	Desarrollo															
OBJETIVOS	Practicar la gestión de tipos para entender mejor como funcionan.															
RECURSOS	Editor de texto y navegador web.															
ENUNCIADO DE LA ACTIVIDAD																
<p>Crea una tabla como la que sigue para probar el funcionamiento de la función gettype()</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th colspan="3">Ejemplos de determinación del tipo de una variable</th> </tr> <tr> <th>Variable</th> <th>Sintaxis</th> <th>Devuelve</th> </tr> </thead> <tbody> <tr> <td>\$a1=347</td> <td>echo gettype(\$a1)</td> <td>integer</td> </tr> <tr> <td>\$a2=2147483647</td> <td>echo gettype(\$a2)</td> <td>integer</td> </tr> <tr> <td>\$a3=-2147483647</td> <td>echo gettype(\$a3)</td> <td>integer</td> </tr> </tbody> </table>		Ejemplos de determinación del tipo de una variable			Variable	Sintaxis	Devuelve	\$a1=347	echo gettype(\$a1)	integer	\$a2=2147483647	echo gettype(\$a2)	integer	\$a3=-2147483647	echo gettype(\$a3)	integer
Ejemplos de determinación del tipo de una variable																
Variable	Sintaxis	Devuelve														
\$a1=347	echo gettype(\$a1)	integer														
\$a2=2147483647	echo gettype(\$a2)	integer														
\$a3=-2147483647	echo gettype(\$a3)	integer														

<code>\$a4=23.7678</code>	<code>echo gettype(\$a4)</code>	<code>double</code>
<code>\$a5=3.1416</code>	<code>echo gettype(\$a5)</code>	<code>double</code>
<code>\$a6="347"</code>	<code>echo gettype(\$a6)</code>	<code>string</code>
<code>\$a7="3.1416"</code>	<code>echo gettype(\$a7)</code>	<code>string</code>
<code>\$a8="Solo literal"</code>	<code>echo gettype(\$a8)</code>	<code>string</code>
<code>\$a9="12.3 Literal con número"</code>	<code>echo gettype(\$a9)</code>	<code>string</code>
<code>\$a10=""</code>	<code>echo gettype(\$a10)</code>	<code>string</code>

COMENTARIOS

Crea las 10 variables como las que aparecen en el ejemplo de la a1 a la a10 y diferencia entre el texto “echo gettype(\$a10)” y el resultado de la función echo gettype(\$a10).

UD2 – ACTIVIDAD 3: Operaciones y tipos.

TIPO	Desarrollo
OBJETIVOS	Practicar la gestión de tipos y como se comportan estos cuando se opera sobre ellos.
RECURSOS	Editor de texto y navegador web.

ENUNCIADO DE LA ACTIVIDAD

Crea una tabla como la que sigue para comprobar el comportamiento de los tipos de datos cuando se opera sobre ellos.

Resultados de operaciones y tipos de variables resultantes					
Valores			Tipos de variables		
A	B	A+B	A	B	A+B
12	16	28	integer	integer	integer
12	2147483647	2147483659	integer	integer	double
-12	-2147483640	-2147483652	integer	integer	double
12	1.2456	13.2456	integer	double	double
1.2456	12	13.2456	double	integer	double
1.2456	123.4567	124.7023	double	double	double
12	abc	12	integer	string	integer
1.2456	abc	1.2456	double	string	double
12	12abc	24	integer	string	integer
12	12.34567abc	24.34567	integer	string	double
1.2456	12.34567abc	13.59127	double	string	double
1.2456	12.3e2abc	1231.2456	double	string	double
abc	12abc	12	string	string	integer
abc	12.34567abc	12.34567	string	string	double
12abc	12.34567abc	24.34567	string	string	double

COMENTARIOS

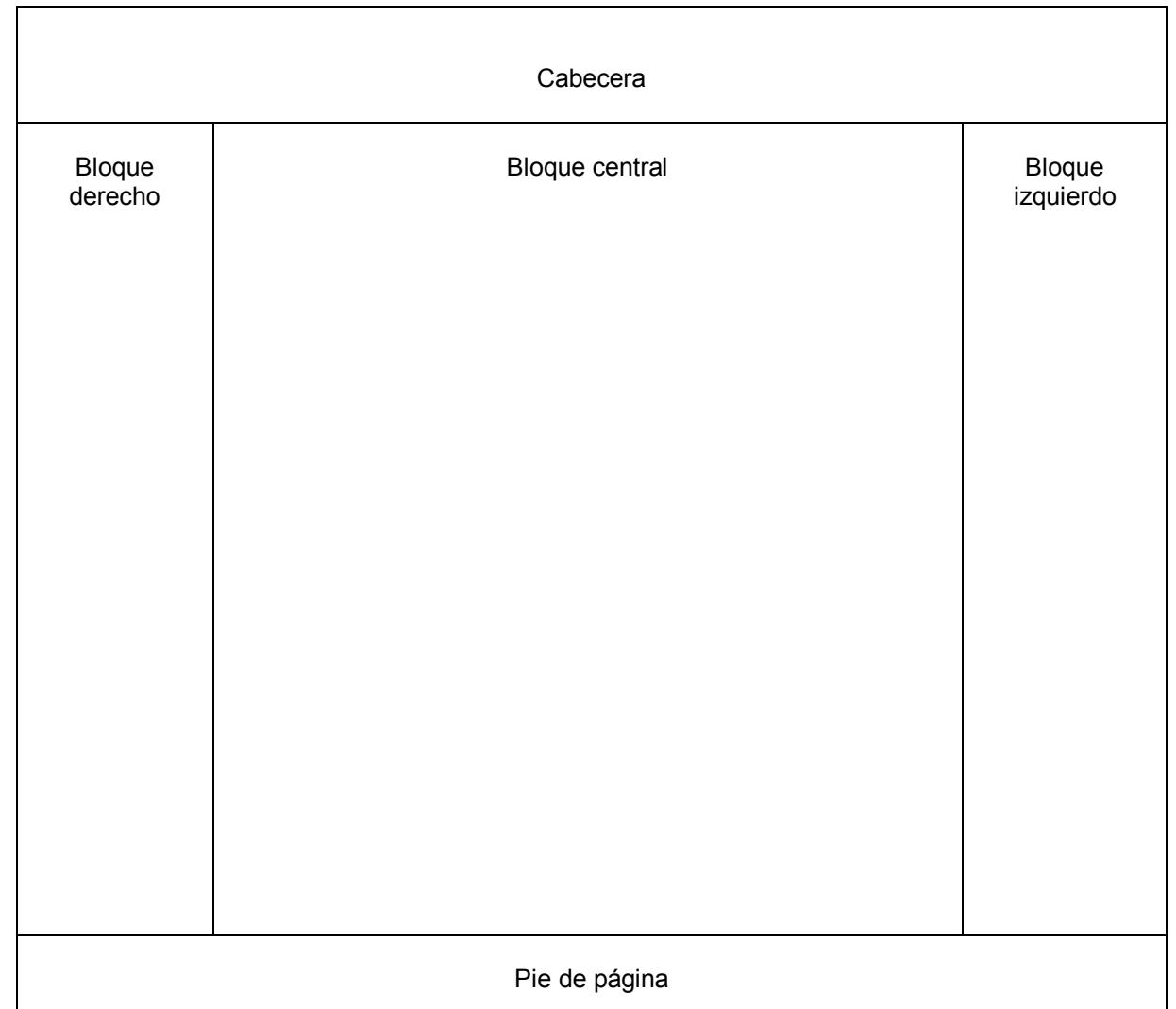
Estudiar con detenimiento el comportamiento de tipos no numéricos cuando se juntan con los numéricos.

UD2 – ACTIVIDAD 4: Diseñando una homepage.

TIPO	Desarrollo
OBJETIVOS	Practicar la inclusión de ficheros mediante include()
RECURSOS	Editor de texto y navegador web.

ENUNCIADO DE LA ACTIVIDAD

Crea una página mediante includes que nos permita diseñar una homepage en diferentes páginas según su contenido. El modelo es el siguiente.



COMENTARIOS

Crea primero una página principal con una tabla en su interior en la que deberás incluir por cada celda un include apuntando al fichero correspondiente.

3 – Estructuras de control.

Controlar el flujo de ejecución de un programa (en función de unas entradas o condiciones) es la parte más importante de una aplicación. A continuación veremos las estructuras de control, tanto selectivas como iterativas, que nos permiten dirigir un algoritmo.

3.1 - Estructuras de control selectivas.

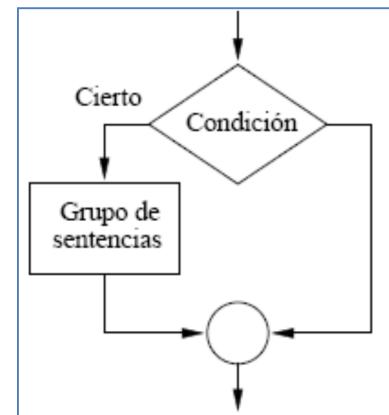
Estas estructuras de control nos van a permitir seleccionar qué sentencias vamos a ejecutar o interpretar.

Sentencia IF

Esta sentencia de control permite ejecutar o no una sentencia simple o compuesta según se cumpla o no una determinada condición. Esta sentencia tiene la siguiente forma general:

```
if (expresion)
    sentencia;
```

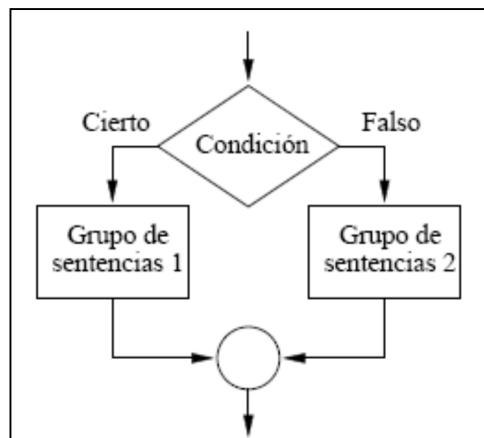
Explicación: Se evalúa **expresion**. Si el resultado es **true** (#0), se ejecuta **sentencia**; si el resultado es **false** (=0), se salta **sentencia** y se prosigue en la línea siguiente. Hay que recordar que **sentencia** puede ser una sentencia simple o compuesta (*bloque* { ... }).



Sentencia IF ... ELSE

Esta sentencia permite realizar una *bifurcación*, ejecutando una parte u otra del programa según se cumpla o no una cierta condición. La forma general es la siguiente:

```
if (expresion)
    sentencia_1;
else
    sentencia_2;
```



Explicación: Se evalúa expresion. Si el resultado es **true** (#0), se ejecuta **sentencia_1** y se prosigue en la línea siguiente a **sentencia_2**; si el resultado es **false** (=0), se salta **sentencia_1**, se ejecuta **sentencia_2** y se prosigue en la línea siguiente. Hay que indicar aquí también que **sentencia_1** y **sentencia_2** pueden ser sentencias simples o compuestas (*bloques* { ... }).

```
<?php
$valor = 23;
$valor2 = 27;
if ($valor < $valor2) {
    echo "La variable valor es menor que valor2";
```

```

} else {
    echo "La variable valor es mayor que valor2";
}
?>

```

Sentencia IF ... ELSE múltiple

Esta sentencia permite realizar una ramificación múltiple, ejecutando *una* entre varias partes del programa según se cumpla *una* entre *n* condiciones. La forma general es la siguiente:

```

if (expresion_1)
    sentencia_1;
else if (expresion_2)
    sentencia_2;
else if (expresion_3)
    sentencia_3;
else if (...)
...
[else sentencia_n;]

```

Explicación: Se evalúa expresion_1. Si el resultado es *true*, se ejecuta sentencia_1. Si el resultado es *false*, se salta sentencia_1 y se evalúa expresion_2. Si el resultado es *true* se ejecuta sentencia_2, mientras que si es *false* se evalúa expresion_3 y así sucesivamente. Si ninguna de las expresiones o condiciones es *true* se ejecuta expresion_n que es la opción por defecto (puede ser la sentencia vacía, y en ese caso puede eliminarse junto con la palabra else). Todas las sentencias pueden ser simples o compuestas.

```

<?php
$dia=4;
if ($dia == 1) {
    echo "El día es Lunes";
} elseif ($dia == 2) {
    echo "El día es Martes";
} elseif ($dia == 3) {
    echo "El día es Miércoles";
} elseif ($dia == 4) {
    echo "El día es Jueves";
}
?>

```

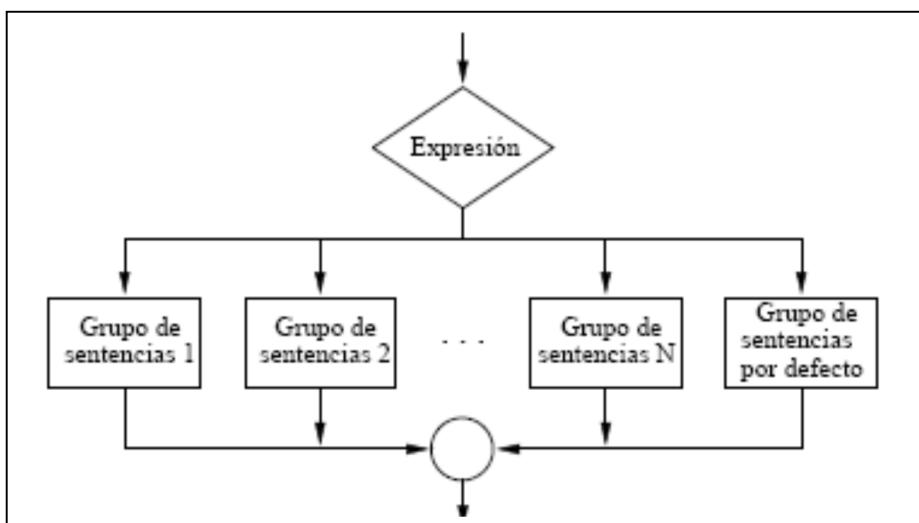
Sentencia SWITCH

La sentencia que se va a describir a continuación desarrolla una función similar a la de la sentencia **if ... else** con múltiples ramificaciones, aunque como se puede ver presenta también importantes diferencias. La forma general de la sentencia **switch** es la siguiente:

```

switch (expresion) {
    case expresion_cte_1:
        sentencia_1;
    case expresion_cte_2:
        sentencia_2;
    ...
    case expresion_cte_n:
        sentencia_n;
    [default: sentencia;]
}

```



Explicación: Se evalúa expresión y se considera el resultado de dicha evaluación. Si dicho resultado coincide con el valor constante expresion_cte_1, se ejecuta sentencia_1 seguida de sentencia_2, sentencia_3, ..., sentencia. Si el resultado coincide con el valor constante expresion_cte_2, se ejecuta sentencia_2 seguida de sentencia_3, ..., sentencia. En general, se ejecutan todas aquellas sentencias que están a continuación de la expresion_cte cuyo valor coincide con el resultado calculado al principio. Si ninguna expresion_cte coincide se ejecuta la sentencia que está a continuación de default. Si se desea ejecutar únicamente una sentencia_i (y no todo un conjunto de ellas), basta poner una sentencia break a continuación. El efecto de la sentencia break es dar por terminada la ejecución de la sentencia switch. Existe también la posibilidad de ejecutar la misma sentencia_i para varios valores del resultado de expresion, poniendo varios case expresion_cte seguidos.

```

<?php
$dia = 4 ;
switch ($dia) {
    case 1:
        echo "El día es Lunes";
        break;
    case 2 :
        echo "El día es Martes";
        break;
    case 3 :
        echo "El día es Miércoles";
        break;
    case 4 :
        echo "El día es Jueves";
        break;
    case 5:
        echo "El día es Viernes";
        break;
    case 6 :
        echo "El día es Sábado";
        break;
    case 7:
        echo "El día es Domingo" ;
        break;
    default:
        echo "El día de la semana es incorrecto";
}
?>
  
```

Sentencias IF anidadas

Una sentencia ***if*** puede incluir otros ***if*** dentro de la parte correspondiente a su **sentencia**, A estas sentencias se les llama **sentencias anidadas** (una dentro de otra), por ejemplo,

```
if (a >= b)
    if (b != 0.0) c = a/b;
```

3.2 - Estructuras de control iterativas.

Con estas estructuras podremos repetir un conjunto de sentencias el número de veces que sea necesario para nuestro propósito. También se les llama "bucles" en general.

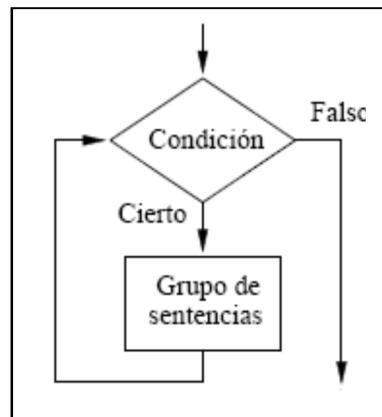
Sentencia WHILE

Esta sentencia permite ejecutar repetidamente, *mientras se cumpla una determinada condición*, una sentencia o bloque de sentencias. La forma general es como sigue:

```
while (expresion_de_control)
    sentencia;
```

Explicación: Se evalúa expresion_de_control y si el resultado es *false* se salta sentencia y se prosigue la ejecución. Si el resultado es *true* se ejecuta sentencia y se vuelve a evaluar expresion_de_control (evidentemente alguna variable de las que intervienen en expresion_de_control habrá tenido que ser modificada, pues si no el *bucle* continuaría indefinidamente). La ejecución de sentencia prosigue hasta que expresion_de_control se hace *false*, en cuyo caso la ejecución continúa en la línea siguiente a sentencia. En otras palabras, sentencia se ejecuta repetidamente mientras expresion_de_control sea *true*, y se deja de ejecutar cuando expresion_de_control se hace *false*. Obsérvese que en este caso el *control* para decidir si se sale o no del *bucle* está antes de sentencia, por lo que es posible que sentencia no se llegue a ejecutar ni una sola vez.

El siguiente ejemplo muestra una instrucción while que no se ejecuta nunca, porque la condición es falsa:



```
<?php
$variable = false;
while ($variable) {
    echo "Esta linea no se ejecuta nunca";
}
?>
```

Existe la posibilidad de que un bucle se ejecute infinitas veces, si dentro de las instrucciones no existe nada que cambie la condición que se evalúa al principio.

```
<?php
$variable = true;
while ($variable) {
    echo "CUIDADO: Esta linea se ejecuta siempre";
}
?>
```

El bucle while se puede utilizar para ejecutar un número determinado de veces las instrucciones que implica. Véase el ejemplo de la serie de Fibonacci.

```
<?php
$numero_anterior = 1;
$numero_posterior = 1;
$serie = 1 ;
$fin = 10000;
echo "Serie de Fibonacci:";
while ($serie < $fin) {
    echo $serie. ", ";
    $serie = $numero_anterior + $numero_posterior;
    $numero_anterior = $numero_posterior;
    $numero_posterior = $serie;
}
?>
```

Cada número de la serie de Fibonacci se forma sumando los dos números anteriores. En el ejemplo, la condición es que el número de la serie sea menor que número que marca la variable \$fin, es decir, si el número que indica la serie es mayor que 10.000 el bucle finaliza.

La variable \$fin sirve de valor máximo y puede cambiarlo en función de la cantidad de números que quiera tener en pantalla.

Sentencia FOR

For es quizás el tipo de bucle más versátil y utilizado del lenguaje C. Su forma general es la siguiente:

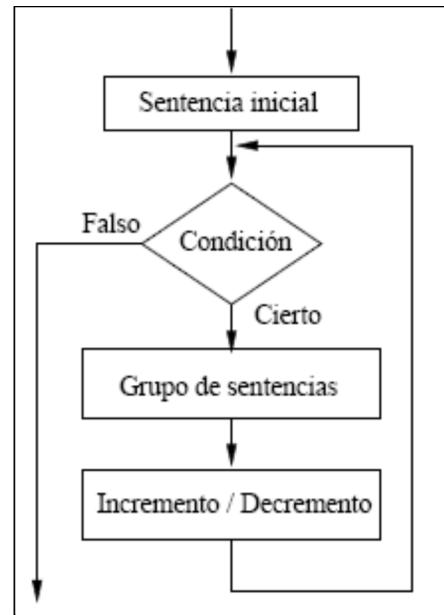
```
for (inicializacion; expresion_de_control; actualizacion)
    sentencia;
```

Explicación: La forma más sencilla de explicar la sentencia *for* sea utilizando la construcción *while* que sería equivalente. Dicha construcción es la siguiente:

```
inicializacion;
while (expresion_de_control) {
    sentencia;
    actualizacion;
}
```

donde *sentencia* puede ser una única sentencia terminada con (;), otra sentencia de control ocupando varias líneas (*if*, *while*, *for*, ...), o una sentencia compuesta o un bloque encerrado entre llaves {...}. Antes de iniciarse el bucle se ejecuta *inicializacion*, que es una o más sentencias que asignan valores iniciales a ciertas variables o contadores. A continuación se evalúa *expresion_de_control* y si es *false* se prosigue en la sentencia siguiente a la construcción *for*; si es *true* se ejecutan *sentencia* y *actualizacion*, y se vuelve a evaluar *expresion_de_control*. El proceso prosigue hasta que *expresion_de_control* sea *false*. La parte de *actualizacion* sirve para actualizar variables o incrementar contadores.

Un ejemplo clásico es la tabla de multiplicar:



```

<html><title>La tabla de multiplicar</title>
<body>
<b>TABLA DE MULTIPLICAR</b>
<table border="1">
<?php
echo ("<tr>");
echo ("<th></th>" );
for ($cabecera = 1; $cabecera <= 10; $cabecera++) {
    echo ("<th>" );
    echo $cabecera;
    echo ("</th>" );
}
echo ("</tr>");
for ($x = 1;$x <= 10; $x++ ) {
    echo ("<tr>" );
    echo ("<th>" );
    echo $x;
    echo ("</th>" );
for ($y = 1;$y <= 10; $y++ ) {
    $multiplicacion = $x * $y;
    echo ("<td>" );
    echo ("$multiplicacion");
    echo ("</td>" );
}
echo ("</tr>");
} ?>
</table>
</body></html>

```

El ejemplo se sirve de un bucle for, que crea una cabecera de tabla con la etiqueta especial `<th>`, donde inserta los números del uno al diez. Estos umeros escritos en horizontal se podrán cruzar con los verticales para atener así el resultado de una multiplicación.

El siguiente paso es un conjunto de dos bucles anidados. El primer bucle almacena en la variable `$x` los valores numéricos que aparecerán como cabecera de la columna uno. El segundo bucle será el encargado de multiplicar el valor actual de `$x` por los valores del 1 al 10. Cada vez que se ejecuta el segundo for se obtiene una celda nueva con un valor determinado por la multiplicación y cada vez que se ejecuta el primer bucle, aparece una nueva fila cuya cabecera es un número del 1 al 10.

		1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10	
2	2	4	6	8	10	12	14	16	18	20	
3	3	6	9	12	15	18	21	24	27	30	
4	4	8	12	16	20	24	28	32	36	40	
5	5	10	15	20	25	30	35	40	45	50	
6	6	12	18	24	30	36	42	48	54	60	
7	7	14	21	28	35	42	49	56	63	70	
8	8	16	24	32	40	48	56	64	72	80	
9	9	18	27	36	45	54	63	72	81	90	
10	10	20	30	40	50	60	70	80	90	100	

Sentencia FOREACH

El bucle foreach es específico de los array y aplicable a ellos tanto si son escalares como si son de tipo asociativo. Tiene dos posibles opciones. En una de ellas lee únicamente los valores contenidos en cada elemento del array. En el otro caso lee además los índices del array.

Lectura de valores. Utiliza la sintaxis:

```
foreach( array as var ){
...instrucciones...
}
```

donde array es el nombre del array (sin incluir índices ni corchetes), as es una palabra obligatoria y var el nombre de una variable (puede ser creada al escribir la instrucción ya que no requiere estar previamente definida).

Las instrucciones escritas entre las {} permiten el tratamiento o visualización de los valores obtenidos.

La variable var no podrá ser utilizada para guardar valores. Hemos de tener en cuenta que su valor se reescribe en cada iteración del bucle y que al acabar este sólo contendrá el último de los valores leídos.

Lectura de índices y valores.

Con una sintaxis como la que sigue se pueden leer no sólo los valores de un array sino también sus índices.

```
foreach( array as v1 => v2 ) {
...instrucciones...
}
```

donde array es el nombre de la matriz, as es una palabra obligatoria, v1 es el nombre de la variable que recogerán los índices, los caracteres => (son obligatorios) son el separador entre ambas variables y, por último, v2 es el nombre de la variable que recoge el valor de cada uno de los elementos del array.

Tanto esta función como la anterior realizan una lectura secuencial que comienza en el primer valor del array.

Arrays bidimensionales

Cuando se trata de arrays bidimensionales la lectura de los valores que contienen sus elementos requiere el uso de dos bucles anidados.

Cuando un array de este tipo es sometido al bucle foreach se extrae como índice el primero de ellos y como valor un array unidimensional que tiene como índice el segundo del array original y como valor el de aquél. La lectura de los valores de cada elemento requiere utilizar un segundo bucle que los vaya extrayendo a partir del array unidimensional obtenido por medio del bucle previo. La sintaxis sería de este tipo:

```
foreach($a as $i1=>$na){
    foreach($na as $i2=>$val){
        ..$i1 es el primer índice...
        ..$i2 es el segundo índice...
        ..$na nuevo array
        ..$valor es el valor
        ....
    }
}
```

En el caso de arrays con dimensiones superiores sería necesario proceder del mismo modo, y habría que utilizar tantos bucles foreach como índices contuviera el array.

```
<?
$a=array("a","b","c","d","e");
$b=array(
"uno" =>"Primer valor",
"dos" =>"Segundo valor",
"tres" =>"Tercer valor",
```

```

);
# en este caso extraeremos índices y valores de ambos arrays
# usaremos $pepe para recoger los índices y $pepe para recoger los valores
# y separaremos ambas variables por => que es el separador obligatorio
# para estos casos

foreach($a as $pepe=>$pepa) {
    echo "Indice: ",$pepe," Valor: ",$pepa,"<br>";
}
foreach($b as $pepe=>$pepa) {
    echo "Indice: ",$pepe," Valor: ",$pepa,"<br>";
}
?>

```

Sentencia DO ... WHILE

Esta sentencia funciona de modo análogo a **while**, con la diferencia de que la evaluación de **expresion_de_control** se realiza al final del bucle, después de haber ejecutado al menos una vez las sentencias entre llaves; éstas se vuelven a ejecutar mientras **expresion_de_control** sea **true**. La forma general de esta sentencia es:

```

do
    sentencia;
    while(expresion_de_control);

```

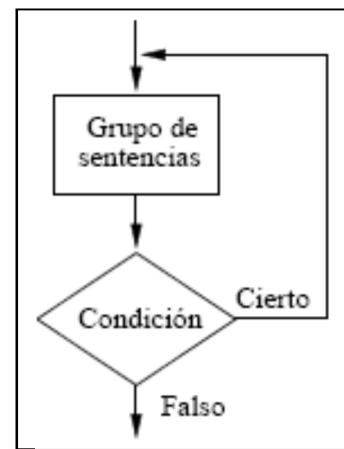
Puede comprobar que la diferencia es mínima cuando utiliza esta estructura de control para la serie de Fibonacci.

```

<?php
$numero_anterior = 1;
$numero_posterior = 1;
$serie = 1;    $fin = 6765;
echo "Serie de Fibonacci:";

do {
    echo $serie.", ";
    $serie = $numero_anterior + $numero_posterior;
    $numero_anterior = $numero_posterior;
    $numero_posterior = $serie;
} while ($serie < $fin);
?>

```



Alteración de los bucles: Sentencias break y continue.

La instrucción **break** interrumpe la ejecución del bucle donde se ha incluido, haciendo al programa salir de él aunque la **expresion_de_control** correspondiente a ese bucle sea verdadera.

La sentencia **continue** hace que el programa comience el siguiente ciclo del bucle donde se halla, aunque no haya llegado al final de las sentencias compuesta o bloque.

El código siguiente muestra la forma de utilizar **break**:

```

<?php
for ($x = 1; $x < 20; $x++) {
    if ($x == 10) {
        break;
    } else {
        echo "$x<br>";
    }
}

```

La salida por pantalla es una sucesión de números del 1 al 9. Al llegar la variable \$x al número 10, la condición del if se cumple y se ejecuta la instrucción break, saliendo del bucle. Veamos ahora el mismo ejemplo, pero utilizando continue:

```
<?php
for ($x = 1; $x < 20; $x++) {
    if ($x == 10) {
        continue;
    } else {
        echo "$x<br>";
    }
?>
```

En este caso, cuando \$x alcanza el valor 10, la instrucción continue es ir hacia el final de bucle y volver al principio comprobando la condición. El resultado por pantalla es una sucesión de números desde el 1 hasta el 19, sin incluir el número 10.

3.3 - Finalizar la ejecución de un programa.

Hay veces que necesitamos parar la ejecución de un programa por diversas causas: ha ocurrido un error, un fallo en la entrada del nombre y la contraseña. La función para la ejecución es **exit()** o **die()**. Estas dos funciones aceptan un parámetro que se imprime en pantalla. Por ejemplo, considere el siguiente código que asume la conexión a una base de datos.

```
<?php
$conexion = conectar_base_datos("libros");
if (! $conexion) {
    die ("Se ha producido algún error en la conexión");
}
?>
```

Se puede apreciar que si la variable \$conexión se evalúa false (no se ha conseguido la conexión), se detiene la ejecución del programa. Una versión más compacta se consigue utilizando el operador or:

```
<?php
$conexion = conectar_base_datos("libros") or
die ("Se ha producido algún error en la conexión");
?>
```

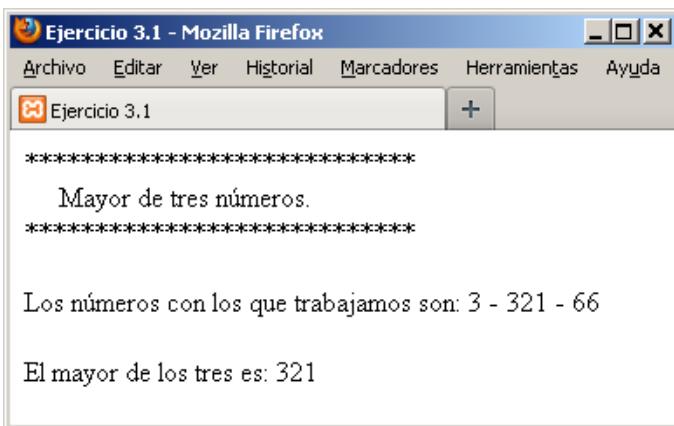
Actividades.

UD3 – ACTIVIDAD 1: El mayor de tres números.

TIPO	Desarrollo
OBJETIVOS	Ejercitarse con las estructuras de control. Elemento IF.

ENUNCIADO DE LA ACTIVIDAD

Diseñar un script php que dadas tres variables de tipo numérico, sea capaz de determinar cual es el mayor de los tres números



COMENTARIOS

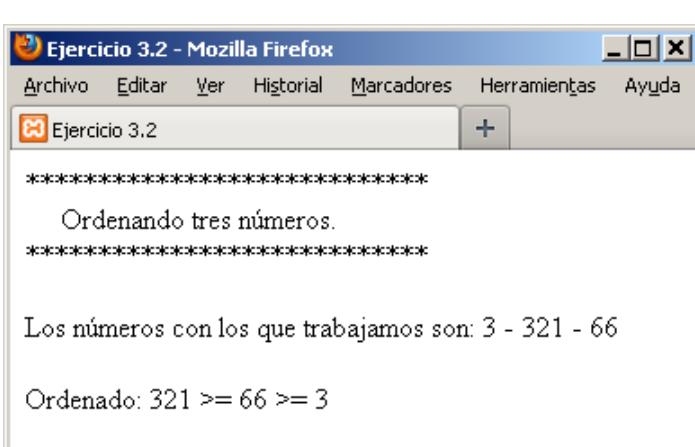
Se debe utilizar la estructura de comparación IF para determinar que variable de las tres es la que contiene una cantidad mayor. Las comparaciones se realizarán por parejas de variables. Ejemplo: si (variable1>=variable2 Y variable1>variable2) .

UD3 – ACTIVIDAD 2: Ordenando números.

TIPO	Desarrollo
OBJETIVOS	Ejercitarse con las estructuras de control. Elemento IF.

ENUNCIADO DE LA ACTIVIDAD

Diseñar un script php que dadas tres variables de tipo numérico, sea capaz de determinar cual es el mayor de los tres números



COMENTARIOS

Teniendo tres números (A,B,C) las combinaciones para ordenarlos son las siguientes:
 $A \geq B \geq C$, $A \geq C \geq B$, $B \geq A \geq C$, $B \geq C \geq A$, $C \geq A \geq B$, $C \geq B \geq A$

3 – Formato usando etiquetas.

UD3 – ACTIVIDAD 3: Cambio de temperatura

TIPO	Consolidación
OBJETIVOS	Establecer las diferencias entre los bucles tipo while y los de tipo for. Afianzar los conocimientos en el uso de sentencias iterativas.

ENUNCIADO DE LA ACTIVIDAD

Realiza un programa en PHP que escriba una tabla de dos columnas para la conversión entre las temperaturas en grados Fahrenheit (comprendidas entre 0 °F y 300 °F, según incrementos de 20 °F) y su equivalente en grados centígrados. Se realizarán dos versiones de este programa: una llamada temp1.c que empleará un bucle while. La otra versión se llamará temp2.c y utilizará un bucle for.

La conversión entre grados Centígrados y grados Fahrenheit obedece a la fórmula de al lado, siendo °C la temperatura en grados Centígrados y °F en grados Fahrenheit.

$$^{\circ}\text{C} = \frac{5 * (^{\circ}\text{F} - 32)}{9}$$

The screenshot shows a Firefox browser window titled "Ejercicio 3.3 - Mozilla Firefox". The address bar also says "Ejercicio 3.3". The page content displays a table with two columns: "grados Fahrenheit" and "grados Centigrados". The table lists temperatures from 0 to 300 in increments of 20, showing their equivalents in Celsius. The data is as follows:

grados Fahrenheit	grados Centigrados
0	-17.777777777778
20	-6.66666666666667
40	4.44444444444444
60	15.555555555556
80	26.666666666667
100	37.777777777778
120	48.888888888889
140	60
160	71.111111111111
180	82.222222222222
200	93.333333333333
220	104.444444444444
240	115.555555555556
260	126.666666666667
280	137.777777777778
300	148.888888888889

COMENTARIOS

Se recomienda la utilización de bucles para resolver el ejercicio, se puede encontrar soluciones análogas utilizando FOR, WHILE, DO WHILE o REPEAT.

Actividades propuestas.

3.4) Realizar un programa que pida al usuario una cantidad en quilogramos (kg). A continuación mostrará por pantalla el resultado en gramos (gr), hectogramos (hg) centigramos.

3.5) Escriba un programa que calcule el área de un triángulo rectángulo, dada la altura y la base.

3.6) Realizar un programa que calcule la suma de los 1000 primeros números.

3.7) Realizar un programa que muestre por pantalla la tabla de multiplicar del 10.

3.8) Escribe un programa que muestre en pantalla lo siguiente usando bucles: pista --> utiliza un doble bucle.

```
*  
**  
***  
****  
*****
```

3.9) Realizar un programa que muestre por pantalla las tablas de multiplicar. Empezará mostrando la tabla del 1, después mostrará la tabla del 2. El proceso se repetirá hasta que muestre la tabla del 10.

3.10) Crear el código PHP necesario para que muestre los números pares entre el 10 y el 50 (incluidos). Usar la sentencia "for".

3.11) Crear el código PHP necesario para que muestre los números múltiplos de 3 entre los números almacenados en dos variables: \$inicio y \$final. Usar la sentencia "for".

3.12) Crear el código PHP necesario para que muestre los números entre los números almacenados en dos variables: \$inicio y \$final y nos diga de cada uno si es múltiplo de 2, de 3 o las dos cosas a la vez. Usar la sentencia "for".

3.13) Escribe un programa que muestre en pantalla lo siguiente usando bucles: pista --> utiliza un doble bucle.

```
*  
**  
***  
****  
*****  
***  
**  
*
```

4 – Funciones.

4.1 – Diseño modular.

La programación era en sus comienzos todo un arte (esencialmente cuestión de inspiración); posteriormente diversas investigaciones teóricas han dado lugar a una serie de principios generales que permiten conformar el núcleo de conocimientos de una metodología de la programación. Ésta consiste en obtener "programas de calidad".

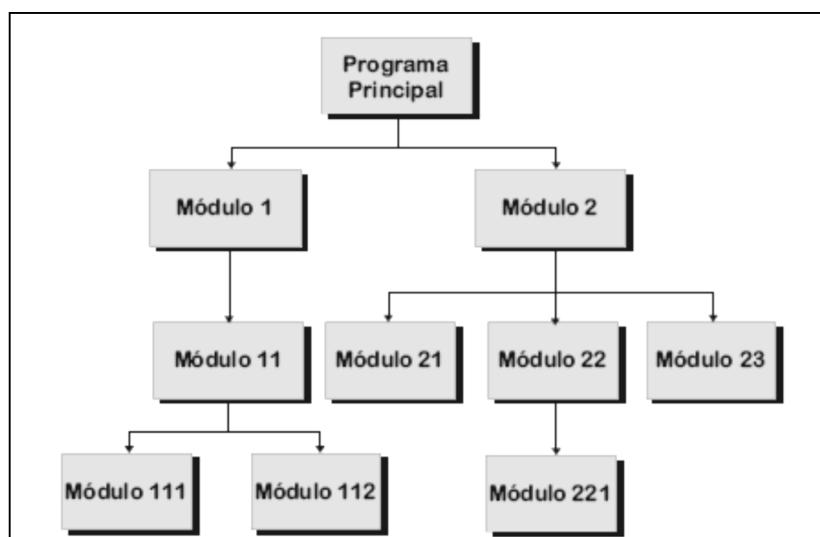
Teniendo en cuenta que un programa, a lo largo de su vida, es escrito solo una vez, pero leído, analizado y modificado muchas más, cobra una gran importancia adquirir técnicas de diseño y desarrollo adecuadas para obtener programas con las características mencionadas. El objetivo de la programación modular es que los programas sean comprensibles, más fáciles de abordar y reutilizables "transportables y fácilmente mantenibles".

La programación modular es la técnica de programación basada en la filosofía del diseño descendente, que consiste en dividir el problema original en diversos subproblemas (y estos a su vez en otros más pequeños, obteniendo una estructura jerárquica o en árbol) que se pueden resolver por separado, para después recomponer los resultados y obtener la solución al problema. Un subproblema se denomina módulo y es una parte del problema que se puede resolver de manera independiente.

Un módulo es una colección estática de declaraciones definidas en un ámbito de visibilidad particular y oculta al resto del programa con el que se comunica por una sección de interfaz donde se incluyen la lista de exportaciones. Usando módulos se construyen las unidades en que se ha de descomponer cualquier programa mínimamente importante. Los módulos se conectan entre sí dando lugar a una estructura modular en árbol que permite resolver el problema de programación planteado.

Un módulo actúa como una caja negra con la cual el resto del programa interactúa a través de una sección de interfaz. La interfaz (o vista pública) es una colección de declaraciones de constantes, tipos, variables, procedimientos, funciones, etc. La otra sección principal de un módulo es la implementación (o vista privada) que incluye el código de las funciones y demás elementos constitutivos de la parte ejecutable del módulo.

Para efectuar un buen diseño modular los algoritmos que se van a desarrollar se han de concebir como una jerarquía de módulos intercomunicados donde cada uno de ellos presenta una función clara y diferenciada y en la que ningún módulo accede directamente al interior de otros módulos sino que siempre utiliza los mecanismos de interfaz.



Ventajas:

- Disminuye la complejidad del algoritmo
- Disminuye el tamaño total del programa
- *Reusabilidad*: ahorro de tiempo de programación
- División de la programación entre un equipo de programadores reducción del tiempo de desarrollo
 - Facilidad en la *depuración*: comprobación individual de los módulos
 - Programas más fáciles de modificar
 - Estructuración en *librerías* específicas (biblioteca de módulos)

Interfaz.

Un módulo puede ofrecer a la "comunidad" de módulos sus propios recursos, tipos y funciones dentro de lo que se conoce como sección de interfaz del módulo. Normalmente es posible exportar cinco tipos de elementos: constantes, tipos, variables, procedimientos y funciones.

La importación y la exportación constituyen acciones simétricas. Para que un módulo pueda importar un tipo, procedimiento u otro elemento es preciso que otro lo exporte. Por supuesto, es posible que el mismo módulo se comporte como exportador e importador.

Implementación.

Para servirse de un tipo o función ajena a un módulo es obligatorio especificar el lugar donde se han definido originalmente dichos elementos. Además, es conveniente que esa declaración de objetos ajenos al módulo pero usados en él, se lleve a cabo en una sección bien diferenciada. Para tal fin se puede utilizar alguna versión de la denominada lista de importaciones.

Cada lenguaje presenta sus peculiaridades, así, en C, las importaciones se pueden incluir explícitamente declarando las funciones en el mismo módulo o haciendo uso de un archivo de cabeceras con la sintaxis:

```
include("fichero");
```

donde fichero corresponde al nombre del archivo concreto que se utilice.

4.2 - Concepto de función.

Las aplicaciones informáticas que habitualmente se utilizan, incluso a nivel de informática personal, suelen contener decenas y aún cientos de miles de líneas de código fuente. A medida que los programas se van desarrollando y aumentan de tamaño, se convertirían rápidamente en sistemas poco manejables si no fuera por la *modularización*, que es el proceso consistente en dividir un programa muy grande en una serie de módulos mucho más pequeños y manejables.

A estos módulos se les ha solido denominar de distintas formas (*subprogramas*, *subrutinas*, *procedimientos*, *funciones*, etc.) según los distintos lenguajes. El lenguaje C hace uso del concepto de **función (function)**. Sea cual sea la nomenclatura, la idea es sin embargo siempre la misma: dividir un programa grande en un conjunto de subprogramas o funciones más pequeñas que son llamadas por el programa principal; éstas a su vez llaman a otras funciones más específicas y así sucesivamente.

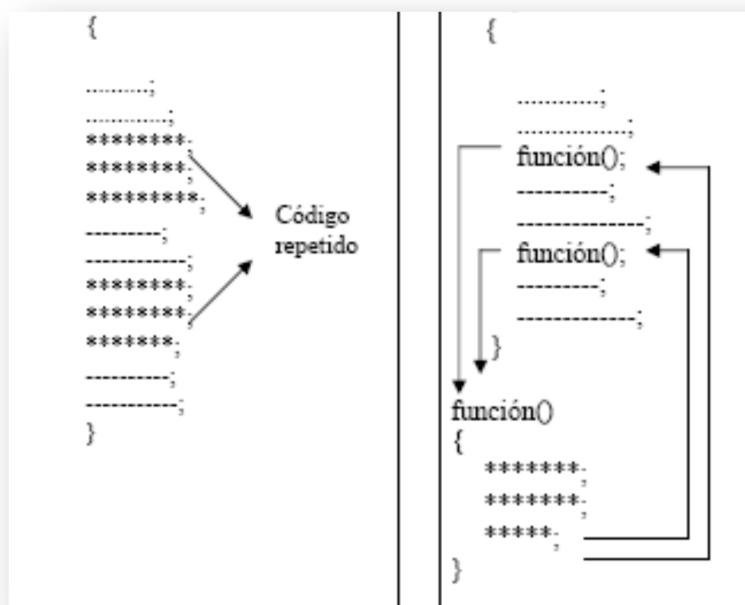
La división de un programa en unidades más pequeñas o funciones presenta las ventajas siguientes:

1. **Modularización.** Cada función tiene una misión muy concreta, de modo que nunca tiene un número de líneas excesivo y siempre se mantiene dentro de un tamaño manejable. Además, una misma función (por ejemplo, un producto de matrices, una resolución de un sistema de ecuaciones lineales,...) puede ser llamada muchas veces en

un mismo programa, e incluso puede ser reutilizada por otros programas. Cada función puede ser desarrollada y comprobada por separado.

2. **Ahorro de memoria y tiempo de desarrollo.** En la medida en que una misma función es utilizada muchas veces, el número total de líneas de código del programa disminuye, y también lo hace la probabilidad de introducir errores en el programa.

3. **Independencia de datos y ocultamiento de información.** Una de las fuentes más comunes de errores en los programas de computador son los *efectos colaterales* o perturbaciones que se pueden producir entre distintas partes del programa. Es muy frecuente que al hacer una modificación para añadir una funcionalidad o corregir un error, se introduzcan nuevos errores en partes del programa que antes funcionaban correctamente. Una función es capaz de mantener una gran independencia con el resto del programa, manteniendo sus propios datos y definiendo muy claramente la *interfaz* o comunicación con la función que la ha llamado y con las funciones a las que llama, y no teniendo ninguna posibilidad de acceso a la información que no le compete.



4.3 – Funciones de usuario.

Una **función** es una parte de código independiente del programa principal, que puede ser llamada enviándole unos datos (o sin enviarle nada), para que realice una determinada tarea y/o proporcione unos resultados.

Utilidad de las funciones.

- *Modularidad*, esto es su división en partes más pequeñas de finalidad muy concreta.
- Las funciones facilitan el desarrollo y mantenimiento de los programas, evitan errores, y ahorran memoria y trabajo innecesario.
- Una misma función puede ser utilizada por diferentes programas, y por tanto no es necesario reescribirla.

- Además, una función es una parte de código independiente del programa principal y de otras funciones, manteniendo una gran independencia entre las variables respectivas, y evitando errores y otros efectos colaterales de las modificaciones que se introduzcan.

- Mediante el uso de funciones se consigue un código limpio, claro y elegante. La adecuada división de un programa en funciones constituye un aspecto fundamental en el desarrollo de programas de cualquier tipo.

- Las funciones, ya compiladas, pueden guardarse en **librerías**. Las librerías son conjuntos de funciones compiladas, normalmente con una finalidad análoga o relacionada, que se guardan bajo un determinado nombre listas para ser utilizadas por cualquier usuario.

Definición de una función.

La **definición de una función** consiste en la definición del código necesario para que ésta realice las tareas para las que ha sido prevista. La definición de una función se debe realizar en alguno de los ficheros que forman parte del programa. La forma general de la definición de una función es la siguiente:

```
function nombre_funcion(lista de parámetros con tipos) {
    declaración de variables y/o de otras funciones
    código ejecutable
    return (expresión); // optativo
}
```

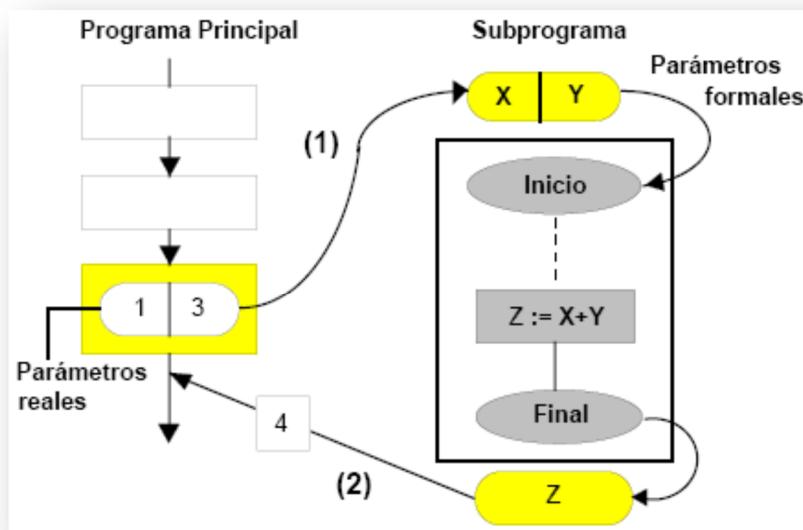
- La primera línea recibe el nombre de **encabezamiento** (header) y el resto de la definición –encerrado entre llaves– es el **cuerpo** (**body**) de la función.

- Cada función puede disponer de sus propias variables, *declaradas* al comienzo de su código. Estas variables, por defecto, son de tipo **auto**, es decir, sólo son visibles dentro del bloque en el que han sido definidas, se crean cada vez que se ejecuta la función y permanecen ocultas para el resto del programa. Si estas variables se definen como **static**, conservan su valor entre distintas llamadas a la función. También pueden hacerse visibles a la función *variables globales* definidas en otro fichero (o en el mismo fichero, si la definición está por debajo de donde se utilizan), declarándolas con la palabra clave **extern**.

- El **código ejecutable** es el conjunto de instrucciones que deben ejecutarse cada vez que la función es llamada.

- La **lista de parámetros con tipos**, también llamados **parámetros formales**, es una lista de declaraciones de variables, precedidas por su *tipo* correspondiente y separadas por comas.

Los **parámetros formales** son los utilizados en la cabecera y son la forma más natural y directa para que la función reciba valores desde el programa que la llama, correspondiéndose en número y tipo con otra lista de argumentos -los **parámetros actuales**- que son los utilizados en el programa que realiza la llamada a la función.



Normas para escribir la lista de argumentos formales:

- El número de parámetros formales y actuales debe ser coincidente.
- La asociación de argumentos es posicional: el primer parámetro actual se corresponde con el primer parámetro formal, el segundo con el segundo, etc...
- Los parámetros actuales y formales deben ser tipos de datos compatibles.
- Los argumentos actuales pueden ser variables, constantes o cualquier expresión del tipo de dato correspondiente al parámetro formal.
- Las llamadas pueden ser por valor o por referencia.
- Cuando una función es ejecutada, puede devolver al programa que la ha llamado un valor (el **valor de retorno**).

La sentencia **return** permite devolver el control al programa que llama. Puede haber varias sentencias **return** en una misma función. Si no hay ningún **return**, el control se devuelve cuando se llega al final del *cuerpo* de la función. La palabra clave **return** puede ir seguida de una *expresión*, en cuyo caso ésta es evaluada y el valor resultante devuelto al programa que llama como *valor de retorno* (si hace falta, con una conversión previa al *tipo* declarado en el encabezamiento). Los paréntesis que engloban a la *expresión* que sigue a **return** son optativos.

El valor de retorno es un valor único: *no puede ser un vector o una matriz*, aunque sí un *puntero* a un vector o a una matriz. Sin embargo, *el valor de retorno sí puede ser una estructura*, que a su vez puede contener vectores y matrices como elementos miembros.

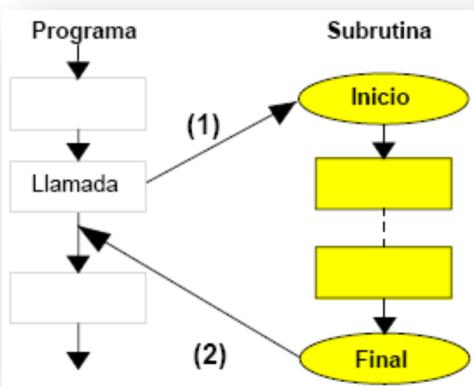
Como ejemplo supóngase que se va a calcular a menudo el *valor absoluto* de variables de tipo *double*. Una solución es definir una función que reciba como argumento el valor de la variable y devuelva ese valor absoluto como valor de retorno. La definición de esta función podría ser como sigue:

```
function valor_abs(double x) {
    if (x < 0.0)
        return -x;
    else
        return x;
}
```

Flujo de ejecución en la llamada a una función.

En la siguiente imagen de la derecha se muestra el flujo de ejecución en la llamada a una función desde el programa principal.

- 1) Se realiza la llamada y el programa principal cede el control de ejecución a la función. Esta realiza sus operaciones.
- 2) Una vez ejecutada la función (devueltos o no resultados de la función) se devuelve el control al programa principal para continuar con su ejecución.



Llamada a una función.

La **llamada a una función** se hace incluyendo su *nombre* en una expresión o sentencia del programa principal o de otra función. Este nombre debe ir seguido de una lista de *argumentos* separados por comas y encerrados entre paréntesis. A los argumentos incluidos en la llamada se les llama **parámetros actuales**, y pueden ser no sólo variables y/o constantes, sino también *expresiones*. Cuando el programa que llama encuentra el nombre de la función, evalúa los *argumentos actuales* contenidos en la llamada, los convierte si es necesario al tipo de los *argumentos formales*, y **pasa copias de dichos valores** a la función junto con el control de la ejecución.

El número de *parámetros actuales* en la llamada a una función debe coincidir con el número de *parámetros formales* en la definición y en la declaración.

```

nombre_funcion([parametros]);           // para funciones que no
                                         devuelven resultados

variable = nombre_funcion([parametros]); // para funciones
                                         que devuelven resultados

```

Existe la posibilidad de definir funciones con un *número variable o indeterminado* de argumentos. Este número se concreta luego en el momento de llamarlas. Las funciones **`printf()`** es un ejemplo de función con número variable de argumentos.

SINTAXIS DE LA LLAMADA:

```
nombre_funcion([parametros]);
```

SINTAXIS DEL DESARROLLO:

```
function nombre_funcion ([parametros])
{
    cuerpo;
}
```

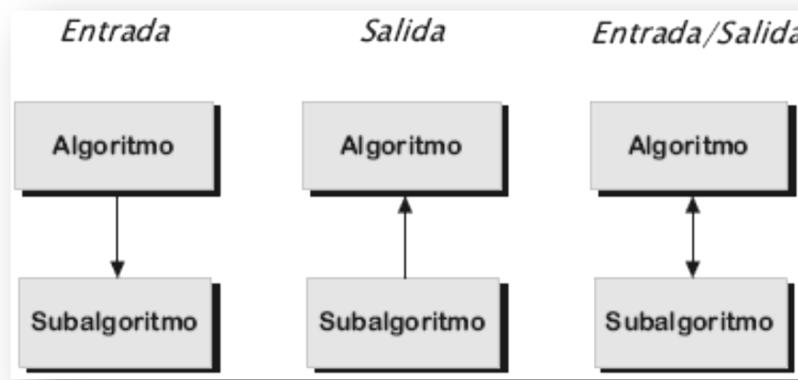
Devolución de resultados.

Cuando se llama a una función, después de realizar la conversión de los argumentos actuales, se ejecuta el código correspondiente a la función hasta que se llega a una sentencia ***return*** o al final del cuerpo de la función, y entonces se devuelve el control al programa que realizó la llamada, junto con el *valor de retorno* si es que existe (convertido previamente al *tipo* especificado en el *prototipo*, si es necesario). Recuérdese que el valor de retorno puede ser un valor numérico, una dirección (un puntero), o una estructura, pero no una matriz o un vector.

Tipos de parámetros.

En función de su papel en la función, los parámetros, formales o reales, pueden ser de tres clases:

- **Parámetros de entrada:** son aquellos que se utilizan para aportar datos a la función. Si dentro de éste se produce un cambio en el valor del parámetro formal el parámetro real no se verá afectado. Los parámetros reales en este caso pueden ser constantes, variables o expresiones.
- **Parámetros de salida:** son aquellos que se utilizan para exportar datos desde la función. No aportan valor inicial por lo que son directamente inicializados por la función que les asigna valores. Así, los cambios producidos en el valor del parámetro formal afectarán al parámetro real que deberá ser una variable.
- **Parámetros de entrada/salida:** son aquellos cuya función incluye a las dos anteriores. Por un lado aportan valores y por otro son modificados por la función para exportar valores. Los parámetros reales tienen también que ser variables.



Paso de parámetros.

Por valor: únicamente nos interesa el valor, no las modificaciones que pueda tener dentro del subalgoritmo. Se trabaja con una copia del valor pasado. Son parámetros unidireccionales, que pasan información desde el algoritmo al subalgoritmo. Puede ser cualquier expresión evaluable en ese momento.

Por referencia: se pasa una referencia a la posición de memoria donde se encuentra dicho. Se utilizan tanto para recibir como para transmitir información entre el algoritmo y el subalgoritmo. Debe ser obligatoriamente una variable.

En la llamada a una función los *argumentos actuales* son evaluados y se pasan *copias* de estos valores a las variables que constituyen los *argumentos formales* de la función. Aunque los argumentos actuales sean variables y no expresiones, y haya una correspondencia biunívoca entre ambos tipos de argumentos, los cambios que la función realiza en los argumentos formales no se trasmitten a las variables del programa que la ha llamado, precisamente porque lo que la función ha recibido son *copias*. El modificar una copia no repercute en el original. A este mecanismo de paso de argumentos a una función se le llama **paso por valor**.

Todos los cambios de valor que sufra la variable en este entorno no afectarán al programa fuera de la función.

```
<?php
function elevado($nuraero,$ índice) {
    $resultado = $numero;
    for ($x = $índice; $x > 0; $x--) {
        $resultado = $resultado * $nuraero;
    }
    $numero = $resultado;
    return $numero;
}
$numero = 2;
$índice = 5;
echo $numero." elevado a ".$índice." es igual a ".elevado($numero,$ índice). "<br>";
echo $numero." elevado a ".$índice." es igual a ".elevado ($numero,$ índice). "<br>";
?>
```

El resultado por pantalla es:

2 elevado a 5 es igual a 64
2 elevado a 5 es igual a 64

El ejemplo anterior calcula el resultado de \$numero elevado a \$índice. La función elevado () crea un bucle para multiplicar \$numero por las veces que indica el \$índice y lo va almacenando en la variable temporal \$resultado.

Cuando el bucle termina, este resultado se vuelve a pasar a la variable \$numero, que es devuelta como salida de la función. Cuando volvemos a ejecutar la función con los mismos parámetros, vemos que la variable \$numero no ha cambiado en la ejecución anterior, porque, en realidad, lo que se pasa es una copia del valor y todas las modificaciones se hacen sobre la copia y no sobre la variable original.

PHP ofrece actualmente dos caminos diferentes para cambiar sus argumentos: en la definición de la función y en la llamada a la misma. Las variables pasadas por referencia pueden ser modificadas durante el proceso de una llamada, porque lo que se pasa no es una copia, sino la variable en sí misma.

Para pasar variables por referencia hay que utilizar el operador (&) delante de la variable. Lo mejor es verlo en un ejemplo:

```
<?php
function elevado(&$numero,&$índice) {
    $resultado = $numero;
    for ($x = $índice; $x > 0; $x--) {
        $resultado = $resultado * $numero;
    }
    $numero = $resultado;
    return $numero;
}
$numero = 2 ;
$índice = 5;
echo $numero." elevado a ".$índice." es igual a ".elevado($numero,$ índice). "<br>";
echo $numero." elevado a ".$índice." es igual a ".elevado($numero,$ índice). "<br>";
?>
```

El resultado por pantalla es ahora:

2 elevado a 5 es igual a 64
64 elevado a 5 es igual a 68719476736

Al mecanismo de paso de argumentos mediante direcciones en lugar de valores se le llama **paso por referencia**, y deberá utilizarse siempre que la función deba devolver argumentos modificados. Un caso de particular interés es el paso de *arrays* (vectores, matrices y cadenas de caracteres). Este punto se tratará con más detalle un poco más

adelante. Baste decir ahora que como *los nombres de los arrays son punteros* (es decir, direcciones), dichos datos se *pasan por referencia*, lo cual tiene la ventaja adicional de que no se gasta memoria y tiempo para pasar a las funciones copias de cantidades grandes de información.

Un caso distinto es el de las **estructuras**, y conviene tener cuidado. Por defecto **las estructuras se pasan por valor**, y pueden representar también grandes cantidades de datos (pueden contener arrays como miembros) de los que se realizan y transmiten copias, con la consiguiente pérdida de eficiencia. Por esta razón, *las estructuras se suelen pasar de modo explícito por referencia, por medio de punteros a las mismas*.

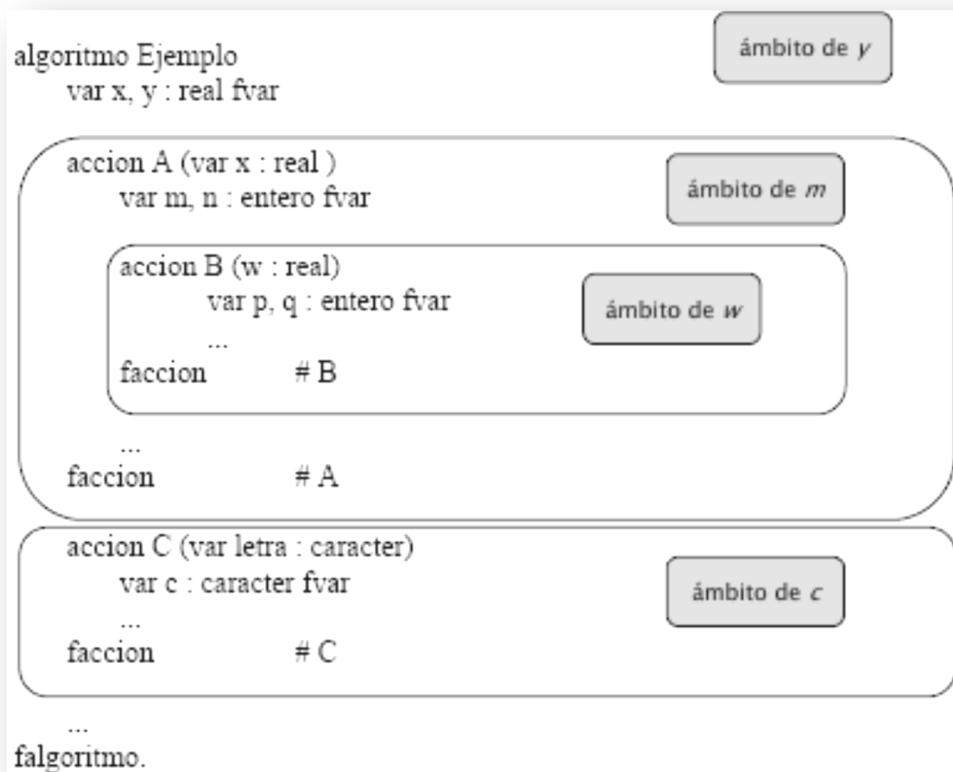
Ámbito de identificador.

El **ámbito de un identificador** es el conjunto de sentencias donde puede utilizarse ese identificador.

Variables locales: variable declarada dentro de un subprograma y, por tanto, sólo disponible durante el funcionamiento del mismo.

Variables globales: variables declaradas en el programa principal y, por ello, pueden ser utilizadas por el programa principal y por todos sus subprogramas.

Efecto Lateral: efecto de un módulo sobre otro módulo que no es parte de la interfaz definida explícitamente entre ellos.



Reglas para el cálculo del ámbito de un identificador:

1. Un identificador declarado en un bloque es accesible únicamente desde ese bloque y todos los bloques incluidos en él (se considera *local* a ese bloque). Un parámetro formal se considera también una declaración local al bloque de la función.
2. Los identificadores declarados fuera de cualquier bloque se consideran *globales* y pueden ser utilizados desde cualquier punto del fichero.
3. Cuando tenemos un bloque dentro de otro bloque y en ambos se declaran identificadores con el mismo nombre, el del bloque interno "oculta" al del bloque externo.

4.4 - Funciones con número de argumentos variables.

Es habitual que el número de parámetros que se le pase a una función dependa de la situación en la cual es llamada. Hay tres formas de hacer esto en PHP:

- Definiendo la función con argumentos por defecto. Este método permite hacer llamadas con menos parámetros sin que aparezca un error.
- Usando un *array* para pasar las variables.
- Usando las funciones de argumento variable *func_num_args()*, *func_get_arg()* y *func_get_args()* , ya utilizadas en PHP 4.

Argumentos por defecto.

Para definir parámetros de este tipo, se sustituyen las variables por expresiones de asignación. El formato es el siguiente:

```
function nombre_función($argumento1=valor1, $argumento2=valor2, ..)
```

De esta forma, al llamar a la función, podemos hacerlo con varios parámetros. Si alguno de los parámetros es obviado, la variable tendrá como valor, el valor por defecto de la definición.

```
<?php
function capitales($Pais,$Capital = "Madrid",$habitantes = "muchos") {
    return ("La capital de $Pais es $Capital y tiene $habitantes habitantes.<br>") ;
}
echo capitales("España");
echo capitales("Portugal","Lisboa");
echo capitales("Francia","Paris","muchísimos");
?>
```

La salida por pantalla es la siguiente:

La capital de España es Madrid y tiene muchos habitantes.

La capital de Portugal es Lisboa y tiene muchos habitantes.

La capital de Francia es Paris y tiene muchísimos habitantes.

La función *capitales()* tiene un parámetro fijo y dos parámetros por defecto. El parámetro *\$Pais* es obligatorio, pero los demás pueden quedarse en blanco, recibiendo el valor por defecto. La limitación de este método es que los argumentos tienen un orden y no podemos saltarnos ninguno, es decir, no podemos dar los valores *\$Capital* y *\$habitantes*, porque en medio hay otro parámetro.

Argumentos mediante un array.

Si no encontró alguna utilidad a la anterior forma de pasar valores variables, quizás necesite utilizar un array para pasarlo. El ejemplo siguiente usa esta estrategia y algunos trucos más, como el operador ternario y los array asociativos.

```
<?php
function capitales($datos) {
    $Pais = isset ($datos['Pais']) ? $datos ['Pais'] : "España";
    $Capital = isset ($datos['Capital']) ? $datos ['Capital'] : "Madrid";
    $habitantes = isset ($datos['habitantes']) ? $datos ['habitantes'] : "muchos";
    return ("La capital de $Pais es $Capital y tiene $habitantes habitantes.<br>");
}
// Introducimos en el array los datos uno por uno para que sea más fácil de entender
$datos ['Pais'] = "España";
echo capitales($datos);
$datos ['Pais'] = "Portugal";    $datos ['Capital'] = "Lisboa";
echo capitales($datos);
$datos ['Pais'] = "Francia";    $datos ['Capital'] = "Paris",    $datos ['habitantes'] = "muchísimos";
echo capitales($datos);
?>
```

Como puede ver, la función tiene un único argumento, que contiene todos los datos que nuestro programa necesita. Esta vez, los valores por defecto los introducimos mediante el operador ternario, que chequea si existe el dato referido al País, Capital o habitantes y, si no existe, se añaden los datos "España", "Madrid" y "muchos". El ejemplo utiliza la función con 0, 2 y 3 parámetros. La ventaja de este método es que podrá utilizar los argumentos que quiera y en el orden que necesite, sin encasillarse en una forma especial de pasar los datos.

Múltiples argumentos con func_num_args().

Desde la versión 4, PHP ofrece algunas funciones que pueden ser utilizadas para recuperar los argumentos. Son muy similares al lenguaje C:

- `func_num_args()`: Devuelve el número de argumentos que recibe la función desde la que es llamada.
- `func_get_arg()`: Devuelve uno a uno los argumentos pasados de la siguiente forma: `func_get_arg(0)`, `func_get_arg(1)`, `func_get_arg(5)`.
- `func_get_args()`: Devuelve un array con todos los argumentos pasados a la función, con los índices del array empezando desde 0.

Cualquiera de estas funciones dará un error si son llamadas fuera del entorno de una función, y `f u n c _ g e t _ a r g ()`, producirá un fallo si es llamada con un número más alto que los argumentos que se reciben.

Las funciones anteriores dan una ventaja a largo plazo, ya que si, durante el período de vida de una función necesita añadir algún argumento más, puede capturar sin necesidad de cambiar el código de las llamadas o el de definición de la función.

En el ejemplo siguiente puede comprobar cómo se utiliza este método:

```
<?php
function capitales() {
    $numero_argumentos = func_num_args();
    $Pais = $numero_argumentos > 0 ? func_get_arg(0) : "España";
    $Capital = $numero_argumentos > 1 ? func_get_arg(1) : "Madrid";
    $habitantes = $numero_argumentos > 2 ? func_get_arg(2) : "muchos" ;
    return ("Número de argumentos es: $numero_argumentos. La capital de $Pais es $Capital
y tiene $habitantes habitantes.<br>");
}
echo capitales();
echo capitales("Portugal", "Lisboa");
echo capitales("Francia", "Paris", "muchísimos"); ?>
```

Esta forma de utilizar los argumentos sigue teniendo una limitación. Los argumentos deben ser pasados en un lugar determinado, sino la función no hará bien su trabajo.

Utilizar los parámetros como array, es el método más flexible y el más utilizado en los programas PHP. Aún así, es muy útil cuando no sepa cuántos datos necesita manejar una función. Podemos utilizarlo para funciones que sumen todos los parámetros que introduzca o concaténen todas las palabras, como el ejemplo siguiente:

```
<?php
function concatenar() {
    $resultado = "";
    $numero_argumentos = func_num_args();
    $array_parametros = func_get_args();
    for ($x = 0; $x <= $numero_argumentos; $x++){
        $resultado = $resultado . $array_parametros[$x] ;
    }
    return $resultado."<br>" ;
}
echo concatenar("Hola ","Mundo");
echo concatenar("Esto ","es ","una ","prueba ","de","la
potencia"," de este"," método"); ?>
```

4.5 - Recursividad.

Definición de recursividad.

Un objeto es recursivo cuando forma parte de sí mismo o se define en función de sí mismo. La definición de una función recursiva consta de dos etapas:

1.- Definición de la ley de recurrencia.-

Dividir el problema en una sucesión de problemas más simples que el original y de una naturaleza similar.

2.- Definición del caso base.-

Identificar el caso más simple conocido; que es el que determinará el fin de la recursividad. También se conoce como condición de parada.

La potencia de la recursividad reside en la posibilidad de definir un número infinito de objetos mediante una fórmula o enunciado finito. Particularizando, su aplicación es especialmente apropiada cuando el problema a resolver, la función a calcular o la estructura de datos a utilizar vienen ya definidos de forma recursiva. Cuerpo general de una función recursiva en C. Los argumentos pasados a la función deben ser tales, que su cálculo aproxime la nueva llamada al caso trivial.

Veamos un ejemplo de una función recursiva. Se trata de la función factorial ($n!$) definida para enteros no negativos:

```
function nombre_funcion (tipo arg1, tipo arg2,...tipo argN) {
    if (caso trivial)
        hacer cualquier cosa menos llamada recursiva;
    else
        nombre_funcion (arg1, arg2, ...argN);
}
```

Si $n = 0$ entonces $n! = 0$

Si $n > 0$ entonces $n! = n * (n-1)!$

Definida formalmente:

- Ley de recurrencia: $n! = n * (n-1)!$
- Caso base $0! = 1$

```
function factorial ( int n)
{
    if (n == 0)
        return 1;
    else
        return n * factorial(n-1);
}
```

Ventajas e inconvenientes.

Ventajas:

- Resolución de problemas de una manera natural, sencilla y elegante.
- Facilidad para comprobar y convencerse de que la solución funciona correctamente.
- Flexibilidad: se adapta con mucha facilidad a los problemas.
- La evaluación de la complejidad de funciones recursivas es más fácil que la de los iterativos.

Para ilustrar los tres primeros puntos pondremos el ejemplo de la

función Factorial; tanto en su versión recursiva como iterativa. Del punto que trata sobre la complejidad nada se va a decir, ya que la complejidad de algoritmos es un tema muy amplio y que no podemos abordar en este curso. Sería interesante estudiarlo a fondo. De momento, nos basta con saber que es más fácil evaluar la complejidad de subprogramas recursivos que la de iterativos.

Función Factorial F(n):

$$\begin{aligned} F(n) &= n && \text{Si } n = 0 \\ F(n) &= n * F(n-1) && \text{Si } n > 0 \end{aligned}$$

Solución recursiva:

```
Function factorial ( int n){  
    if (n==0)  
        return 1;  
    else  
        return n * factorial(n-1);  
}
```

Solución iterativa:

```
long Factorial(int n){  
    long f=1;  
    int i;  
    for (i=n;i>0;i--)  
        f=f*i;  
    return f;  
}
```

Inconvenientes:

- Las soluciones recursivas son más lentas que las iterativas. Se pierde mucho tiempo en hacer las llamadas recursivas (almacenamiento de los parámetros, de la dirección de retorno, de las variables locales en cada llamada recursiva).
- Requieren más memoria (para ser usada por la pila o stack). A veces puede suceder que haya tal número de llamadas recursivas sin liberar que tengamos un desbordamiento de la pila. Aquí la memoria juega un papel muy importante.

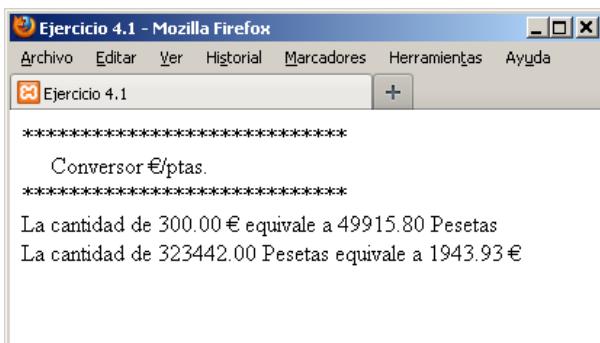
Actividades.

UD4 – ACTIVIDAD 1: Función conversora de Euros a Pesetas y viceversa.

TIPO	Desarrollo
OBJETIVOS	Familiarizarse con el uso de las funciones en PHP.

ENUNCIADO DE LA ACTIVIDAD

Realizar una función que pase de euros a pesetas y otra de pesetas a euros. Realizar un programa donde se puedan probar las dos funciones.



COMENTARIOS

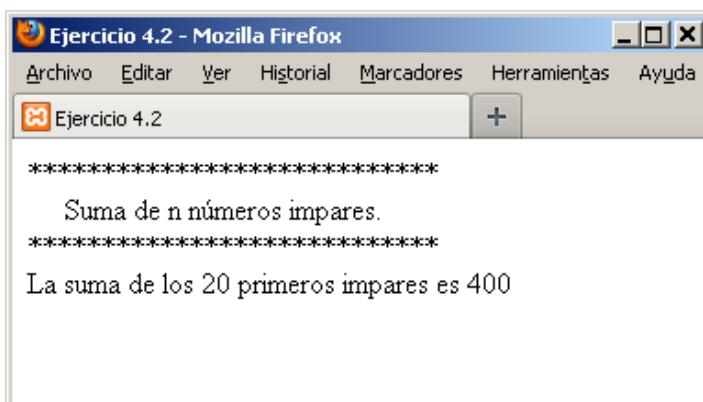
Utiliza la salida con formato print para delimitar la cantidad de decimales que se obtienen. El siguiente ejemplo imprime un número con 2 decimales:
`printf("%' 0.2f ",$valor);`

UD4 – ACTIVIDAD 2: Suma de los n primeros números de forma recursiva.

TIPO	Desarrollo
OBJETIVOS	Trabajar con las funciones recursivas para afianzar el conocimiento de este tipo de funciones.

ENUNCIADO DE LA ACTIVIDAD

Realizar un programa para hallar la suma de los n primeros números enteros impares. Llama sum.c al fichero correspondiente. En este caso, la función se puede basar en la fórmula recursiva siguiente: $\text{suma}(n) = (2*n-1) + \text{suma}(n-1)$



COMENTARIOS

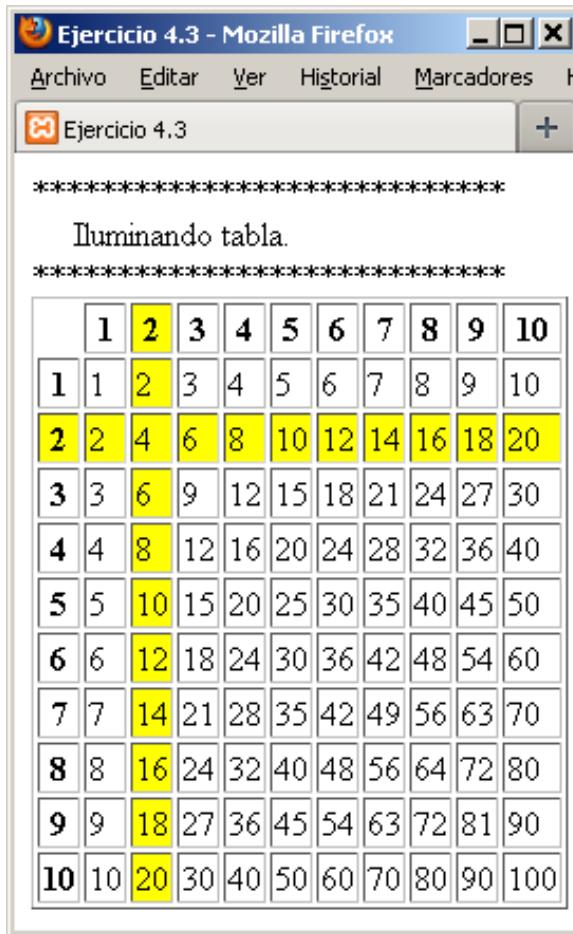
Sólo reseñar que no se trata de la suma de los impares hasta el número n que se decida, sino que es la suma de los n primeros números impares, es decir, si por ejemplo el número es 3, el resultado será 9 que es la suma de 1 más 3 más 5.

UD4 – ACTIVIDAD 3: Suma de los n primeros números de forma recursiva.

TIPO	Desarrollo
OBJETIVOS	Trabajar con las funciones para afianzar su conocimiento.

ENUNCIADO DE LA ACTIVIDAD

Crear una función que reciba un argumento numérico y nos muestre la tabla de multiplicar de los 10 primeros números con el introducido iluminado.



COMENTARIOS

Recordamos que para iluminar una celda podemos utilizar CSS o la propiedad bgcolor.

5 - Arrays.

Hasta ahora, los datos manejados en los programas han sido los denominados datos simples (numéricos, carácter,...). En numerosas ocasiones es necesario utilizar un conjunto de datos relacionados entre sí. Por ejemplo, si se quiere manipular una lista de 100 edades de personas, es conveniente tratar este conjunto de datos de forma unitaria en lugar de utilizar 100 variables, una para cada edad.

Un conjunto de datos homogéneos (del mismo tipo) que se tratan como una sola unidad se denomina **estructura de datos**. Si una estructura de datos reside en la memoria central del ordenador se denomina estructura de datos interna. Recíprocamente, si reside en un soporte externo, se denomina estructura de datos externa. Las estructuras de datos internas pueden ser de dos tipos:

- **Estáticas**: Tienen un número fijo de elementos que queda determinado desde la declaración de la estructura en el comienzo del programa.
- **Dinámicas**: Tienen un número de elementos que varía a lo largo de la ejecución del programa.

Toda estructura de datos se caracteriza por:

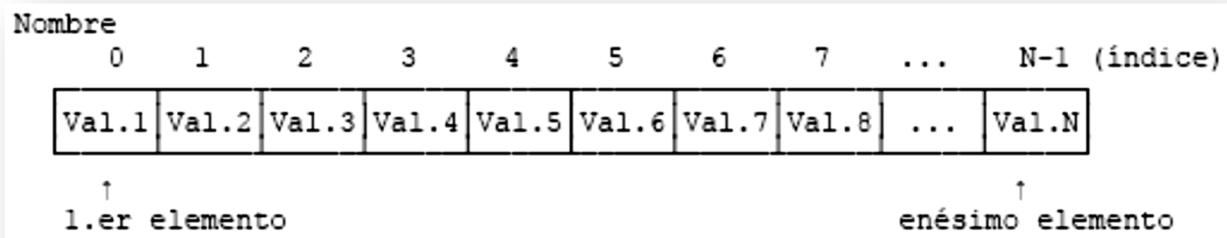
- El tipo o los tipos de los componentes
- La forma de la organización
- La forma de acceso a los componentes

5.1- Escalares y asociativos.

Un array es una estructura de datos caracterizada por:

- Estar referenciada mediante un mismo **nombre**.
- Tiene un número fijo y finito de elementos al que se denomina **longitud** o **tamaño** del array.
- Todos los componentes son del mismo tipo (homogéneo).
- Se tiene un acceso directo a sus componentes. Todos sus componentes se pueden seleccionar arbitrariamente y son igualmente accesibles mediante un **índice**. Los índices deben ser números enteros; señalan el orden relativo en que están almacenados los elementos de la tabla.
- Almacenados en posiciones de memoria físicamente contiguas, de forma que, la dirección de memoria más baja corresponde a la del primer elemento, y la dirección de memoria más alta corresponde a la del último elemento.

Según las dimensiones del array se corresponde con los conceptos matemáticos de vector (1 dimensión), matriz (2 dimensiones) y poliedro (3 o más dimensiones).

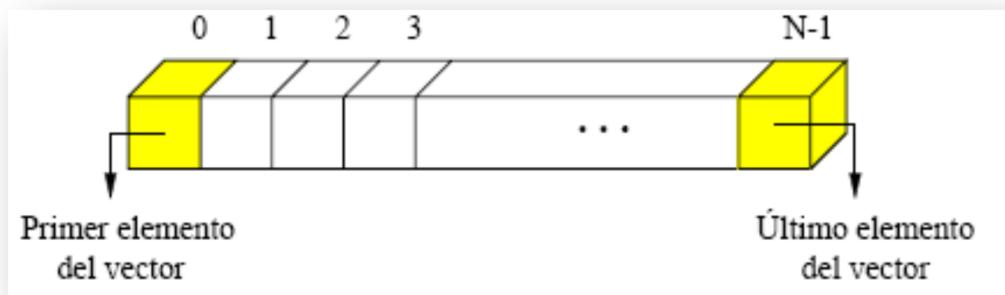


Las operaciones que se suelen realizar habitualmente sobre una tabla son:

- **Recorrido**: Procesamiento de cada elemento de la tabla.
- **Búsqueda**: Obtener la posición ocupada por un elemento con un determinado valor.

- **Ordenación:** Organizar los elementos de la tabla de acuerdo con algún criterio. Solo se realiza sobre tablas unidimensionales: vectores.

También se pueden **insertar** y/o **borrar** elementos de la tabla, pero sólo de una forma lógica, nunca física.



La sintaxis que permite definir elementos en un array es esta: **`$nombre[indice]`**. **\$nombre** utiliza exactamente la misma sintaxis empleada para definir variables, con la única particularidad de que ahora deben añadirse los corchetes y los índices.

El **índice** puede ser *un número* (habría que escribirlo dentro del corchete *sin comillas*), *una cadena* (que habría que poner en el corchete encerrada entre *comillas sencillas* –'–), o una variable PHP en cuyo caso tampoco necesitaría ir entre comillas.

Cuando los **índices** de un array son *números* se dice que es **escalar** mientras que si fueran *cadenas* se le llamaría array **asociativo**.

Arrays escalares.

Los elementos de un *array* escalar puede escribirse con una de estas sintaxis:

`$a[]=valor` ó **`$a[xx]=valor`**

En el primero de los casos PHP asigna los índices de forma automática atribuyendo a cada elemento el valor **entero siguiente** al último asignado. Si es el **primero** que se define le pondrá índice **0** (CERO).

En el segundo de los casos, seremos nosotros quienes pongamos (**xx**) el **número** correspondiente al **valor del índice**.

Si ya existiera un elemento con ese índice, se cambiaría el valor de su contenido, en caso contrario crearía un nuevo elemento del *array* y se le asignaría como valor lo especificado detrás del signo igual, que –de la misma forma que ocurría con las variables– debería ir entre comillas si fuera una cadena o sin ellas, si se tratara de números.

Arrays asociativos.

Los elementos de un *array* asociativo pueden escribirse usando la siguiente sintaxis: **`$a['indice']=valor`**

En este caso estamos obligados a escribir el nombre del índice que habrá de ser una **cadena** y debe ponerse entre comillas. Tanto en este supuesto como en el anterior, es posible –y bastante frecuente– utilizar como índice el contenido de una variable. El modo de hacerlo sería: **`$a[$ind]=valor`**

En este caso, sea cual fuere el valor de la variable `$ind`, el nombre de la variable **nunca** se pone entre comillas.

5.1 - Unidimensionales.

Mediante el uso de arrays podemos utilizar el mismo nombre para varias variables diferenciándolas entre sí mediante índices distintos

Arrays unidimensionales					
Array escalar			Array asociativo		
Variable	Indice	Valor	Variable	Indice	Valor
\$a[0]	0	Domingo	\$a['Primero']	Primero	Domingo
\$a[1]	1	Lunes	\$a['Segundo']	Segundo	Lunes
\$a[2]	2	Martes	\$a['Tercero']	Tercero	Martes
\$a[3]	3	Miércoles	\$a['Cuarto']	Cuarto	Miércoles
\$a[4]	4	Jueves	\$a['Quinto']	Quinto	Jueves
\$a[5]	5	Viernes	\$a['Sexto']	Sexto	Viernes
\$a[6]	6	Sábado	\$a['Septimo']	Septimo	Sábado

La función array().

Esta función crea un array con los valores que pase como datos de entrada.

Los índices serán añadidos automáticamente empezando desde 0. Si no asigna parámetros a array(), la función le devolverá un array vacío.

```
<?php
    $mi_array = array(2,45,76,23,65);
?>
```

El método es similar a:

```
<?php
    $mi_array[0] = 2;
    $mi_array[1] = 45;
    $mi_array[2] = 76;
    $mi_array[3] = 23;
    $mi_array[4] = 65;
?>
```

La función array() permite también añadir índices a los valores que se introducen. Para ello se utiliza el operador => de esta forma:

```
<?php
    $mi_array = array{0 => 2, 1 => 45, 2 => 76};
?>
```

También es posible añadir índices que no sean correlativos o índices alfanuméricos, incluso mezclar los dos tipos.

```
<?php
    $mi_array = array("cero" => 2, "uno" => 45, 2 => 76);
?>
```

Para recuperar cualquier valor se utiliza el índice dentro de los corchetes:

```
<?php
    $mi_array = array("cero" => 2, "uno" => 45, 2 => 76);
    echo $mi_array [ "uno" ]. "<br>";
    echo $mi_array[2] ;
?>
```

5.3 - Bidimensionales.

Los *arrays bidimensionales* pueden entenderse como algo muy similar a una *tabla de doble entrada*.

Cada uno de los elementos se identifica –sigue siendo válido el nombre único que se usaba en los unidimensionales – por un nombre (*\$nombre*) seguido de dos (*[]*) que contienen los *índices* (en este caso son dos índices) del array.

Los *índices* pueden ser de tipo **escalar** –equivalentes al número de fila y columna que la **celda** ocupa en la tabla– o puede ser **asociativos** lo que equivaldría en alguna medida a usar como índices los *nombres de la fila y de la columna*.

En este supuesto, también, se empiezan a numerar los arrays escalares a partir de **CERO**.

Arrays escalares.

Los elementos de un **array bidimensional** escalar pueden escribirse usando una de estas sintaxis:

`$a[][]=valor` o `$a[xx][]=valor` o `$a[][xx]=valor` o también `$a[xx][yy]=valor`

En el primero de los casos PHP asigna automáticamente **comoprimer índice** el valor que sigue al último asignado y, si es el **primero** que se define, le pondrá como índice **0** (CERO)

Sea cual fuere el valor de primer índice al **segundo** se le asignará **cero** ya que es en este mismo momento cuando se habrá creado el **primero** y, por tanto, *aún carecerá de elementos*.

En el segundo de los casos, asignamos un valor al **primer índice(xx)** y será el segundo quien se incremente en **una unidad** respecto al de valor más alto de todos aquellos cuyo **primer índice** coincide con el especificado.

La tercera opción es bastante similar a la anterior. Ahora se modificaría automáticamente el primer índice y se escribiría el contenido (xx) como valor del segundo. En la cuarta de las opciones se asignan libremente cada uno de los índices (**xx e yy**) poniéndoles valores numéricos.

Arrays asociativos.

Los elementos de un **array asociativo bidimensional** se pueden escribir usando la siguiente sintaxis:

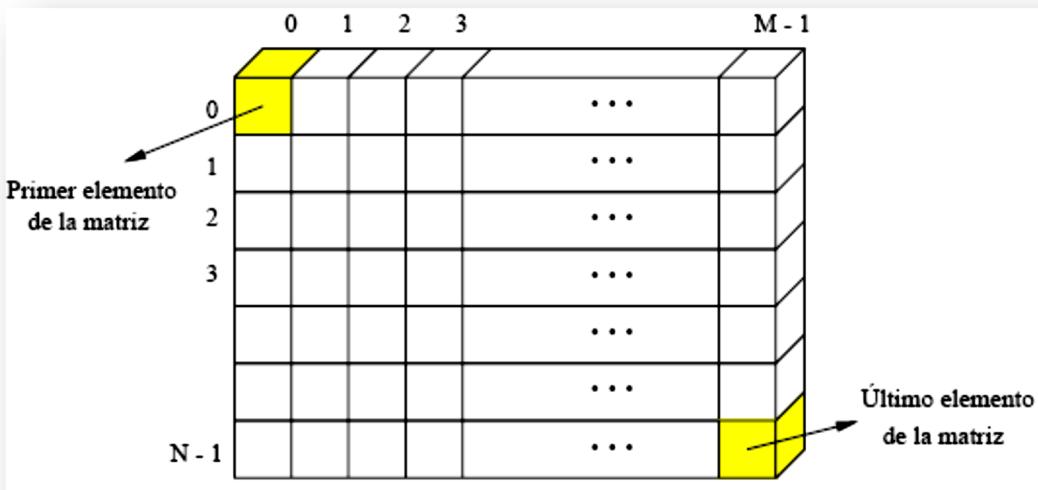
`$a["indice1"]["indice2"]=valor`

En este caso, los índices serán **cadenas** y se escribirán entre comillas.

Arrays mixtos.

PHP permite utilizar también **arrays mixtos**. Sería este el caso de que uno de ellos fuera escalar y el otro asociativo.

Igual que ocurría con los unidimensionales, también aquí podemos utilizar valores de variables como índices.



5.4 - Multidimensionales.

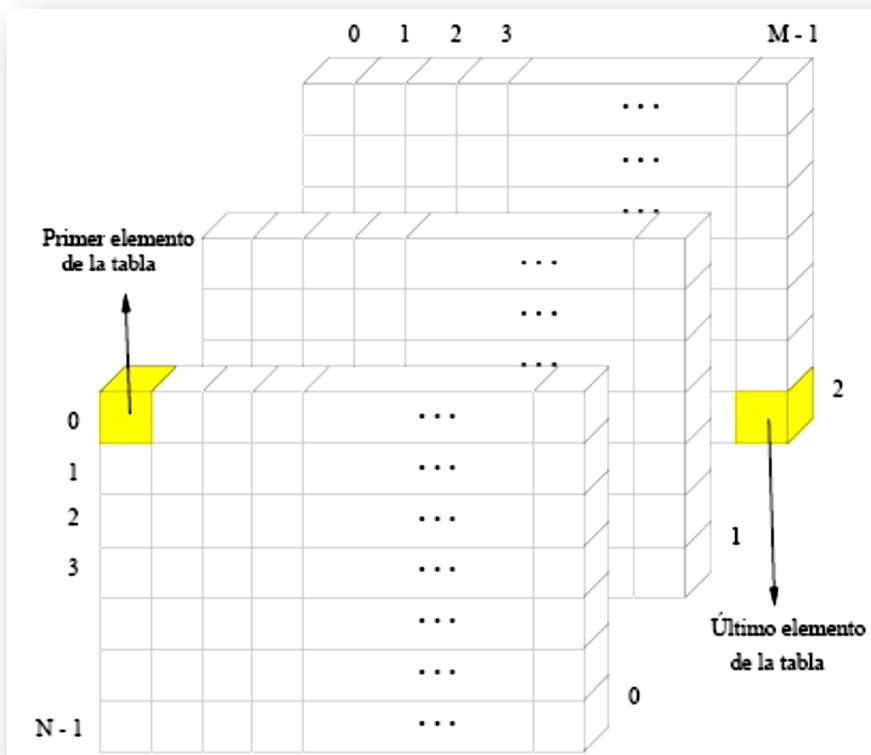
PHP permite el uso de arrays con dimensión superior a dos. Para modificar la dimensión del array basta con ir añadiendo nuevos índices.

`$a[x][y][z]=valor;`

asignaría un valor al elemento de índices **x**, **y** y **z** de un array *tridimensional* y

`$a[x][y][z][w]=valor;`

haría lo mismo, ahora con un array de dimensión **cuatro**. Pueden tener cualquier tipo de *índices*: *escalares*, *asociativos* y, también, *mixtos* :



5.5 – Propiedades de arrays.

Existen numerosas funciones que son capaces de averiguar datos de los *arrays*, tales como el tamaño, si un valor forma parte del conjunto o si un determinado índice está registrado.

count()

Cuenta el número de elementos que contiene un *array*.

```
<?php
    echo "elementos de 1 dimensión " . count ($colores) . "<br>";
    echo "elementos de 2 dimensiones " . count($colores["fuertes"]);
?>
```

El ejemplo anterior se basa en la definición del *array* de colores inicializado antes. Para contar el número de elementos debe ponerse también la dimensión. Si el *array* es de una sola dimensión no hace falta. La función **sizeof()** actúa de la misma forma.

in_array()

Busca dentro de un *array* un valor pasado como parámetro y, si lo encuentra devuelve el valor *true*, si no, devuelve *false*. Toma dos argumentos, el valor a buscar y el *array* dónde buscar.

```
<?php
    $colores = array ("rojo","verde","amarillo","azul");
    if (in_array("rojo",$colores)) {
        echo "Se ha encontrado el valor rojo";
    } else {
        echo "No se ha encontrado" ;
    }
?>
```

5.6 – Interactuar con arrays.

Borrar elementos.

Para borrar un elemento, simplemente se utiliza la misma función que borra las variables definidas: **unset()**.

```
<?php
    $colores = array ("rojo","verde","amarillo","azul","rosa") ;
    echo "El número de elementos de colores es: " . count($colores) . "<br>";
    unset ($colores[2]);
    echo "El número de elementos de colores es: " . count($colores) . "<br>";
?>
```

Se puede observar que para borrar un elemento, hay que conocer el índice. El resultado del ejemplo por pantalla es:

El número de elementos de colores es: 5

El número de elementos de colores es: 0.

Recorrer un array.

Podemos utilizar la estructura **foreach** para tal propósito. Veamos un ejemplo:

```
<?php
    $ciudades = array ("Badajoz","Mérida","Cáceres","Plasencia") ;
    foreach ($colores as $valor) {
        echo ("El valor es $valor<br>");
    }
?>
```

La construcción anterior recorre el *array* desde el principio. En el ejemplo se puede ver que **foreach** toma el *array* a recorrer y sus valores los va almacenando en la

variable \$valor a medida que el bucle se ejecuta. Existe una segunda construcción que permite recuperar el índice y el valor. Veamos un ejemplo de esto:

```
<?php
    $ciudades = array ("Badajoz","Mérida","Cáceres","Plasencia");
    foreach ($ciudades as $índice => $valor) {
        echo ("El índice $índice tiene el valor: $valor<br>") ;
    }
?>
```

Desplazarse por un array.

Cada vez que creamos un *array* dentro de un programa PHP, se crea un puntero que permite recorrer en su totalidad el conjunto de valores. Este puntero se inicializa al valor inicial del *array*. La función **current()** devuelve el valor al que apunta el puntero. La función **next()** hace avanzar el puntero una posición en el conjunto de datos.

Si el puntero se encuentra al final del conjunto **next()** devuelve un valor **false**. El ejemplo siguiente muestra cómo utilizar estas funciones para recorrer un *array*:

```
<?php
$ciudades = array ("Badajoz","Mérida","Cáceres","Plasencia");
$ciudades["España"] = "Madrid";
$ciudades["Portugal"] = "Lisboa";
$ciudades["Francia"] = "Paris";
do {
    $valor = current($ciudades);
    echo ("El valor es: $valor<br>");
}while (next($ciudades) ) ;
?>
```

Una vez recorrido *el array*, el puntero queda fijado al final del conjunto de datos. Para volver al principio se puede utilizar la función **reset()**, que envía al puntero al principio.

La función **prev()** retrocede una posición el puntero y la función **end()** se coloca al final de una lista de valores. Además, para obtener el valor del índice puede utilizar la función **key()**.

Intercambio de valores.

La función **array_flip()** intercambia los valores de índices y datos, es decir, los índices serán guardados como datos y los valores serán sus nuevos índices.

```
<?php
function recorre ($numero) {
    foreach ($numero as $indice => $valor) {
        echo "$indice: $valor<br> , -"
    }
}
$numero = array("uno" => 1,"dos" => 2, "tres" => 3,"cuatro" => 4);
echo ("Números<br> ");
recorre($numero);
echo ("Números intercambiados<br> ");
recorre(array_flip{$numero});
?>
```

El resultado es el esperado:

Números	Números intercambiados
uno: 1	1: uno
dos : 2	2 : dos
tres: 3	3: tres
cuatro: 4	4: cuatro

Inversión de contenido.

También es posible ordenar a la inversa una lista de datos. Esto se consigue con la función **array_reverse()**. Si modifica el código anterior tendrá:

```
echo ("Números intercambiados<br>") ;
recorre(array^reverse($numero));
```

Dando como resultado:

Números	Números invertidos
uno: 1	cuatro: 4
dos : 2	tres: 3
tres: 3	dos: 2
cuatro: 4	uno: 1

5.5 – Ordenación de arrays.

Finalmente, PHP ofrece una gran variedad de funciones para ordenar arrays. Las funciones pueden verse en la tabla siguiente:

Función	Explicación
asort()	Ordena de forma ascendente el array pasado como argumento. Ordena las parejas índice/valor atendiendo al dato. Es un buen método para los arrays asociativos.
arsort()	Igual que asort(), pero ordena en sentido descendente.
ksort()	Ordena de forma ascendente el array pasado como argumento. Ordena las parejas índice/valor atendiendo esta vez al índice.
krsort()	Igual que ksort (), pero ordena en sentido descendente.
sort()	Ordena de forma ascendente el array pasado como argumento. Se pierde el valor asociativo entre el índice y el valor.
rsort()	Igual, pero en orden descendente.

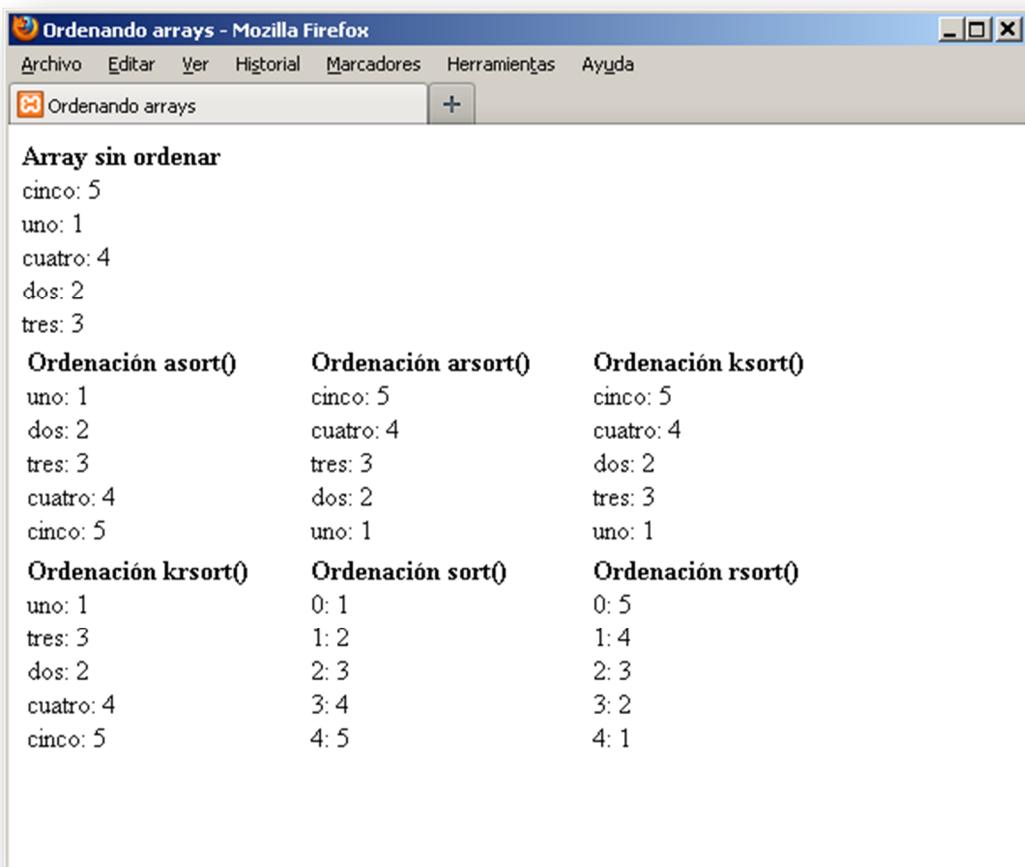
Puede ver actuar todas las funciones de ordenación en el ejemplo siguiente:

```
<?php
function recorre ($numero) {
    foreach ($numero as $indice => $valor) {
        echo "$indice: $valor<br>";
    }
}
$pila = array("cinco"=>5,"uno"=>1,"cuatro"=>4,"dos"=>2,"tres"=>3) ;
echo "<HTML><HEAD><TITLE>Ordenando arrays</TITLE></HEAD><BODY>" ;
    echo "<B>Array sin ordenar<br></B>" ;
recorre($pila);
echo "<TABLE width='500'><TR>" ;
echo "<TD>" ;
    echo "<B>Ordenación asort()<br></B>" ;
asort($pila);
recorre($pila);
echo "</TD><TD>" ;
    echo "<B>Ordenación arsort()<br></B>" ;
```

```

arsort($pila);
recorre($pila);
echo "</TD><TD>";
    echo "<B>Ordenación ksort()<br></B>";
ksort($pila);
recorre($pila);
echo "</TD></TR>";
echo "</TR><TD>";
    echo "<B>Ordenación krsort()<br></B>";
krsort ($pila);
recorre($pila);
echo "</TD><TD>";
    echo "<B>Ordenación sort()<br></B>";
sort($pila);
recorre($pila);
echo "</TD><TD>";
    echo "<B>Ordenación rsort()<br></B>";
rsort($pila);
recorre($pila);
echo "</TD>";
echo "</TR></TABLE>";
echo "</BODY></HTML>";
?>

```



Actividades.

UD5 – ACTIVIDAD 1: Máximo elemento del conjunto	
TIPO	Desarrollo
OBJETIVOS	Ilustrar el funcionamiento de los arrays en PHP.
ENUNCIADO DE LA ACTIVIDAD	
<p>Realiza un programa que calcule el máximo entre un conjunto de números enteros que previamente han sido almacenados en un array.</p> <p><u>Indicaciones:</u> para averiguar el máximo, se crea una variable llamada \$max, a la que se da inicialmente el valor de \$conjunto[0]. Luego se recorren paso a paso todos los elementos del vector, comparando el valor almacenado en la posición considerada del vector con el valor de la variable \$max. Si el valor de la posición considerada del vector es mayor que \$max, entonces se copia (se sustituye el valor) en la variable \$max este valor. De esta forma, una vez recorrido todo el vector, la variable max contendrá el máximo valor.</p>	
COMENTARIOS	
<p>Prestar especial atención a la forma por la que se consigue conocer la posición del valor máximo, que no es otra que guardando la información sobre el índice. Esto se realiza a través de una variable temporal. Se puede ver el contenido directamente del array a través de la instrucción <code>printf("%d: ", conjunto[\$temporal]);</code></p>	

UD5 – ACTIVIDAD 2: Día de la semana.	
TIPO	Desarrollo
OBJETIVOS	Practicar el uso de funciones y arrays.
RECURSOS	Editor de texto y navegador web.
ENUNCIADO DE LA ACTIVIDAD	
<p>Crea una función que maneje un array en su interior con los días de la semana. Esta función recibe un número entre el 1 y el 7 y devuelve el nombre del día de la semana. Si el número es diferente a los indicados, no hará nada.</p>	

6 – Cadenas.

Si pensamos en los tipos de datos que circulan por la red, llegaremos a la conclusión de que una gran porción la ocupan las imágenes, animaciones en Macromedia Flash, videos o subprogramas escritos en Java. La otra gran porción, mayoritaria, son los textos o las cadenas de caracteres.

Las cadenas de caracteres o *string* son secuencias de caracteres que pueden ser tratadas como una unidad, asignadas a variables, pasadas como parámetros a funciones o enviadas como salida al navegador. Un *string* se diferencia de otro tipo de dato en PHP 5, porque va encerrado entre comillas dobles (") o simples (').

"Cadena entre comillas dobles"
'Cadena entre comillas simples'

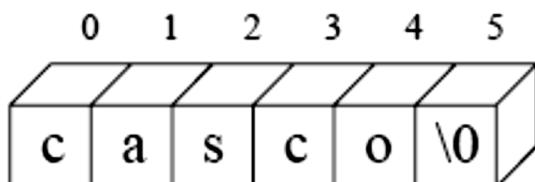
PHP interpreta de distinta forma las cadenas que van entre comillas dobles y las que van entre comillas simples. Los strings entre comillas dobles pueden sustituir ciertos símbolos por acciones, como la inclusión del valor de una variable. Las comillas simples, simplemente muestran todo el contenido, sin atender a caracteres especiales. Podemos ver un ejemplo donde sucede esto:

```
<?php
$variable = "Domingo";
$frase_1 = "Hoy es $variable, el cielo está gris";
$frase_2 = 'Hoy es $variable, el cielo está gris';
echo $frase_1;
echo $frase_2;
?>
```

El resultado en el navegador es:

Hoy es Domingo, el cielo está gris
Hoy es \$variable, el cielo está gris

Como se puede observar, la cadena \$frase_1 es capaz de sustituir la variable por su valor, por el simple hecho de estar entre comillas dobles.



6.1 – Propiedades de las cadenas.

Índices de un string.

Si pensamos en las cadenas como una sucesión de caracteres en un orden determinado, podemos llegar a desear acceder libremente a parte de los caracteres. Esto

es posible gracias a los símbolos de llave ({ }) y un índice numérico que se corresponderá con la posición del carácter que buscamos.

El ejemplo muestra cómo crear una función que duplica las letras de una cadena aprovechando esta forma de acceder a los caracteres:

```
<?php
function duplicar_caracteres($cadena) {
    $tamanio = strlen ($cadena);
    $cadena_auxiliar = "";
    for ($x = 0;$x < $tamanio; $x++) {
        $cadena_auxiliar = $cadena_auxiliar . $cadena{$x} . $cadena{$x};
    }
    return $cadena_auxiliar;
}
$cadena = "Duplicar las letras";
echo duplicar_caracteres($cadena);
?>
```

Operadores.

En este punto, aprovecharemos para hacer un breve repaso de los operadores de *string* vistos en el capítulo 3. En otros lenguajes, como Java, se utiliza el operador suma (+) para unir dos cadenas. En PHP este mismo resultado se obtiene con el operador punto (.). Así, podemos concatenar varias cadenas de la forma siguiente:

```
<?php
$cadenal = "Hola";
$cadena2 = "Mundo";
$cadena3 = "¡Qué típico!";
$supercadena = $cadenal . " " . $cadena2 . " " . $cadena3;
echo $supercadena;
?>
```

Es fácil intuir que el operador puede concatenar caracteres y variables de tipo *string* de forma conjunta. Es posible que desee ahora añadir texto a una cadena ya existente; esto se puede hacer de dos formas muy similares. La primera es asignando a la variable su valor más el valor a añadir, de la siguiente forma:

```
<?php
$cadenal = "Hola";
$cadena2 = "Mundo";
$cadenal = $cadenal . $cadena2;
echo $cadenal;
?>
```

O utilizando el operador de concatenación y asignación (.=), como en el ejemplo:

```
<?php
$cadenal = "Hola";
$cadena2 = "Mundo";
$cadenal .= $cadena2;
echo $cadenal;
?>
```

6.2 – Funciones de string.

Algunas de las funciones que permiten manejar los formatos de las cadenas de caracteres son estas:

chr(*n*) Devuelve el carácter cuyo código ASCII es *n*.

ord(*cadena*) Devuelve el código ASCII del primero de los caracteres de la cadena.

strlen(*cadena*) Devuelve la *longitud* (número de caracteres) de la cadena. Los espacios son considerados como un carácter más.

strtolower(*cadena*) Cambia todos los caracteres de la cadena a *minúsculas*.

strtoupper(*cadena*) Convierte en *mayúsculas* todos los caracteres de la cadena.

ucwords(*cadena*) Convierte a *mayúsculas* la *primera letra* de cada palabra.

ucfirst(*cadena*) Convierte a *mayúsculas* la primera letra de la cadena y pone en *minúsculas* todas las demás.

ltrim(*cadena*) Elimina todos los *espacios* al *principio* de la cadena.

rtrim(*cadena*) Elimina todos los *espacios* que existieran al *final de la cadena*.

trim(*cadena*) Elimina los *espacios* tanto al *principio* como al *final* de la cadena.

chop(*cadena*) Elimina los *espacios* al final de la cadena. Es *idéntica* a **rtrim**.

Cuidado: Tanto **trim**, como **ltrim** y **rtrim** eliminan, además de los espacios, las secuencias: \n, \r, \t, \v y \0; llamadas también *caracteres protegidos*.

substr(*cadena,n*) Si el valor de *n* es positivo extrae todos los caracteres de la cadena a partir del que ocupa la posición *nésima* a contar desde la izquierda. Si el valor de *n* es negativo serán extraídos los *n* últimos caracteres contenidos en la cadena.

substr(*cadena,n,m*) Si *n* y *m* son positivos extrae *m* caracteres a partir del que ocupa la posición *nésima*, de izquierda a derecha. Si *n* es negativo y *m* es positivo extrae *m* (contados de izquierda a derecha) a partir del que ocupa la posición *nésima* contada de derecha a izquierda. Si *n* es positivo y *m* es negativo extrae la cadena comprendida entre el *nésimo* carácter (contados de izquierda a derecha) hasta el *negativo*, contando en este caso de derecha a izquierda. Si *n* es negativo y *m* también es negativo extrae la porción de cadena comprendida entre el *emésimo* y el *enésimo* carácter contando, en ambos casos, de derecha a izquierda. Si el valor absoluto de *n* es menor que el de *m* devuelve una cadena vacía.

strrev(*cadena*) Devuelve la cadena invertida.

str_repeat(*cadena, n*) Devuelve la cadena **repetida** tantas veces como indica *n*.

str_pad(*cad, n, rell, tipo*) Añade a la cadena *cad* los caracteres especificados en *rell*(uno o varios, escritos entre comillas) hasta que alcance la longitud que indica *n* (un número). El parámetro **tipo** puede tomar uno de estos tres valores (sin comillas):

6 – Cadenas.

STR_PAD_BOTH (rellena por ambos lados)
STR_PAD_RIGHT (rellena por la derecha)
STR_PAD_LEFT (rellena por la izquierda).

Si se omite la cadena de *Relleno* utilizará *espacios* y si se omite el *tiporellenará* por la *derecha*.

He aquí algunos ejemplos de aplicación de las funciones de manejo de cadenas:

Código ASCII y viceversa		
Función	Ejemplo	Resultado
<code>chr(código ASCII)</code>	<code>chr(97)</code>	a
<code>ord("cadena")</code>	<code>ord("abadesa")</code>	97
Longitudes y conversiones mayúsculas/minúsculas		
Función	Ejemplo	Resultado
<code>strlen("cadena")</code>	<code>strlen("Mide la longitud de esta cadena")</code>	31
<code>strtolower("cadena")</code>	<code>strtolower("CONVIERTA A MINÚSCULAS")</code>	convierte a minúsculas
<code>strtoupper("cadena")</code>	<code>strtoupper("pasa a mayúsculas")</code>	PASA A MAYÚSCULAS
<code>ucwords("cadena")</code>	<code>ucwords("todas empiezan por mayúscula")</code>	Todas Empiezan Por Mayúscula
<code>ucfirst("cadena")</code>	<code>ucfirst("mayúscula al principio")</code>	Mayúscula al principio
Eliminar espacios		
Función	Ejemplo	Resultado
<code>ltrim("cadena")</code>	<code>ltrim("\n \nEliminar espacios")</code>	Eliminar espacios
<code>rtrim("cadena")</code>	<code>rtrim("Eliminar espacios\n \n")</code>	Eliminar espacios
<code>trim("cadena")</code>	<code>trim("\n \nEliminar espacios\n \n")</code>	Eliminar espacios
<code>chop("cadena")</code>	<code>chop("\n \nEliminar espacios\n \n")</code>	Eliminar espacios
Extraer porciones de una cadena		
Función	Ejemplo	Resultado
<code>substr("cadena",n)</code>	<code>substr("Extrae caracteres",3)</code>	rae caracteres
<code>substr("cadena",n)</code>	<code>substr("Extrae caracteres",0)</code>	Extrae caracteres
<code>substr("cadena",n)</code>	<code>substr("Extrae caracteres",-5)</code>	teres
<code>substr("cadena",n)</code>	<code>substr("Extrae caracteres",-2)</code>	es
<code>substr("cadena",n,m)</code>	<code>substr("Extrae caracteres",2,6)</code>	trae c
<code>substr("cadena",n,m)</code>	<code>substr("Extrae caracteres",0,8)</code>	Extrae c
<code>substr("cadena",n,m)</code>	<code>substr("Extrae caracteres",2,-3)</code>	trae caracte
<code>substr("cadena",n,m)</code>	<code>substr("Extrae caracteres",-7,5)</code>	acter
<code>substr("cadena",n,m)</code>	<code>substr("Extrae caracteres",-7,-5)</code>	ac
<code>substr("cadena",n,m)</code>	<code>substr("Extrae caracteres",-5,-7)</code>	

Modificaciones de cadenas		
Función	Ejemplo	Resultado
<code>strrev("cadena")</code>	<code>strrev("Invierte la cadena")</code>	anedac al etreivnl
<code>str_repeat("cadena",n)</code>	<code>str_repeat("Rep",5)</code>	RepRepRepRepRep
<code>str_pad("cadena",n,"Relleno",Tipo)</code>	<code>str_pad("Pepe",10,"*",STR_PAD_BOTH)</code>	***Pepe***
<code>str_pad("cadena",n,"Relleno",Tipo)</code>	<code>str_pad("Pepe",10,"*",STR_PAD_LEFT)</code>	*****Pepe
<code>str_pad("cadena",n,"Relleno",Tipo)</code>	<code>str_pad("Pepe",10,"*",STR_PAD_RIGHT)</code>	Pepe*****
<code>str_pad("cadena",n,"Relleno",Tipo)</code>	<code>str_pad("Pepe",10,"")</code>	Pepe*****
<code>str_replace ("lo que dice",lo que dira,"Cadena")</code>	<code>str_replace("e","a","Pepe")</code>	Papa
<code>str_replace ("lo que dice",lo que dira,"Cadena")</code>	<code>str_replace("pe","pa","Pepepe")</code>	Pepapa
<code>str_replace ("lo que dice",lo que dira,"Cadena")</code>	<code>str_replace("Pepe","Luis","Pepe")</code>	Luis
<code>substr_replace ("Cadena",lo que dira,n,m)</code>	<code>substr_replace("Pepe","Luis",2,-1)</code>	PeLuise

AddSlashes(cadena) Inserta el carácter \ delante los siguientes: ", ', \ y **NUL** (el bit nulo).

stripslashes(cadena) Quita las marcas añadidas a una cadena con la función **AddSlashes()**.

chunk_split(cad, n, sep) Devuelve la cadena (**cad**) después de haberle insertado, cada **n**caracteres, la cadena indicada en el parámetro **sep**. Si no se indica **sep** PHP pondrá un **espacio**. Si no se establece el parámetro **n** insertará el separador cada **76** caracteres. Esta función coloca siempre *un separador al final de la cadena*.

parse_str(cadena) Devuelve las variables –con su valor– indicadas dentro de la cadena (observa la sintaxis del ejemplo). Dentro de la cadena cada variable se denomina con un nombre que va seguido de un signo igual. Los espacios se señalan con el signo + y los separadores de variables son signos &

explode(sep, cad,n) Devuelve un **array** cuyos elementos contienen cada una de las porciones de la cadena (**cad**) comprendidas entre dos de los caracteres señalados como (**sep**) hasta el máximo de porciones señaladas (**n**). Los caracteres separadores no son incluídos en las cadenas resultantes. Si no se indica la cantidad de porciones, será fraccionada toda la cadena. Si se indica número, el último trozo contendrá **toda** la cadena restante.

implode(sep, array) Devuelve una cadena formada por todos los elementos del **array**separados mediante los caracteres indicados en **sep**.

join(sep, array) Es *idéntica* a **implode**.

strtok(cad , sep) Esta función divide la cadena **cad**entrozos delimitados por el separador que se indica en **sep**. Cuando se invoca la primera vez –extrae el primer trozo– debe llevar las sintaxis **strtok(cadena,sep)**. Al invocarla sucesivamente, se escribe solo **strtok** (" ") e irá recogiendo de forma secuencial los trozos sucesivos.

Encriptación de cadenas

PHP dispone de funciones que permiten *codificar* o *encriptar* cadenas de caracteres.

bin2hex(cadena) Devuelve una cadena ASCII que contiene la representación **hexadecimal** de la *cadena*. La conversión se realiza byte a byte, con los 4 bits superiores primero.

crypt(cadena) Devuelve la cadena *encriptada* utilizando una *semilla aleatoria* de dos caracteres. Por su carácter aleatorio, si se ejecuta dos veces seguidas –tal como puedes observar en el ejemplo– dará dos resultados diferentes.

crypt(cadena,"xx") Devuelve la cadena *encriptada* utilizando como *semilla* los dos caracteres (entre comillas) que se escriben como segundo parámetro de la función. Tanto en este supuesto como en el anterior, los dos primeros caracteres de la cadena encriptada coinciden con los que han sido utilizados como *semilla*.

md5(cadena,"xx") Aplica el algoritmo *md5* –lo comentamos a la derecha– y devuelve la *huella digital* generada por él.

crc32(cadena) Aplica el algoritmo *crc32* *decomprobación de integridad* y devuelve el valor del mismo.

Se utiliza muchísimo en los programas de *compresión* y *descompresión* de ficheros. Se aplica en el momento de *comprimir* y se incluye el valor obtenido dentro del fichero comprimido. Después de la *descompresión* se vuelve a aplicar el mismo algoritmo y se comparan ambos valores. La coincidencia será la garantía de que el fichero obtenido es idéntico al original.

6.3 – Expresiones regulares.

Las expresiones regulares en PHP son una potente herramienta de programación que muchas veces se deja de lado ya que se cree que es muy compleja (aunque sí lo puede ser). En este post explicamos de manera sencilla cómo utilizar las expresiones regulares en PHP y brindamos soluciones a ejercicios cotidianos de programación utilizando expresiones regulares. También puede servir como un pequeño recordatorio.

La única salvedad que debemos hacer es que se debe considerar muy bien cuando usar una expresión regular, y no usarlas en exceso. Esto ya que el procesamiento de una expresión regular es considerablemente más costoso que una simple búsqueda de strings (utilizando strstr por ejemplo).

La idea, entonces, de una expresión regular es crear un string especial para hacer una búsqueda en otro string. Si nuestra expresión regular encaja (hace “match”) en el string, la operación es exitosa. Existe algunos caracteres que tienen un significado especial en una expresión regular:

Metacaracteres

- . Match con cualquier carácter
- ^ Match al principio del string
- \$ Match al final del string

6 – Cadenas.

\s	Match con cualquier espacio en blanco
\d	Match con cualquier dígito
\D	Match con cualquier carácter que no sea un dígito
\w	Match con cualquier carácter que pueda ser parte de una palabra (letra, número, guión bajo)
\W	Match con cualquier carácter que NO pueda ser parte de una palabra (letra, número, guión bajo)
\A	Inicio de un string.
\z	Final de un string.

Cuantificadores

*	el carácter puede aparecer cero o más veces.
+	el carácter puede aparecer una o más veces.
?	el carácter puede aparecer cero o una vez.
{n}	el carácter aparece exactamente n veces.
{n,}	el carácter aparece n o más veces.
{n,m}	el carácter puede aparecer entre n y m veces.

Como parte de la notación, siempre encerramos las expresiones regulares entre /, llaves ({}) o #. Por ejemplo, la expresión /ab?c/ hace match con ac y abc. La expresión regular /ab{1,3}c/ hace match con abc, abbccy abbbc.

Agrupadores

[]	permiten agrupar creando rangos, por ejemplo /ab[0-5]+c/ hará match con cualquier string que contenga ab, una o más veces un número entre 0 y 5, y finalmente una c. Por ejemplo: ab12c.
()	Nos permiten crear sub-expresiones, expresiones regulares contenidas dentro de otras: /a(bc.)+e/. Tiene un uso especial en formas como (...), que permite capturar todo lo que encierran los paréntesis, y (a b) que hace match con a o b

Modificadores

i	Coincidir indistintamente entre mayúsculas y minúsculas.
m	Match multilínea.
s	El metacaracter . hará match también con el carácter de cambio de línea.
u	Hacer los matches en modo UTF8
x	Ignorar espacios.

Otros modificadores X,e,A,D,S,U,J [ver más](#)

Permiten cambiar el modo en que se ejecute la expresión regular. Se agregan después del delimitador de cierre.

Funciones PHP

preg_match

Nos permite evaluar si un string hace match con una expresión regular. Por ejemplo, para validar un email haríamos lo siguiente :

```
function verificar_email($email)
{
    if(preg_match("/^([a-zA-Z0-9])+([a-zA-Z0-9\._-])*@([a-zA-Z0-9_-])+([a-zA-Z0-9\._-]+)+$/,$email))
    {
        return true;
    }
    return false;
}
```

Otros ejemplos interesantes pueden ser:

```
//1. verificar si un password es seguro
function verificar_password_strenght($password)
{
    if (preg_match("/^.*(?=.{8,})(?=.*\d)(?=.*[a-z])(?=.*[A-Z]).*$/", $password))
        echo "Su password es seguro.";
    else
        echo "Su password no es seguro.";
}

//2. Verificar el formato de una IPv4
function verificar_ip($ip)
{
    return preg_match("/^([1-9]|1[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])".
        "(\.(0-9|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])){3}$/, $ip );
}

//3. Verificar formato de número telefónico en EU
function verificar_telefono_eu($telefono)
{
    $regex = '/^(?:1(?:[.-])?)?(?:\((?:=\d{3})\))?(?:([2-9]\d{2})'.
        '!(?:($0', $text);
}
```

Al usar preg_replace se vuelve muy útil utilizar las retroreferencias. Estas son simplemente una sintaxis para hacer referencia a los matches que ocurrirán al ejecutar una expresión regular. Por ejemplo, para convertir una fecha en formato MM/DD/YYYY a formato DD/MM/YYYY podemos utilizar retroreferencias:

```
function cambiar_formato_fecha($fecha)
{
    return preg_replace("/([0-9]{4})V([0-9]{2})V([0-9]{2})/i", "$3/$2/$1", $fecha);
}
```

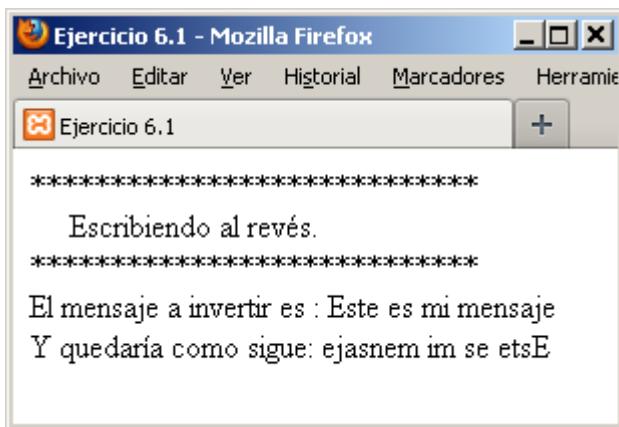
Actividades.

UD6 – ACTIVIDAD 1: Escribiendo al revés

TIPO	Desarrollo
OBJETIVOS	Mostrar el funcionamiento de las cadenas en PHP y su semejanza con los arrays.

ENUNCIADO DE LA ACTIVIDAD

Crear un programa para leer una palabra, contar las letras y escribirla al revés.
Indicaciones: lo primero que hay que hacer es pedir la palabra y almacenarla. Una vez que tenemos la palabra en un array, para contar el número de letras se chequea letra por letra (bucle while) sumando 1 cada vez. Este bucle finaliza cuando el carácter leído es el último carácter (el tamaño lo puedes averiguar con strlen()). Para escribir la palabra al revés, basta con leerla empleando un bucle que cuente hacia atrás.



COMENTARIOS

Para comprobar que lo has hecho bien puedes utilizar la función strrev();

7 – Formularios.

Hasta ahora hemos visto como operar con PHP en un solo script y procesar la información según nuestras necesidades. En este tema estudiaremos la forma de comunicar con el exterior, más concretamente mediante formularios HTML.

Existen diferentes formas de pasar la información entre páginas web, las más conocidas son utilizando ficheros, variables sesión, cookies, empleando argumentos GET y por el POST. Estas dos últimas son comúnmente utilizadas por los formularios.

7.1 – Utilizando GET.

Los argumentos **GET** pasan la información como parte de la URL. Cuando utilice esta forma de trabajar, las parejas de variable / valor podrán verse en la casilla de dirección del navegador. Para utilizarlas debemos escribir el nombre de la página Web seguido de un símbolo de interrogación (?) y el conjunto de parejas de variable / valor separadas del símbolo (&). Por ejemplo una dirección como `script.php?variable1=contenido1&variable2= contenido2` lo que hace es pasarle a la página web `script.php` los argumentos `variable1` con valor `contenido1` y `variable 2` con valor `contenido2`.

Para indicarle a un formulario que queremos utilizar el método GET para comunicar, debemos especificarlo en el atributo **method** del tag `<FORM>`, tal como sigue:

```
<form name="formulario" method="GET" action="destino.php" >
```

Donde `destino.php` sera la página PHP que procesará el formulario. Podemos hacer uso del siguiente código para leer todas las variables y sus contenidos pasados por GET.

```
<html>
<body>
<?php
    echo "Variables pasadas mediante GET:<br>";
    foreach ($_GET as $indice => $valor) {
        echo "$indice : $valor";
    }
?>
</body>
</html>
```

Veamos un simple ejemplo con un campo texto y como se recoge este en otra página para su posterior proceso.

```
<<formulario.html>>
```

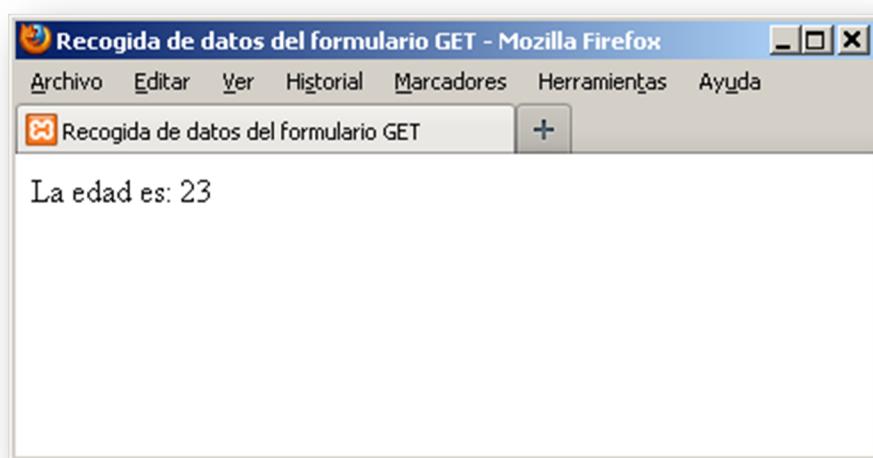
7 – Formularios.

```
<HTML>
<HEAD><TITLE>Formulario GET</TITLE></HEAD>
<BODY>
<FORM METHOD="GET" ACTION="destino.php">
    Edad: <INPUT TYPE="text" NAME="edad">
    <INPUT TYPE="submit" VALUE="aceptar">
</FORM>
</BODY>
</HTML>
```



<<destino.php>>

```
<HTML>
<HEAD><TITLE>Recogida de datos del formulario GET</TITLE></HEAD>
<BODY>
<?php
    $edad= $_GET['edad'];
    print("La edad es: $edad");
?>
</BODY>
</HTML>
```



7.2 – Utilizando POST.

La utilización del método GET es totalmente insegura, porque no hay forma de ocultar datos privados en la dirección Web, tales como la contraseña de entrada o el número de cuenta bancaria. El método POST arregla estos problemas ya que los parámetros y sus valores se pasan de forma oculta.

Las modificaciones a llevar a cabo en el ejemplo anterior son mínimas.

<<formulario.html>>

```
<HTML>
<HEAD><TITLE>Formulario POST</TITLE></HEAD>
<BODY>
<FORM METHOD="POST" ACTION="destino.php">
    Edad: <INPUT TYPE="text" NAME="edad">
    <INPUT TYPE="submit" VALUE="aceptar">
</FORM>
</BODY>
</HTML>
```

<<destino.php>>

```
<HTML>
<HEAD><TITLE>Recogida de datos del formulario POST</TITLE></HEAD>
<BODY>
<?php
    $edad= $_POST['edad'];
    print("La edad es: $edad");
?>
</BODY>
</HTML>
```

7.3 – Accediendo a los diferentes tipos de elementos.

Hemos visto que la manera de acceder por GET y por POST es muy similar. Los ejemplos anteriores se han centrado en elementos de formulario simples que únicamente almacenan un valor. En HTML existen diferentes tipos de componentes de formulario monovalor o multivalor.

Un único valor	Multivalor
<ul style="list-style-type: none"> - Text - Textarea - Button - Select (simple) - Hidden - Password - Submit 	<ul style="list-style-type: none"> - Radio - Checkbox - File - Select (múltiple)

El proceso para leer los campos multivalor es diferente. Veamos un ejemplo:

```
<INPUT TYPE="checkbox" NAME="extras[]" VALUE="garaje" CHECKED>Garaje  
<INPUT TYPE="checkbox" NAME="extras[]" VALUE="piscina">Piscina  
<INPUT TYPE="checkbox" NAME="extras[]" VALUE="jardin">Jardín
```

Primero recogemos el elemento general y posteriormente con un foreach recorremos todos los valores seleccionados.

```
<?PHP  
$extras = $_REQUEST['extras'];  
foreach($extras as $extra) print("$extra<BR>\n");  
?>
```

Otro ejemplo utilizando un elemento select múltiple:

```
Idiomas:  
<SELECT MULTIPLE SIZE="3" NAME="idiomas">  
  <OPTION VALUE="ingles" SELECTED>Inglés  
  <OPTION VALUE="frances">Francés  
  <OPTION VALUE="aleman">Alemán  
  <OPTION VALUE="holandes">Holandés  
</SELECT>
```

```
<?PHP  
$idiomas= $_REQUEST['idiomas'];  
foreach($idiomas as $idioma) print("$idioma<BR>\n");  
?>
```

7.4 – Enviando ficheros a través de formulario.

Para subir un fichero al servidor se utiliza el elemento de entrada FILE. Hay que tener en cuenta una serie de consideraciones importantes:

- El elemento FORM debe tener el atributo ENCTYPE="multipart/form-data"
- El fichero tiene un límite en cuanto a su tamaño. Este límite se fija de dos formas diferentes:
 - En el fichero de configuración php.ini
 - En el propio formulario.

<<php.ini>>

```
.....  
; File Uploads ;  
.....  
; Whether to allow HTTP file uploads.  
file_uploads= On  
; Temporary directory for HTTP uploaded files (will use  
; system default if not specified).  
;upload_tmp_dir=  
; Maximum allowed size for uploaded files.  
upload_max_filesize= 2M
```

<<formulario>>

```
<INPUT TYPE="HIDDEN" NAME="MAX_FILE_SIZE" VALUE='102400'>
<INPUT TYPE="FILE" NAME="fichero">
```

–Debe darse al fichero un nombre que evite coincidencias con ficheros ya subidos. Por ello, y como norma general, debe descartarse el nombre original del fichero y crear uno nuevo que sea único

–El fichero subido se almacena en un directorio temporal y hemos de moverlo al directorio de destino usando la función move_upload_file()

Procedimiento:

si se ha subido correctamente el fichero:
 Asignar un nombre al fichero
 Mover el fichero a su ubicación definitiva
si no:
 Mostrar un mensaje de error
fsi:

<<fichero HTML>>

```
<INPUT TYPE="HIDDEN" NAME="MAX_FILE_SIZE" VALUE="102400">
<INPUT TYPE="FILE" SIZE="44" NAME="imagen">
```

La variable \$_FILES contiene toda la información del fichero subido:

- \$_FILES['imagen']['name'] •Nombre original del fichero en la máquina cliente
- \$_FILES['imagen']['type'] •Tipo mime del fichero. Por ejemplo, "image/gif"
- \$_FILES['imagen']['size'] •Tamaño en bytes del fichero subido
- \$_FILES['imagen']['tmp_name'] •Nombre del fichero temporal en el que se almacena el fichero subido en el servidor
- \$_FILES['imagen']['error'] •Código de error asociado al fichero subido.

<<fichero PHP>>

```
if (is_uploaded_file($_FILES['imagen']['tmp_name'])){
    $nombreDirectorio= "img/";
    $nombreFichero= $_FILES['imagen']['name'];
    $nombreCompleto= $nombreDirectorio. $nombreFichero;
    if(is_file($nombreCompleto)){
        $idUnico= time();
        $nombreFichero= $idUnico. "-" . $nombreFichero;
    }
    move_uploaded_file($_FILES['imagen']['tmp_name'],$nombreDirectorio. $nombreFichero);
} else
    print("No se ha podido subir el fichero\n");
```

7.4 – Variables super globales.

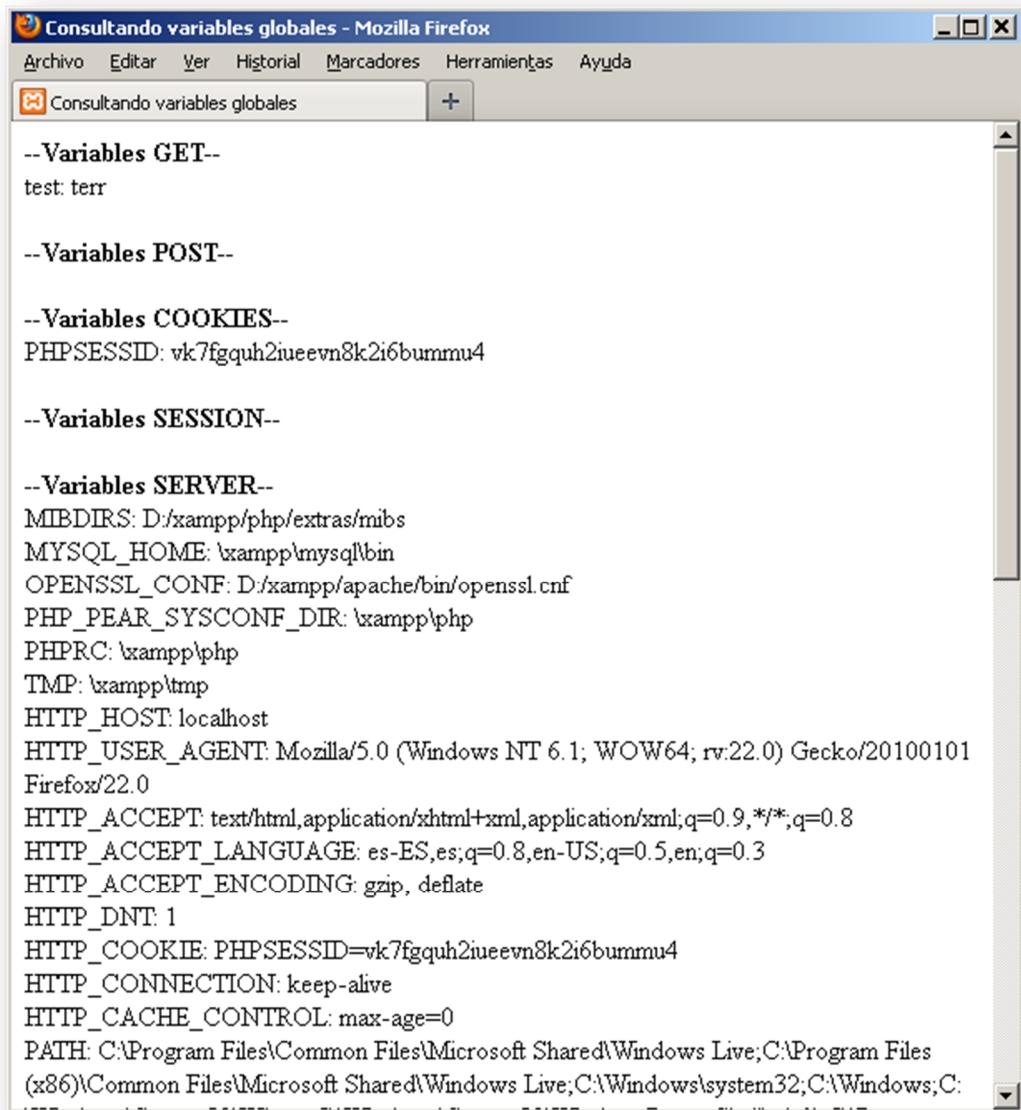
Desde PHP 4 se vienen utilizando los arrays súper-globales para almacenar los valores que pasan de unas páginas a otras, se almacenan en el ordenador del usuario o en el servidor. En PHP 5 el uso de estas variables se hace obligatorio para recuperar todos los valores. Las variables a utilizar son:

- `$_GET`: Almacena las variables que se pasan desde un formulario mediante el método GET.
- `$_POST`: Almacena las variables pasadas por POST.
- `$_COOKIE`: Guarda los valores que están almacenados en cookies.
- `$_SESSION`: Guarda las variables que se pasan entre sesiones.
- `$_SERVER`: Contiene numerosos valores relativos al servidor.
- `$_FILES`: Los archivos que envíemos a través de un formulario serán recogidos en este array.

Para ver y tratar la información que tienen estas variables globales podemos utilizar un bucle foreach para recorrer los elementos que contienen. La función siguiente muestra todos los valores de los arrays súper-globales:

```
<?php
    session_start(); // Conviene empezar la session en caso contrario fallará.

function depuracion() {
    echo "<b>--Variables GET--</b><br>";
    foreach ($_GET as $indice => $valor) {
        echo "$indice: $valor<br>";
    }
    echo "<br><b>--Variables POST--</b><br>";
    foreach ($_POST as $indice => $valor) {
        echo "$indice: $valor<br>";
    }
    echo "<br><b>--Variables COOKIES--</b><br>";
    foreach ($_COOKIE as $indice => $valor) {
        echo "$indice: $valor<br>";
    }
    echo "<br><b>--Variables SESSION--</b><br>";
    foreach ($_SESSION as $indice => $valor) {
        echo "$indice: $valor<br>";
    }
    echo "<br><b>--Variables SERVER--</b><br>";
    foreach ($_SERVER as $indice => $valor) {
        echo "$indice: $valor<br>";
    }
}
depuracion();
?>
```



A screenshot of a Mozilla Firefox browser window titled "Consultando variables globales - Mozilla Firefox". The window shows the output of a PHP script. The output is a list of global variables categorized by type:

- Variables GET--
test: terr
- Variables POST--
- Variables COOKIES--
PHPSESSID: vk7fgquh2iueevn8k2i6bummu4
- Variables SESSION--
- Variables SERVER--
MIBDIRS: D:/xampp/php/extras/mibs
MYSQL_HOME: \xampp\mysql\bin
OPENSSL_CONF: D:/xampp/apache/bin/openssl.cnf
PHP_PEAR_SYSCONF_DIR: \xampp\php
PHPRC: \xampp\php
TMP: \xampp\tmp
HTTP_HOST: localhost
HTTP_USER_AGENT: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0
HTTP_ACCEPT: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
HTTP_ACCEPT_LANGUAGE: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
HTTP_ACCEPT_ENCODING: gzip, deflate
HTTP_DNT: 1
HTTP_COOKIE: PHPSESSID=vk7fgquh2iueevn8k2i6bummu4
HTTP_CONNECTION: keep-alive
HTTP_CACHE_CONTROL: max-age=0
PATH: C:\Program Files\Common Files\Microsoft Shared\Windows Live;C:\Program Files (x86)\Common Files\Microsoft Shared\Windows Live;C:\Windows\system32;C:\Windows;C:\

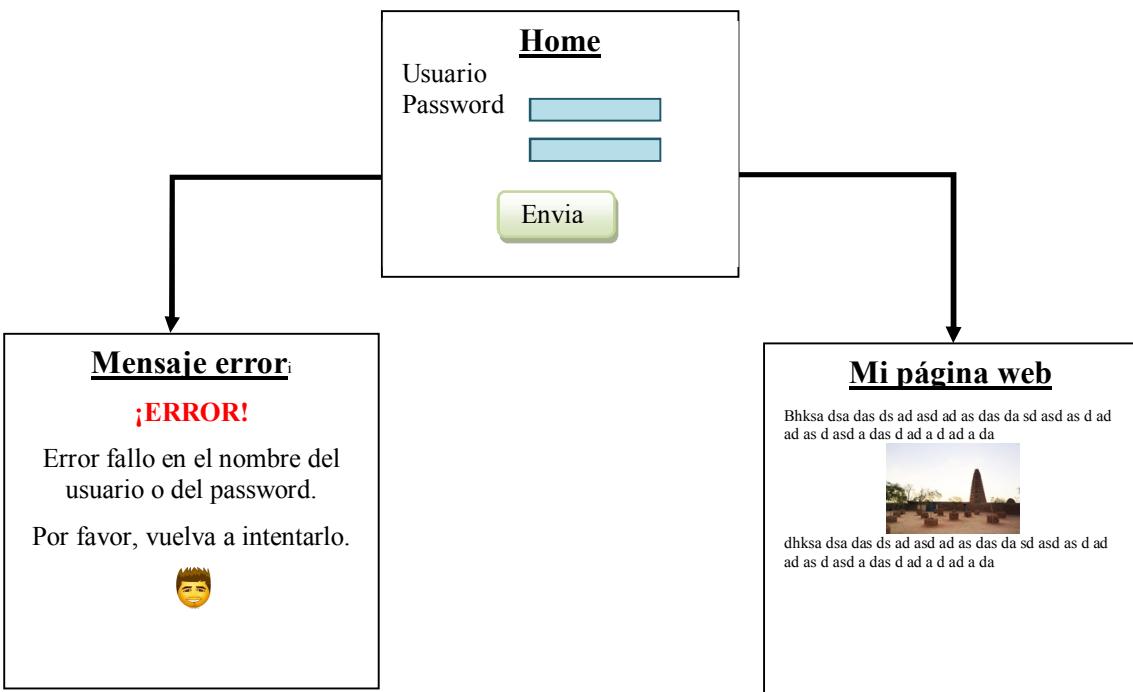
Actividades.

UD7 – ACTIVIDAD 1: Identificando usuarios.

TIPO	Desarrollo
OBJETIVOS	Practicar el uso de formularios.
RECURSOS	Editor de texto y navegador web.

ENUNCIADO DE LA ACTIVIDAD

Crea una estructura de páginas como la que se muestra abajo. La primera página llamada *home.php* contendrá un formulario con los campos Usuario y Password. La información se enviará a otra página de proceso llamada *proceso.php*. En el caso que los datos coincidan con los establecidos por el usuario, se permitirá el acceso a la página *mipagina.php* en caso contrario se mostrará un error mediante la página *error.php* y se le permitirá al usuario que pueda volver a intentarlo.



COMENTARIOS

Se pueden utilizar para el usuario y el password la encriptación de cadenas vista anteriormente. La redirección en php se puede realizar mediante la función header("Location: nombre de página").

UD7 – ACTIVIDAD 2: Enviando un currículum.

TIPO	Desarrollo
OBJETIVOS	Practicar el uso de formularios y la recogida de datos.
RECURSOS	Editor de texto y navegador web.

ENUNCIADO DE LA ACTIVIDAD

Crea una página con el siguiente formulario y otra llamada recogerdatos.php para mostrar la información enviada por el formulario.

The screenshot shows a Mozilla Firefox window with the title "7.1 - Mozilla Firefox". The main content is a form titled "Rellena tu CV". The form contains the following fields:
 - Nombre: A text input field.
 - Apellidos: A text input field.
 - Contraseña: A text input field.
 - DNI: A text input field.
 - Sexo: Radio buttons for "Hombre" (selected) and "Mujer".
 - Incluir mi foto: A checkbox. Next to it is a "Examinar..." button and a message stating "No se ha seleccionado ningún archivo.".
 - Suscribirme al boletín de novedades: A checkbox.
 - Buttons at the bottom: "Guardar cambios" and "Borrar los datos introducidos".

COMENTARIOS

Envia los datos siempre por POST para que el proceso sea más seguro.

Recuerda que los datos del fichero los debes procesar con `$_FILES["nombre del campo file"]['propiedad']`, donde propiedad puede ser alguna de las siguientes:

- `$_FILES['imagen']['name']` •Nombre original del fichero en la máquina cliente
- `$_FILES['imagen']['type']` •Tipo mime del fichero. Por ejemplo, "image/gif"
- `$_FILES['imagen']['size']` •Tamaño en bytes del fichero subido
- `$_FILES['imagen']['tmp_name']` •Nombre del fichero temporal en el que se almacena el fichero subido en el servidor
- `$_FILES['imagen']['error']` •Código de error asociado al fichero subido:

UD7 – ACTIVIDAD 3: Información sobre producto.

TIPO	Desarrollo
OBJETIVOS	Practicar el uso de formularios y la recogida de datos.
RECURSOS	Editor de texto y navegador web.

ENUNCIADO DE LA ACTIVIDAD

Crea una página con el siguiente formulario y otra llamada recogidados.php para mostrar la información enviada por el formulario.

The screenshot shows a Mozilla Firefox window with the title "7.3 - Mozilla Firefox". The menu bar includes Archivo, Editar, Ver, Historial, Marcadores, Herramientas, and Ayuda. A tab labeled "7.3" is open. The main content area displays a form titled "Información sobre el producto".

Datos básicos

Nombre:

Descripción:

Foto: No se ha seleccionado ningún archivo.

Añadir contador de visitas

Datos económicos

Precio: € Impuestos:

Promoción:

Ninguno
 Transporte gratuito
 Descuento 5%

COMENTARIOS

Recuerda que los campos multivalores como el checkbox, radiobutton o select multiple deben tener el mismo nombre pero diferentes valores.

8 – Cookies.

Una cookie es una pequeña parte de información que queda almacenada en el ordenador local de los usuarios de una página Web. Debe contener siempre un nombre y un valor. Es muy usual utilizar cookies para guardar preferencias de cada usuario en el uso de una Web (idioma, colores y formas, últimas consultas, etc.), pero hay que tener cuidado con el número de variables que se pueden almacenar, pues normalmente hay un máximo de 20 por dominio, según el navegador.

Las cookies tienen la siguiente forma: **xxx@nombre[z].txt** donde xxx suele ser el nombre que figura en el registro de Windows como nombre del equipo (el que se pone al instalar Windows); nombre suele ser el nombre del directorio de servidor desde el que se envió la cookie y el número z suele ser el ordinal del números de accesos a la página que envía la cookie.

8.1 – Creando cookies.

Para crear una cookie PHP pone a nuestra disposición la función setcookie cuya sintaxis es la siguiente:

```
bool setcookie( string nombre [, string valor [, int expirer [, string ruta [, string dominio [, bool segura]]]]])
```

Como podemos ver se trata de una función que devuelve un valor booleano. Si existe salida antes de llamar esta función, setcookie() fallará y devolverá FALSE. Si setcookie() se ejecuta con éxito, devolverá TRUE. Esto no indica si el usuario aceptó la cookie. Todos los argumentos exceptuando el argumento name son opcionales.

- **name** El nombre de la cookie.
- **value** El valor de la cookie. Este valor se guarda en el computador del cliente; no almacene información sensible. Asumiendo que el name es 'cookiename', este valor se obtiene con \$_COOKIE['cookiename'].
- **expire** El tiempo en el que expira la cookie, está en número de segundos a partir de la presente época. En otras palabras, probablemente utilizará la función time() más el número de segundos que quiere que dure la cookie. También podría utilizar la función mktime(). time() + 60 * 60 * 24 * 30 configurará la cookie para expirar en 30 días. Si se pone 0, o se omite, la cookie expirará al final de la sesión (al cerrarse el navegador).
- **path** La ruta dentro del servidor en la que la cookie estará disponible. Si se utiliza '/', la cookie estará disponible en la totalidad del domain. Si se configura como '/foo/', la cookie sólo estará disponible dentro del directorio /foo/ y todos sus subdirectorios en el domain, tales como /foo/bar/. El valor por defecto es el directorio actual en donde se está configurando la cookie.
- **domain** El dominio para el cual la cookie está disponible. Establecer el dominio a 'www.example.com' hará que la cookie esté disponible en el subdominio www y subdominios superiores. Las cookies disponibles en un dominio inferior, como 'example.com', estarán disponibles en dominios superiores, como 'www.example.com'.
- **secure** Indica que la cookie sólo debiera transmitirse por una conexión segura HTTPS desde el cliente. Cuando se configura como TRUE, la cookie sólo se creará si es que existe una conexión segura. Del lado del servidor, depende del

programador el enviar este tipo de cookies solamente a través de conexiones seguras (por ejemplo, con `$_SERVER["HTTPS"]`).

- **httponly** Cuando es TRUE la cookie será accesible sólo a través del protocolo HTTP. Esto significa que la cookie no será accesible por lenguajes de scripting, como JavaScript.

Las cookies deben ser enviadas antes de cualquier salida (ésta es una restricción de protocolo). Esto requiere que coloque las llamadas a esta función antes de cualquier salida, incluyendo las etiquetas `<html>` y `<head>` así como cualquier espacio en blanco.

El siguiente código muestra tres formas de utilizar `setcookie`:

```
<?php
$value = 'cualquier cosa';

setcookie("TestCookie", $value);
setcookie("TestCookie", $value, time() + 3600); /* expira en una hora */
setcookie("TestCookie", $value, time() + 3600, "/~rasmus/", "example.com", 1);

?>
```

También puede crear arrays de cookies utilizando la notación de arrays en el nombre de la cookie. El efecto de ésto es de crear tantas cookies como elementos hay en el array, pero al recibir el script la cookie, todos los valores son colocados en un array con el nombre de la cookie:

```
<?php
// crear las cookies
setcookie("cookie[tres]", "cookiетres");
setcookie("cookie[dos]", "cookiедос");
setcookie("cookie[uno]", "cookieuno");

// imprimirlas luego que la página es recargada
if (isset($_COOKIE['cookie'])) {
    foreach ($_COOKIE['cookie'] as $name => $value) {
        $name = htmlspecialchars($name);
        $value = htmlspecialchars($value);
        echo "$name : $value <br />\n";
    }
}
?>
```

El resultado del ejemplo sería:

tres : cookietres
dos : cookiedos
uno : cookieuno.

8.2 – Accediendo a las cookies.

Al invocar la variable mediante la que fue escrita la cookie –desde cualquier página que esté alojada en el *subdirectorio del servidor* desde el que fué creada– tomará –de forma automática– el valor contenido en la cookie.

Esto sólo sería posible en el caso de que la cookie hubiera sido creada con anterioridad y que no hubiera expirado.

Tal como comentábamos al estudiar los formularios, las cookies pueden leerse directamente (imprimir la variable PHP con nombre idéntico a la cookie) sólo en el caso de que contemos con la opción **register_globals=on**.

Si no fuera así (de forma similar al caso de los formularios) para poder leer su contenido tendríamos que recurrir al uso de la variable superglobal **\$_COOKIE**.

Puedes comprobar que en los ejemplos se visualizan los contenidos de las cookies usando la instrucción **echo** (o **print**) y utilizando el array superglobal **\$_COOKIE** para garantizar el funcionamiento con cualquier configuración de **register globals**.

```
<HTML>
<BODY>
    Usamos la cookie:<BR>
<?PHP
    echo($_COOKIE["value"]);
?>
</BODY>
</HTML>
```

Podemos acceder a todas las variables almacenadas en las cookies mediante el siguiente código haciendo uso de un **foreach** para recorrer la supervariable **\$_COOKIE**.

```
<HTML>
<BODY>
<?php
    echo "Mostrar las COOKIES<br>" ;
    foreach ($_COOKIE as $indice => $valor) {
        echo "$indice: $valor<br>" ;
    }
?>
</BODY>
</HTML>
```

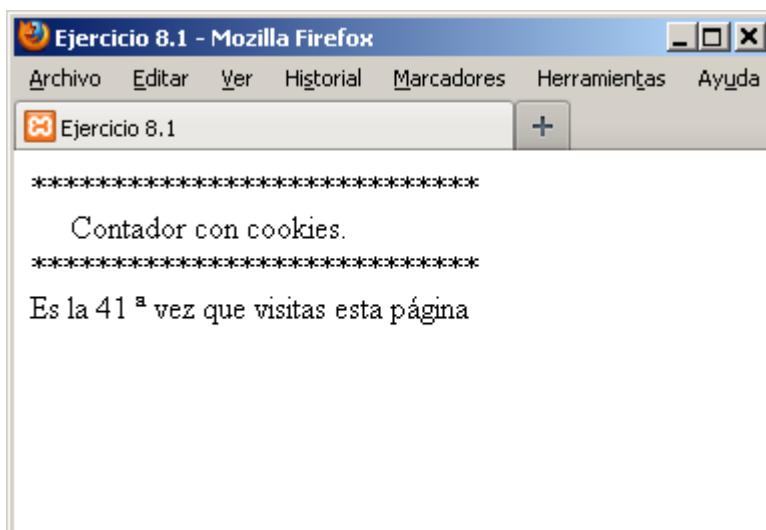
Actividades.

UD8 – ACTIVIDAD 1: Contador de visitas.

TIPO	Desarrollo
OBJETIVOS	Practicar el uso de la etiqueta cookies para almacenar información.
RECURSOS	Editor de texto y navegador web.

ENUNCIADO DE LA ACTIVIDAD

Crea un script que mediante cookies nos permita tener controlado el número de visitas de un usuario en nuestra página web.

**COMENTARIOS**

Recuerda que debes utilizar la supervariable \$_COOKIE para acceder a la cookie que has registrado.

9 – Sesiones.

A veces es necesario mantener el estado de una conexión entre distintas páginas o entre distintas visitas a un mismo sitio. Casos que se pueden citar como ejemplo serían: aplicaciones personalizadas, carrito de la compra, control de acceso

HTTP es un **protocolo sin estado**: cada conexión entre el cliente y el servidor es independiente de las demás y para mantener el estado entre diferentes conexiones hay que establecer lo que se conoce como una **sesión**. Las sesiones permiten disponer de unas variables con valores persistentes durante toda la conexión del usuario. Estas variables pueden almacenarse en el cliente mediante *cookies* o en el servidor. PHP dispone de una biblioteca de funciones para la gestión de sesiones

9.1 – Funciones para el manejo de sesiones.

Las funciones que nos permiten manejar sesiones son las siguientes:

- `session_cache_expire` — Devuelve la caducidad de la caché actual
- `session_cache_limiter` — Obtener y/o establecer el limitador de caché actual
- `session_commit` — Alias de `session_write_close`
- `session_decode` — Decodifica la información de sesión desde una cadena de sesión codificada
- **`session_destroy`** — Destruye toda la información registrada de una sesión
- `session_encode` — Codifica los datos de la sesión actual como un string codificado de sesión
- `session_get_cookie_params` — Obtener los parámetros de la cookie de sesión
- `session_id` — Obtener y/o establecer el id de sesión actual
- `session_name` — Obtener y/o establecer el nombre de la sesión actual
- `session_regenerate_id` — Actualiza el id de sesión actual con uno generado más reciente
- `session_register_shutdown` — Función de cierre de sesiones
- `session_save_path` — Obtener y/o establecer la ruta de almacenamiento de la sesión actual
- `session_set_cookie_params` — Establecer los parámetros de la cookie de sesión
- `session_set_save_handler` — Establece funciones de almacenamiento de sesiones a nivel de usuario
- **`session_start`** — Iniciar una nueva sesión o reanudar la existente. Si la sesión ya está iniciada, carga todas las variables de sesión. Este código debe ser puesto antes de cualquier código HTML.
- `session_status` — Devuelve el estado de la sesión actual
- `session_unset` — Libera todas las variables de sesión
- `session_write_close` — Escribir información de sesión y finalizar la sesión

9.2 – Trabajando con sesiones.

PHP ha hecho un esfuerzo para simplificar el trabajo con sesiones y ha puesto a nuestra disposición una supervariable llamada `$_SESSION`. Con las operaciones básicas de asignación de valor o consulta podemos establecer y leer nuestras propias variables de sesión. Veamos un ejemplo: Creamos dos páginas web, la primera `pagina1.php` establece unos valores en la sesión y la segunda `pagina2.php` los consulta.

<<`pagina1.php`>>

```
<?php
    session_start();

    echo 'Bienvenido a la página #1';

    $_SESSION['color'] = 'verde';
    $_SESSION['animal'] = 'gato';
    $_SESSION['time'] = time();

    // Trabajar si la sesión fue aceptada
    echo '<br /><a href="pagina2.php">página 2</a>';

?>
```

<<`pagina2.php`>>

```
<?php
    session_start();

    echo 'Bienvenido a la página #2<br />';

    echo $_SESSION['color']; // verde
    echo $_SESSION['animal']; // gato
    echo date('Y m d H:i:s', $_SESSION['time']);

?>
```

Podemos acceder a todas las variables almacenadas en la sesión mediante el siguiente código haciendo uso de un `foreach` para recorrer la supervariable `$_SESSION`.

```
<?php
    session_start () ;
?>
<HTML>
<BODY>
<?php
    echo "Mostrar las variables de sesión<b>"; 
    foreach ($_SESSION as $indice => $valor) {
        echo "<br>$indice: $valor";
    }
?>
</BODY>
</HTML>
```

Actividades.

UD9 – ACTIVIDAD 1: De formulario a sesión.

TIPO	Desarrollo
OBJETIVOS	Practicar el manejo de sesiones con PHP.
RECURSOS	Editor de texto y navegador web.

ENUNCIADO DE LA ACTIVIDAD

Almacena en la sesión los datos introducidos en el formulario del ejercicio 7.2.

The screenshot shows a Mozilla Firefox window with the title "7.1 - Mozilla Firefox". The menu bar includes Archivo, Editar, Ver, Historial, Marcadores, Herramientas, and Ayuda. A tab labeled "7.1" is open. The main content area displays a form titled "Rellena tu CV". The form contains the following fields:
 - Nombre: An input field.
 - Apellidos: An input field.
 - Contraseña: An input field.
 - DNI: An input field.
 - Sexo: Radio buttons for "Hombre" (selected) and "Mujer".
 - Incluir mi foto: A checkbox followed by a "Examinar..." button and a message stating "No se ha seleccionado ningún archivo".
 - Suscribirme al boletín de novedades: A checkbox.
 - Buttons at the bottom: "Guardar cambios" and "Borrar los datos introducidos".

COMENTARIOS

Utiliza `$_SESSION` tanto para almacenar como para leer los datos en la sesión

10– Almacenamiento de datos. Ficheros

Previo a la proliferación de los gestores de bases de datos se hiciera efectiva, los lenguajes de programación utilizaban el acceso a ficheros para almacenar sus datos.

Actualmente, la información puede almacenarse en ficheros o en bases de datos. Los ficheros pueden almacenar textos que pueden ser leídos fácilmente por nosotros y por un ordenador. Además, existe una serie de utilidades que permiten al sistema operativo (Windows, Unix o Mac OSX) interactuar con ellos, hacer búsquedas, etcétera.

PHP dispone de funciones mediante las cuales se pueden crear, modificar, borrar y leer ficheros de cualquier tipo así como extraer información sobre ellos y sus contenidos.

10.1 – Apertura y cierre de ficheros.

La función **fopen()** asigna a una variable un puntero (un descriptor) al fichero que quiera abrir. La variable puede utilizarse después para hacer cualquier tipo de operación. Si el fichero que intenta abrir no existe o no puede utilizarse en ese momento, fopen() devolverá un valor false.

```
$f1=fopen(fichero,modo);
```

\$f1 es una variable que recoge el identificador del recurso, un valor importante (será utilizado para referirnos a este fichero en instrucciones posteriores), fichero es el nombre (con extensión) del fichero a abrir o crear y deberá escribirse entre comillas, y modo, que es una cadena que debemos poner entre comillas, el indicador del modo de apertura elegido.

Si el fichero que pretendemos abrir está en un directorio distinto al del script, debe incluirse el path completo delante del nombre del fichero y la cadena resultante debe ir entre comillas.

Valores del parámetro modo de la función fopen	
Valor	Funcionalidad
r	Abre el fichero en modo lectura y coloca el puntero al comienzo del fichero
r+	Abre el fichero en modo lectura y escritura y coloca el puntero al comienzo del fichero
w	Abre el fichero en modo escritura y coloca el puntero al comienzo del fichero, reduce su tamaño a cero y si el fichero no existe intenta crearlo
w+	Abre el fichero en modo lectura y escritura y coloca el puntero al comienzo del fichero, reduce su tamaño a cero y si el fichero no existe intenta crearlo
a	Abre el fichero en modo escritura y coloca el puntero al final del fichero y si no existe intenta crearlo
a+	Abre el fichero en modo lectura y escritura y coloca el puntero al final del fichero y si no existe intenta crearlo

Una vez finalizado el uso de un fichero es necesario cerrarlo. Para ello PHP dispone de la siguiente instrucción **fclose()**:

```
fclose($f1);
```

Esta función - que devuelve un valor booleano- permite cerrar el fichero especificado en \$f1 que, como recordarás, es el valor del identificador de recurso que le fue asignado automáticamente por PHP en el momento de la apertura.

10.2 – Punteros internos.

La función **fopen()** asigna a una variable un puntero (un descriptor) al fichero que quiera abrir. La variable puede utilizarse después para hacer cualquier tipo de operación. Si el fichero que intenta abrir no existe o no puede utilizarse en ese momento, fopen() devolverá un valor false.

PHP dispone de funciones para situar sus *punteros internos* y también para determinar la posición a la que apuntan en un momento determinado. Se trata de las siguientes:

feof(\$f1) Es un operador booleano que devuelve CIERTO (1) si el puntero señala el final del fichero y FALSO si no lo hace.

rewind(\$f1) Coloca el *puntero interno* al comienzo del fichero indicado por *el identificador del recurso* \$f1.

fseek(\$f1, posición) Sitúa el apuntador del fichero señalado por *el identificador del recurso* \$f1 en la posición (expresada en bytes) señalada por posición.

ftell(\$f1) Devuelve (expresada en bytes) la posición actual del puntero interno del fichero.

Antes de utilizar funciones es necesario que el fichero que señala el *identificador de recursos* haya sido **abierto**.

10.3 – Lectura de ficheros.

Para realizar la lectura de ficheros utilizamos la función **fread()** que tiene como parámetro un descriptor de fichero, devuelto por la función fopen() y el tamaño del bloque de datos que queremos leer.

```
$variable = fread($descriptor, tamaño);
```

```
<?php  
// poner el contenido de un fichero en una cadena  
$nombre_fichero = "fichero.txt";  
$gestor = fopen($nombre_fichero, "r");
```

```

$contenido = fread($gestor, filesize($nombre_fichero));
fclose($gestor);           //Cerramos el fichero
?>

```

Normalmente no conocerá el tamaño del fichero y, por lo tanto, se debe utilizar alguna técnica para leerlo correctamente. Lo más sencillo es apoyar se en la función **filesize()** que devuelve el tamaño en bytes del fichero que pase como argumento. Así, la lectura anterior quedaría:

```
$variable = fread($descriptor, filesize("noticias.txt"));
```

La función anterior devuelve un bloque completo de caracteres leídos desde un fichero y puede hacerse poco manejable cuando intente buscar algún dato específico. PHP posee algunas funciones para leer línea a línea desde un fichero, fread() tiene como parámetros el descriptor del fichero y un bloque máximo de caracteres a leer. Veamos un ejemplo:

```

<?php
$descriptor = fopen ("prueba.txt","a +");
$linea_numero = 1;
while (!feof($descriptor)) {
    $linea = fgets ($descriptor,4096) ;
    echo "Línea número: $numero_linea es: $linea";
    $linea_numero++;
}
fclose($descriptor);
?>

```

El bucle actúa hasta que la función feof() encuentra el final del fichero apuntado por el descriptor. Mientras que no sea el fin del fichero, la función fgets() irá leyendo línea a línea todo el contenido e imprimiéndolo en pantalla. La forma ideal de trabajar con ficheros es ir leyendo poco a poco su contenido, para poder buscar datos que nos interesen en cada línea o conjunto de caracteres.

10.5 – Escritura de ficheros.

Para realizar la lectura de ficheros utilizamos la función **fwrite()** que tiene como parámetro un descriptor de fichero, devuelto por la función fopen(), una cadena con la información a almacenar y la longitud a escribir.

```
int fwrite (descriptor $handle , cadena $string [, int $longitud ] );
```

```

<?php
$nombre_archivo = 'prueba.txt';
$contenido = "Añade esto al archivo\n";

// Primero vamos a asegurarnos de que el archivo existe y es escribible.
if (is_writable($nombre_archivo)) {

    // En nuestro ejemplo estamos abriendo $nombre_archivo en modo de adición.
}

```

```

// El puntero al archivo está al final del archivo
// donde irá $contenido cuando usemos fwrite() sobre él.
if (!$gestor = fopen($nombre_archivo, 'a')) {
    echo "No se puede abrir el archivo ($nombre_archivo)";
    exit;
}

// Escribir $contenido a nuestro archivo abierto.
if (fwrite($gestor, $contenido) === FALSE) {
    echo "No se puede escribir en el archivo ($nombre_archivo)";
    exit;
}

echo "Éxito, se escribió ($contenido) en el archivo ($nombre_archivo)";

fclose($gestor);

} else {
    echo "El archivo $nombre_archivo no es escribible";
}
?>
```

10.5 – Otras acciones con ficheros.

Borrado de ficheros.

Para borrar ficheros se utiliza la siguiente instrucción: **unlink(fichero)** fichero ha de ser una cadena que contenga el nombre y la extensión del fichero y, en su caso, también el path.

```
<?php
$fh = fopen('test.html', 'a');
fwrite($fh, '<h1>¡Hola mundo!</h1>');
fclose($fh);
unlink('test.html');           //Borramos el fichero
?>
```

Duplicado de ficheros.

La función: **copy(fich1, fich2)** Copia el fichero fich1 (debe indicarse nombre y extensión) en otro fichero cuyo nombre y extensión se establecen en la cadena fich2. Esta función devuelve un valor booleano indicando si la copia se ha realizado con éxito TRUE (1) o FALSE (nul) si por alguna razón no ha podido realizarse la copia.

```
<?php
$archivo = 'ejemplo.txt';
$nuevo_archivo = 'ejemplo.txt.bak';

if (!copy($archivo, $nuevo_archivo)) {
    echo "Error al copiar $archivo...\n";
}
?>
```

Renombrar ficheros.

La función: **rename(fich1, fich2)** cambia el nombre del fichero fich1(hay que poner nombre y extensión) por el indicado en la cadena fich2. También devuelve TRUE o FALSE. Si tratamos de cambiar el nombre a un fichero inexistente nos dará error.

```
<?php
    rename("archivo_tmp.txt", "mi_archivo.txt");
?>
```

10.5 – Otras acciones con ficheros.

PHP dispone de funciones que nos facilitan información sobre ficheros. Algunas de ellas son las siguientes:

file_exists(fichero) Esta función devuelve TRUE si el fichero existe, en caso contrario devuelve FALSE.

filesize(fichero) Devuelve el *tamaño* del fichero expresándolo en *bytes*. En caso de que el fichero no existiera nos dará un error.

filetype(fichero) Devuelve una *cadena* en la que se indica el *tipo del fichero*. En caso de que el fichero no existiera nos dará un error.

filemtime(fichero) Devuelve –en *tiempo Unix*– la *fecha de la última modificación* del fichero.

stat(fichero) Devuelve un *array* que contiene información sobre el fichero. En la siguiente tabla se puede ver los contenidos asociados a cada uno de sus índices. Recogeremos en un array, que llamaremos \$d, el resultado de la función stat aplicada sobre el fichero domingo.txt. Para ello vamos a utilizar la siguiente sintaxis:

\$d=stat("domingo.txt")

El contenido y significado de los valores asociados a los índices de array **\$d** son los que tenemos en la tabla de la derecha.

Indice	Significado	Sintaxis	Resultado
0	Dispositivo	<? echo \$d[0] ?>	3
1	I node	<? echo \$d[1] ?>	0
2	Modo de protección de I node	<? echo \$d[2] ?>	33206
3	Número de enlaces	<? echo \$d[3] ?>	1
4	Id de usuario del propietario	<? echo \$d[4] ?>	0
5	Id de grupo del propietario	<? echo \$d[5] ?>	0
6	tipo de dispositivo si es un inode device	<? echo \$d[6] ?>	3
7	Tamaño en bytes	<? echo \$d[7] ?>	126
8	Fecha del último acceso	<? echo \$d[8] ?>	1372704784

9	Fecha de la última modificación	<? echo \$d[9] ?>	1243009480
10	Fecha del último cambio	<? echo \$d[10] ?>	1372704784
11	Tamaño del bloque para el sistema I/O	<? echo \$d[11] ?>	-1
12	Número de bloques ocupados	<? echo \$d[12] ?>	-1

10.5 – Gestión de directorios.

Para la gestión de directorios o carpetas PHP pone a nuestra disposición una serie de funciones que explicamos a continuación.

chdir — Cambia de directorio.

```
<?php
// directorio actual
echo getcwd() . "\n";
chdir('public_html');
// directorio actual
echo getcwd() . "\n";
?>
```

chroot — Cambia el directorio raíz.

closedir — Cierra un gestor de directorio.

```
<?php
$dir = "c:/prueba/";
// Abrir un directorio conocido, lee el directorio en una variable y lo cierra.
if (is_dir($dir)) {
    if ($dh = opendir($dir)) {
        $directory = readdir($dh);
        closedir($dh);
    }
}
?>
```

dir — Devuelve una instancia de la clase Directory.

getcwd — Obtiene el directorio actual en donde se esta trabajando.

opendir — Abre un gestor de directorio.

```
<?php
$dir = "c:/prueba/";

// Abre un directorio conocido, y procede a leer el contenido
if (is_dir($dir)) {
    if ($dh = opendir($dir)) {
        while (($file = readdir($dh)) !== false) {
            echo "nombre archivo: $file : tipo archivo: " . filetype($dir . $file) . "\n";
        }
        closedir($dh);
    }
}
?>
```

readdir — Lee una entrada desde un gestor de directorio.

rewinddir — Regresar el gestor de directorio.

scandir — Enumera los ficheros y directorios ubicados en la ruta especificada.

```
<?php
$dIRECTORIO = '/tmp';
$fICHEROS1 = scandir($dIRECTORIO);
$fICHEROS2 = scandir($dIRECTORIO, 1);
print_r($fICHEROS1);
print_r($fICHEROS2);
?>
```

El resultado del ejemplo sería algo similar a:

```
Array
(
    [0] => .
    [1] => ..
    [2] => bar.php
    [3] => foo.txt
    [4] => somedir
)
```

Actividades.

UD10 – ACTIVIDAD 1: Contador de visitas.

TIPO	Desarrollo
OBJETIVOS	Practicar la manipulación de ficheros.
RECURSOS	Editor de texto y navegador web.

ENUNCIADO DE LA ACTIVIDAD

Realizar un contador de visitas utilizando un fichero llamado contador.txt para tal efecto. Para empezar comprobamos si existe el fichero contador. Si existe leemos las visitas registradas e incrementamos su valor en una unidad y si no existe, registramos 1 como valor de número de visitas.


COMENTARIOS

Tener especial cuidado con los modos de apertura de fichero, diferenciar entre la lectura y escritura. No olvidar al final del script cerrar el fichero.

UD10 – ACTIVIDAD 2: Libro de visitas.

TIPO	Desarrollo
OBJETIVOS	Practicar el uso de ficheros para almacenar información.
RECURSOS	Editor de texto y navegador web.

ENUNCIADO DE LA ACTIVIDAD

Crea un libro de visitas basado en el modelo de abajo. En una misma página php, destina una parte para el formulario de datos y otra para el procesado de la información. Crea previamente un fichero llamado guestbook.txt.

Un libro de visitas muy sencillo - Mozilla Firefox

Archivo Editar Ver Historial Marcadores Herramientas Ayuda

Un libro de visitas muy sencillo

Libro de visitas

Tu nombre:

Tu e-mail:

publica:

Mostrar todos los comentarios

Juan Carlos Ruiz (jcruz@skype.com) escribió el 06-07-2013 a las 13:34:15;
Estoy aprendiendo mucho con tu página, gracias por los ejemplos que pones y el tiempo que le dedicas. Un saludo.

Maria Sánchez (msanchez@gmail.com) escribió el 06-07-2013 a las 13:33:06;
Sigue así, impresionándonos con tus conocimientos.

Marcos López (mlopez@yahoo.es) escribió el 06-07-2013 a las 13:32:22;
Tu página me parece muy buena, ánimo y continúa trabajando en ello.

COMENTARIOS

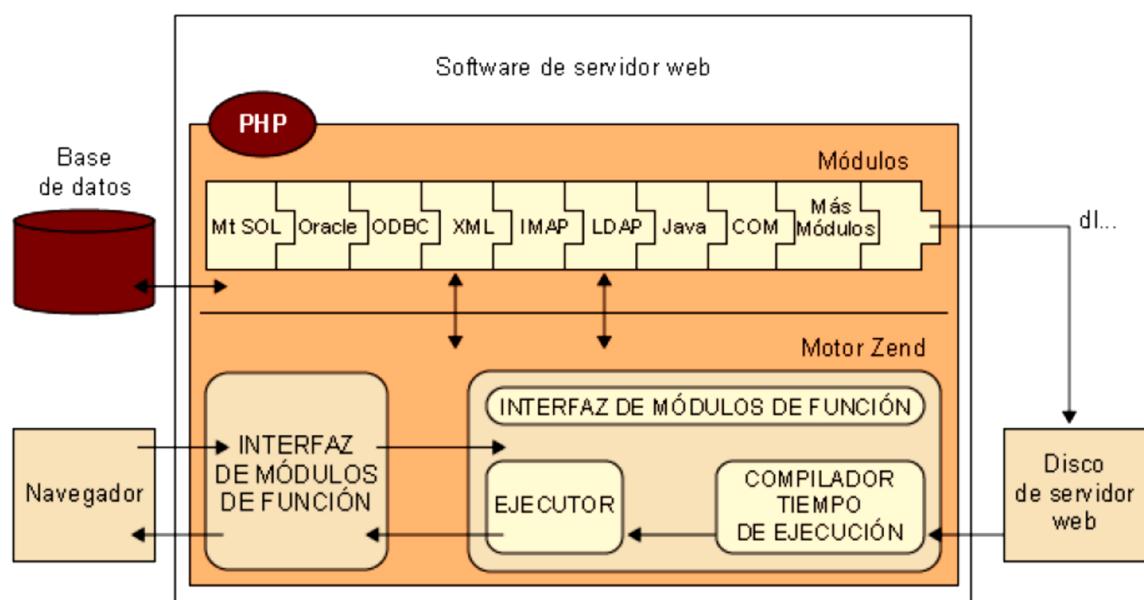
Puedes utilizar también PHP para la validación de datos como el email.

11 – Programación orientada a objetos.

11.1 – Introducción.

PHP comenzó siendo un simple lenguaje de scripts. A medida que las versiones avanzaban, se fueron incluyendo algunas características que permitían programar con orientación a objetos. Con la aparición de PHP 5, los desabolladores tenemos una verdadera plataforma de programación orientada a objetos, gracias a la potencia del nuevo engine Zend 2.0, que permite ejecutar más rápido y eficientemente este tipo de programas.

Estructura interna de PHP



El nombre Zend se refiere al motor del lenguaje, es decir, el núcleo de PHP. El término PHP se refiere al sistema completo tal y como aparece desde fuera. Zend ocupa la parte de **intérprete** (analiza el código de entrada de un *script*, lo traduce y lo ejecuta), y también un poco de la parte de **funcionalidad** (implementa la funcionalidad del sistema). PHP ocupa la parte de funcionalidad y la de **interfaz** (habla con el servidor web, etc.). Juntos forman el paquete completo PHP.

Zend forma realmente el núcleo del lenguaje, mientras que PHP contiene todos los módulos externos (los cuales se pueden cargar en tiempo de ejecución) e incorporados (los que se compilan directamente con PHP) que crean las posibilidades destacadas del lenguaje.

La programación orientada a objetos utiliza los elementos clase y objeto con punto de inicio. Actualmente, en las carreras técnicas, es posible encontrar enseñanzas sobre lenguajes orientados a objetos como Java o C++.

¿Porqué usar PHP OO?

En PHP, una función declarada en una página PHP podrá ser usada inmediatamente después a su declaración. Como primera aproximación, uno podría plantearse programar en cada fichero PHP todas las funciones necesarias que vayan a usarse en el fichero. En proyectos grandes, esto supone un alto coste de mantenimiento debido a la complejidad de seguimiento del código y a la duplicación del mismo código en distintas páginas. En caso de detectarse un error y el código que lo corrige, o en caso de querer aplicar una mejora en una función, deberán revisarse todos los scripts PHP para comprobar si contienen la función que hay que corregir.

Sin mucha dificultad puede observarse que la práctica de repetir códigos en scripts PHP queda lejos de ser una vía óptima para la programación en proyectos.

PHP permite definir clases, que poseen variables (también llamadas propiedades) y métodos (funciones). Estas clases nos permiten una serie de ventajas (optimización y reutilización de código, estructuración y modelización de problemas y soluciones, extensibilidad, etc.) sobre la programación procedural o de funciones que explicaremos a continuación.

Con la programación orientada a objetos, las clases quedan localizadas dentro de un único fichero PHP. Durante el proceso de programación, suelen darse errores dentro del comportamiento de las aplicaciones, por lo que es necesario efectuar un análisis para detectar la función que está incorrectamente implementada.

En la programación orientada a objetos, dada su organización, solamente será necesario corregir el error dentro de la clase correspondiente, y todos los objetos de la clase creados se beneficiarán de la corrección.

11.2 – Clases y objetos.

Una **clase** no es otra cosa que una especie de *plantilla* en la que se pueden definir una serie de **variables** –que pueden contener valores predefinidos– conocidas como **atributos** y un conjunto de **funciones**, llamadas **métodos**, que pueden ser *invocadas* desde cualquier parte del documento. Desde una clase se pueden construir varios objetos del mismo tipo.

La sintaxis es la siguiente:

```
class nombre {
    ...
    ... definición de atributos...
    ...
    ... constructores (opcional)...
    ...
    ... definición de métodos...
    ...
}
```

```
<?php
class SimpleClass
{
    // Declaración de la propiedad
    public $var = 'a default value';

    // Declaración del método
}
```

```

public function displayVar() {
    echo $this->var;
}
?>

```

Siempre que desde una función –contenida en una clase– se trate de invocar una variable definida en la misma clase ha de hacerse con la siguiente sintaxis:

\$this->variable

Prestemos mucha atención. El \$ va siempre delante de la palabra this y solo se escribe una vez y en esa posición.

El nombre de la variable (que va siempre después de -> no lleva pegado el \$. El operador **\$this** es una variable que contiene el objeto actual desde el que la invocamos. Para acceder dentro de un objeto a funciones o variables propias, debe anteponerse al nombre de éstas la expresión **\$this->**.

Crear y utilizar objetos.

Las **clases** son solo *plantillas* y sus **funciones** no se ejecutan hasta que se les **ordene**. Para poder utilizarlas primero debemos **crear un objeto** y luego **ejecutar** sobre ese **objeto** la función o funciones que deseemos.

Creación de objetos.

Para crear un **objeto** en el que se vaya a utilizar **una clase determinada** debemos usar la siguiente sintaxis:

\$nombre = new clase ();

donde **nombre** es una *palabra cualquiera* con la que identificar el **objeto** (como si se tratara de una variable) y **clase** es el **nombre** de una de las clases definidas. Fíjate que *detrás del nombre de la clase no hemos puesto los ()* que suelen utilizarse para invocar las **funciones**.

```

<?php
    $instance = new SimpleClass();
?>

```

Utilizando objetos.

Una vez definido un **objeto** ya se le pueden aplicar las **funciones** definidas en la **clase**. La sintaxis es la siguiente:

\$nombre->funcion();

Donde **\$nombre** es la *misma variable* utilizada a la hora de **crearel** objeto, el → es obligatorio, **funciones** el nombre de **una de las funciones** definidas en la **clase**invocada y donde los () sí **son obligatorios** y además -como ocurría en las demás funciones PHP- puede contener *valores, variables, etcétera* separadas por comas.

Reiterando llamadas a objetos.

Si hacemos varias *llamadas* a una función utilizando el *mismo objeto* los resultados se van *sobrescribiendo* sobre los de la llamada anterior.

Si queremos conservar *varios resultados* obtenidos de la aplicación de la *misma función* tenemos dos opciones: Crear varios objetos o Utilizar *arrays*.

Invocando varias veces el mismo objeto

En este ejemplo puedes observar cómo al *invocar* dos veces a **\$objeto** el valor que nos devuelve es el resultado de la **última llamada** y también como se pueden crear **dos objetos** distintos.

```
<?
class Multiplica{
    var $resultado;
    function opera($a,$b){
        $this->resultado=$a*$b;
    }
    function imprimelo(){
        echo $this->resultado,"<br>";
    }
}
$objeto= new Multiplica;
$objeto->opera (7,3);
$objeto->opera (11,4);
$objeto1= new Multiplica;
$objeto1->opera (-23,11);
$objeto->imprimelo();
$objeto1->imprimelo();
?>
```

Constantes de Clases

Es posible definir valores constantes en función de cada clase manteniéndola invariable. Las constantes se diferencian de variables comunes en que no utilizan el símbolo \$ al declararlas o usarlas. El valor debe ser una expresión constante, no (por ejemplo) una variable, una propiedad, un resultado de una operación matemática, o una llamada a una función.

A partir de PHP 5.3.0, es posible hacer referencia a una clase utilizando una variable. El valor de la variable no puede ser una palabra clave (por ej., *self*, *parent* y *static*).

```
<?php
class MyClass
{
    const constant = 'valor constante';
    function showConstant() {
        echo self::constant . "\n";
    }
}
echo MyClass::constant . "\n";
```

```
$classname = "MyClass";
echo $classname::constant . "\n"; // A partir de PHP 5.3.0

$class = new MyClass();
$class->showConstant();
echo $class::constant."\n"; // A partir de PHP 5.3.0
?>
```

11.2 – Constructores y destructores.

Cuando se define *una función* cuyo nombre es *idéntico* al de la *clase* que la contiene recibe el nombre de **constructor**. Esa función -el **constructor**- se ejecuta de forma *automática* en el momento en que se define un *nuevo objeto (new)*

Según como esté *definido*, el **constructor** puede ejecutarse de distintas formas:

- *Con los valores predefinidos* en las variables de la *clase*.
- Mediante la asignación de *valores preestablecidos* en los propios parámetros de la función *constructor*.

Un constructor puede utilizarse para inicializar una serie de variables y/o procesos necesarios para el buen desarrollo de los objetos.

Cuando se le *pasan* valores la función se ejecuta *sin tomar en consideración* los asignados por defecto en la función y cuando se *lepasan* sólo **parte** de esos valores utiliza **los valores recibidos** y para los *no asignados en la llamada* utiliza los valores del **constructor**

```
<?
class Multiplica{
    var $producto;
    function Multiplica($a=3,$b=7){
        $this->producto=$a*$b;
        print $this->producto."<br>";
    }
}
$objeto= new Multiplica;
$objeto->Multiplica(90,47);
$objeto->Multiplica(47);
$objeto->Multiplica();
?>
```

En el ejemplo anterior vemos como en el caso de faltar algún parámetro cuando se crea el objeto este se coje directamente de los establecidos en el constructor *Multiplica*.

La forma más actual de crear un constructor o un destructor es mediante una método creado a tal efecto siguiendo la sintaxis:

```
void __construct ([ mixed $args [, $... ] ] )
```

Por motivos de compatibilidad, si PHP no puede encontrar una función `__construct()` para una determinada clase y la clase no heredó uno de una clase padre, buscará el viejo estilo de la función constructora, mediante el nombre de la clase. Efectivamente, esto significa que en el único caso en el que se tendría compatibilidad es si la clase tiene un método llamado `__construct()` que fuese utilizado para diferentes propósitos.

PHP permite a los desarrolladores declarar métodos constructores para las clases. Aquellas que tengan un método constructor lo invocarán en cada nuevo objeto creado, lo que lo hace idóneo para cualquier inicialización que el objeto pueda necesitar antes de ser usado.

Nota: Constructores parent no son llamados implícitamente si la clase child define un constructor. Para ejecutar un constructor parent, se requiere invocar a `parent::__construct()` desde el constructor child. Si el child no define un constructor, entonces se puede heredar de la clase padre como un método de clase normal (si no fue declarada como privada).

```
<?php
class BaseClass {
    function __construct() {
        print "En el constructor BaseClass\n";
    }
}

class SubClass extends BaseClass {
    function __construct() {
        parent::__construct();
        print "En el constructor SubClass\n";
    }
}

class OtherSubClass extends BaseClass {
    // heredando el constructor BaseClass
}

// En el constructor BaseClass
$obj = new BaseClass();

// En el constructor BaseClass
// En el constructor SubClass
$obj = new SubClass();

// In BaseClass constructor
$obj = new OtherSubClass();
?>
```

PHP introduce un concepto de **destructor** similar al de otros lenguajes orientados a objetos, tal como C++. El método destructor será llamado tan pronto como no hayan otras referencias a un objeto determinado, o en cualquier otra circunstancia de finalización.

```
<?php
class MyDestructableClass {
    function __construct() {
        print "En el constructor\n";
        $this->name = "MyDestructableClass";
```

```

}

function __destruct() {
    print "Destruyendo " . $this->name . "\n";
}

$obj = new MyDestructableClass();
?>

```

11.3 – Herencia.

En ocasiones, el código no puede reutilizarse tal cual, sino que debe especializarse o refinarse para tener en cuenta nuevas necesidades. Uno podría crear una nueva clase, reescribiendo todo el código reaprovechable en ella e incluir también las nuevas utilidades implementando las funciones necesarias. No obstante, eso limitaría la posible reutilización de código.

Una de las ventajas de la programación orientada a objetos es el uso de la herencia, que soluciona el problema anterior. La herencia permite crear una nueva clase que “herede” las características (variables y métodos) de otra clase.

PHP también tiene la posibilidad de crear **clases extendidas** cuya virtud es poder disponer tanto de *variables y funciones propias* como de *todas las variables y funciones de la clase padre*.

Para crear **clases extendidas** se requiere utilizar la siguiente sintaxis:

```

class nuev extends base {
    ...
    .... definición de variables, constructores y métodos ....
    ...
}

```

Como habrás podido deducir *nuev* es el *nombre de la nueva clase* (*la extendida*), *base* es el nombre de la clase **padre** y **extends** es la *palabra clave* que indica a PHP que se trata de una clase extendida. PHP no permite las *herencias múltiples*. No es posible crear *una clase extendida de otra clase extendida*.

```

<?php
class ExtendClass extends SimpleClass
{
    // Redefinición del método parent
    function displayVar()
    {
        echo "Clase extendida\n";
        parent::displayVar();
    }
}

$extended = new ExtendClass();
$extended->displayVar();
?>

```

11.4 – Visibilidad.

La visibilidad de una propiedad o método se puede definir anteponiendo una de las palabras claves **public**, **protected** o **private** en la declaración. Miembros de clases declarados como **public** pueden ser accedidos de cualquier lado. Miembros declarados como **protected**, sólo de la clase misma, por herencia y clases parent. Aquellos definidos como **private**, únicamente de la clase que los definió.

Mientras no se especifique otra cosa, los métodos y propiedades de una clase son siempre públicos, es decir, son accesibles desde fuera del objeto de la forma: objeto → función.

Declaración de propiedades

```
<?php
/** Definición de MyClass ***/
class MyClass
{
    public $public = 'Public';
    protected $protected = 'Protected';
    private $private = 'Private';

    function printHello()
    {
        echo $this->public;
        echo $this->protected;
        echo $this->private;
    }
}

$obj = new MyClass();
echo $obj->public; // Funciona
echo $obj->protected; // Error Fatal
echo $obj->private; // Error Fatal
$obj->printHello(); // Muestra Public, Protected y Private

/** Definición de MyClass2 ***/
class MyClass2 extends MyClass
{
    // Se puede redeclarar los métodos public y protected, pero no el private
    protected $protected = 'Protected2';
    function printHello()
    {
        echo $this->public;
        echo $this->protected;
        echo $this->private;
    }
}

$obj2 = new MyClass2();
echo $obj2->public; // Funciona
echo $obj2->private; // Undefined
echo $obj2->protected; // Error Fatal
$obj2->printHello(); // Muestra Public, Protected2, Undefined

?>
```

Declaración de métodos

```
<?php
/** Definición de MyClass ***/
class MyClass
{
    // Declaración de un constructor public
    public function __construct() { }

    // Declaración de un método public
    public function MyPublic() { }

    // Declaración de un método protected
    protected function MyProtected() { }

    // Declaración de un método private
    private function MyPrivate() { }

    // Esto es public
    function Foo()
    {
        $this->MyPublic();
        $this->MyProtected();
        $this->MyPrivate();
    }
}

$myclass = new MyClass;
$myclass->MyPublic(); // Funciona
$myclass->MyProtected(); // Error Fatal
$myclass->MyPrivate(); // Error Fatal
$myclass->Foo(); // Public, Protected y Private funcionan

/** Definición de MyClass2 ***/
class MyClass2 extends MyClass
{
    // Esto es public
    function Foo2()
    {
        $this->MyPublic();
        $this->MyProtected();
        $this->MyPrivate(); // Error Fatal
    }
}

$myclass2 = new MyClass2;
$myclass2->MyPublic(); // Funciona
$myclass2->Foo2(); // Public y Protected funcionan, pero Private no
```

11.5 – Operador de Resolución de Ámbito (::).

El Operador de Resolución de Ámbito (también denominado Paamayim Nekudotayim) o en términos simples, el doble dos-puntos, es un token que permite acceder a elementos [estáticos](#), [constantes](#), y sobrescribir propiedades o métodos de una clase. Cuando se hace referencia a estos items desde el exterior de la definición de la clase, se utiliza el nombre de la clase. El valor de la variable no puede ser una palabra clave (por ej., *self*, *parent* y *static*).

Desde el exterior de la definición de la clase

```
<?php
class MyClass {
    const CONST_VALUE = 'Un valor constante';
}

$classname = 'MyClass';
echo $classname::CONST_VALUE; // A partir de PHP 5.3.0

echo MyClass::CONST_VALUE;
?>
```

Las tres palabras claves especiales *self*, *parent* y *static* son utilizadas para acceder a propiedades y métodos desde el interior de la definición de la clase.

Desde el interior de la definición de la clase

```
<?php
class OtherClass extends MyClass
{
    public static $my_static = 'variable estática';

    public static function doubleColon() {
        echo parent::CONST_VALUE . "\n";
        echo self::$my_static . "\n";
    }
}

$classname = 'OtherClass';
echo $classname::doubleColon(); // A partir de PHP 5.3.0

OtherClass::doubleColon();
?>
```

Cuando una clase extendida sobrescribe la definición parent de un método, PHP no invocará al método parent. Depende de la clase extendida el hecho de llamar o no al método parent. Esto también se aplica a definiciones de métodos [Constructores y Destructores](#), [Sobrecarga](#), y [Mágicos](#).

Invocando a un método parent

```
<?php
class MyClass
{
    protected function myFunc() {
```

```

        echo "MyClass::myFunc()\n";
    }

class OtherClass extends MyClass
{
    // Sobrescritura de definición parent
    public function myFunc()
    {
        // Pero todavía se puede llamar a la función parent
        parent::myFunc();
        echo "OtherClass::myFunc()\n";
    }
}

$class = new OtherClass();
$class->myFunc();
?>

```

11.6 – La palabra clave 'static'.

Declarar propiedades o métodos de clases como estáticos los hacen accesibles sin la necesidad de instanciar la clase. Una propiedad declarada como static no puede ser accedida con un objeto de clase instanciado (aunque un método estático sí lo puede hacer).

Debido a que los métodos estáticos se pueden invocar sin tener creada una instancia del objeto, la pseudo-variable `$this` no está disponible dentro de los métodos declarados como estáticos. Las propiedades estáticas no pueden ser accedidas a través del objeto utilizando el operador flecha (`->`).

Ejemplo de propiedad estática

```

<?php
class Foo
{
    public static $mi_static = 'foo';

    public function valorStatic() {
        return self::$mi_static;
    }
}

class Bar extends Foo
{
    public function fooStatic() {
        return parent::$mi_static;
    }
}

print Foo::$mi_static . "\n";

$foo = new Foo();
print $foo->valorStatic() . "\n";

```

```

print $foo->mi_static . "\n"; // "Propiedad" mi_static no definida

print $foo::$mi_static . "\n";
$nombreClase = 'Foo';
print $nombreClase::$mi_static . "\n"; // A partir de PHP 5.3.0

print Bar::$mi_static . "\n";
$bar = new Bar();
print $bar->fooStatic() . "\n";
?>

```

Ejemplo de método estático

```

<?php
class Foo {
    public static function unMétodoEstático() {
        // ...
    }
}

Foo::unMétodoEstático();
$nombreClase = 'Foo';
$nombreClase::unMétodoEstático(); // A partir de PHP 5.3.0
?>

```

11.7 – Abstracción de clases.

PHP introduce clases y métodos abstractos. Las clases definidas como abstract seguramente no son instanciadas y cualquier clase que contiene al menos un método abstracto debe ser definida como abstract. Los métodos definidos como abstractos simplemente declaran la estructura del método, pero no pueden definir la implementación.

Cuando se hereda de una clase abstracta, todos los métodos definidos como abstract en la definición de la clase parent deben ser redefinidos en la clase child; adicionalmente, estos métodos deben ser definidos con la misma [visibilidad](#) (o con una menos restrictiva). Por ejemplo, si el método abstracto está definido como protected, la implementación de la función puede ser redefinida como protected o public, pero nunca como private. Por otra parte, las estructuras de los métodos tienen que coincidir; es decir, la implicación de tipos y el número de argumentos requeridos deben ser los mismos. Por ejemplo, si la clase derivada define un parámetro opcional, mientras que el método abstracto no, no habría conflicto con la estructura del método.

Ejemplo de clase abstracta

```

<?php
abstract class AbstractClass
{
    // Forzando la extensión de clase para definir este método
    abstract protected function getValue();
    abstract protected function prefixValue($prefix);

    // Método común

```

```

public function printOut() {
    print $this->getValue() . "\n";
}
}

class ConcreteClass1 extends AbstractClass
{
    protected function getValue() {
        return "ConcreteClass1";
    }

    public function prefixValue($prefix) {
        return "{$prefix}ConcreteClass1";
    }
}

class ConcreteClass2 extends AbstractClass
{
    public function getValue() {
        return "ConcreteClass2";
    }

    public function prefixValue($prefix) {
        return "{$prefix}ConcreteClass2";
    }
}

$class1 = new ConcreteClass1;
$class1->printOut();
echo $class1->prefixValue('FOO_') ."\n";

$class2 = new ConcreteClass2;
$class2->printOut();
echo $class2->prefixValue('FOO_') ."\n";
?>

```

El resultado del ejemplo sería:

```

ConcreteClass1
FOO_ConcreteClass1
ConcreteClass2
FOO_ConcreteClass2

```

11.8 – Interfaces de objetos.

En proyectos profesionales a veces es necesario que un equipo de varias personas trabajen juntas. En este caso se hace imprescindible definir unas pautas generales de trabajo para que el resultado final sea el esperado. Si el desarrollo consiste en programar varios objetos, el analista de la aplicación puede definir la estructura básica en papel o crear una pequeña plantilla con métodos que el objeto final debería tener obligatoriamente. Esta plantilla es un **interface** y permite definir una clase con funciones definidas, pero sin desarrollar, que obliga a todas las clases que lo implementen a declarar estos métodos como mínimo..

Ejemplo de interfaz

```

<?php

// Declarar la interfaz 'iTemplate'
interface iTemplate
{
    public function setVariable($name, $var);
    public function getHtml($template);
}

// Implementar la interfaz
// Ésto funcionará
class Template implements iTemplate
{
    private $vars = array();

    public function setVariable($name, $var)
    {
        $this->vars[$name] = $var;
    }

    public function getHtml($template)
    {
        foreach($this->vars as $name => $value) {
            $template = str_replace('{' . $name . '}', $value, $template);
        }

        return $template;
    }
}
?>

```

11.9 – Traits.

Los traits (rasgos) son un mecanismo de reutilización de código en lenguajes de herencia simple, como PHP. El objetivo de un trait es el de reducir las limitaciones propias de la herencia simple permitiendo que los desarrolladores reutilicen a voluntad conjuntos de métodos sobre varias clases independientes y pertenecientes a clases jerárquicas distintas. La semántica a la hora combinar Traits y clases se define de tal manera que reduzca su complejidad y se eviten los problemas típicos asociados a la herencia múltiple y a los Mixins.

Un Trait es similar a una clase, pero con el único objetivo de agrupar funcionalidades muy específicas y de una manera coherente. No se puede instanciar directamente un Trait. Es por tanto un añadido a la herencia tradicional, y habilita la composición horizontal de comportamientos; es decir, permite combinar miembros de clases sin tener que usar herencia.

Ejemplo de Trait

```

<?php
trait ezcReflectionReturnInfo {
    function getReturnType() { /*1*/ }
    function getReturnDescription() { /*2*/ }
}

```

```

class ezcReflectionMethod extends ReflectionMethod {
    use ezcReflectionReturnInfo;
    /* ... */
}

class ezcReflectionFunction extends ReflectionFunction {
    use ezcReflectionReturnInfo;
    /* ... */
}
?>
```

11.10 – Iteración de objetos.

PHP ofrece una manera para que los objetos sean definidos por lo que es posible iterar a través de una lista de elementos, con, por ejemplo, una sentencia `foreach`. Por defecto, todas las propiedades visibles serán utilizados para la iteración.

```

<?php
class MyClass
{
    public $var1 = 'value 1';
    public $var2 = 'value 2';
    public $var3 = 'value 3';
    protected $protected = 'protected var';
    private $private = 'private var';
    function iterateVisible() {
        echo "MyClass::iterateVisible:\n";
        foreach($this as $key => $value) {
            print "$key => $value\n";
        }
    }
}

$class = new MyClass();

foreach($class as $key => $value) {
    print "$key => $value\n";
}
echo "\n";
$class->iterateVisible();
?>
```

El resultado del ejemplo sería:

```

var1 => value 1
var2 => value 2
var3 => value 3
```

```

MyClass::iterateVisible:
var1 => value 1
var2 => value 2
var3 => value 3
protected => protected var
private => private var
```

11.11 – Palabra clave Final.

PHP introduce la nueva palabra clave final, que impide que las clases hijas sobrescriban un método, antecediendo su definición con la palabra final. Si la propia clase se define como final, entonces no se podrá heredar de ella.

```
<?php
class BaseClass {
    public function test() {
        echo "llamada a BaseClass::test()\n";
    }

    final public function moreTesting() {
        echo "llamada a BaseClass::moreTesting()\n";
    }
}

class ChildClass extends BaseClass {
    public function moreTesting() {
        echo "llamada a ChildClass::moreTesting()\n";
    }
}
// Devuelve un error Fatal: Cannot override final method BaseClass::moreTesting()
?>
```

11.12 – Funciones de clases y/o objetos.

Las siguientes funciones son útiles para recuperar información sobre la herencia de las clases, sobre métodos o variables miembro o llamadas a las funciones.

- **`__autoload`** — Intenta cargar una clase sin definir
- **`class_alias`** — Crea un alias para una clase
- **`class_exists`** — Verifica si la clase ha sido definida
- **`get_called_class`** — El nombre de la clase "Vinculante Static Última"
- **`get_class_methods`** — Obtiene los nombres de los métodos de una clase
- **`get_class_vars`** — Obtener las propiedades predeterminadas de una clase
- **`get_class`** — Devuelve el nombre de la clase de un objeto
- **`get_declared_classes`** — Devuelve una matriz con los nombres de las clases definidas
- **`get_declared_interfaces`** — Devuelve un array con todas las interfaces declaradas
- **`get_declared_traits`** — Devuelve un array de todos los traits declarados
- **`get_object_vars`** — Obtiene las propiedades del objeto dado
- **`get_parent_class`** — Recupera el nombre de la clase padre de un objeto o clase
- **`interface_exists`** — Comprueba si una interfaz ha sido definida
- **`is_a`** — Comprueba si un objeto es de una clase o tiene esta clase como una de sus madres
- **`is_subclass_of`** — Verifica si el objeto tiene esta clase como uno de sus padres
- **`method_exists`** — Comprueba si existe un método de una clase
- **`property_exists`** — Comprueba si el objeto o la clase tienen una propiedad
- **`trait_exists`** — Comprobar si el trait existe

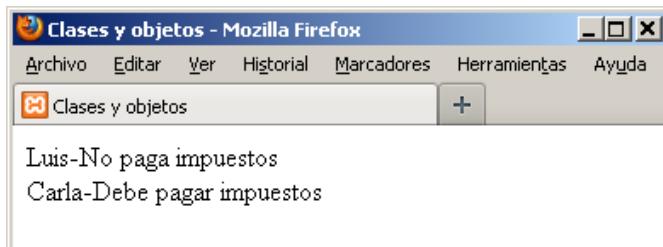
Actividades.

UD11 – ACTIVIDAD 1: Empezando con clases y objetos.

TIPO	Desarrollo
OBJETIVOS	Practicar la declaración de una clase y creación de un objeto.
RECURSOS	Editor de texto y navegador web.

ENUNCIADO DE LA ACTIVIDAD

Confeccionar una clase Empleado, definir como atributos su nombre y sueldo. Definir un método inicializar al que lleguen como dato el nombre y sueldo. Plantear un segundo método que imprima el nombre y un mensaje si debe o no pagar impuestos (si el sueldo supera a 3000 paga impuestos).


COMENTARIOS

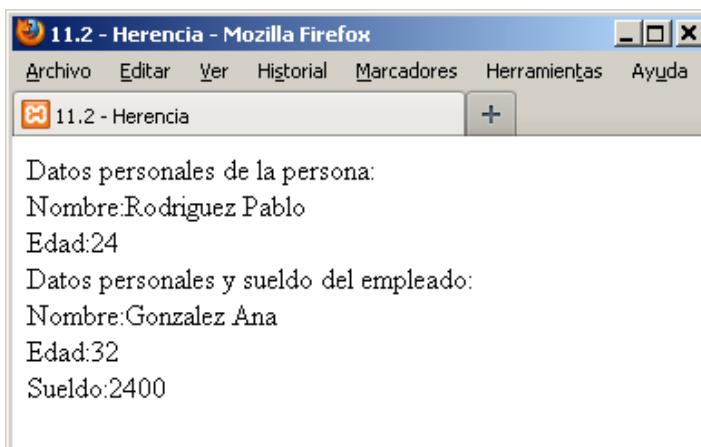
Definir los atributos nombre y sueldo como privados para que no se puedan acceder a ellos más que por medio de los métodos que se definan.

UD11 – ACTIVIDAD 2: Jugando con la herencia.

TIPO	Desarrollo
OBJETIVOS	Practicar la herencia en la estructura de clases en PHP.
RECURSOS	Editor de texto y navegador web.

ENUNCIADO DE LA ACTIVIDAD

Confeccionar una clase Persona que tenga como atributos el nombre y la edad. Definir un método que cargue los datos personales y otro que los imprima. Plantear una segunda clase Empleado que herede de la clase Persona. Añadir un atributo sueldo y los métodos de cargar el sueldo e imprimir su sueldo. Definir un objeto de la clase Persona y llamar a sus métodos. También crear un objeto de la clase Empleado y llamar a sus métodos.


COMENTARIOS

Fíjate bien como se comporta el objeto Empleado y como es capaz de acceder a las propiedades y métodos de la clase Persona de la que hereda.

UD11 – ACTIVIDAD 3: Sobreescribiendo métodos.

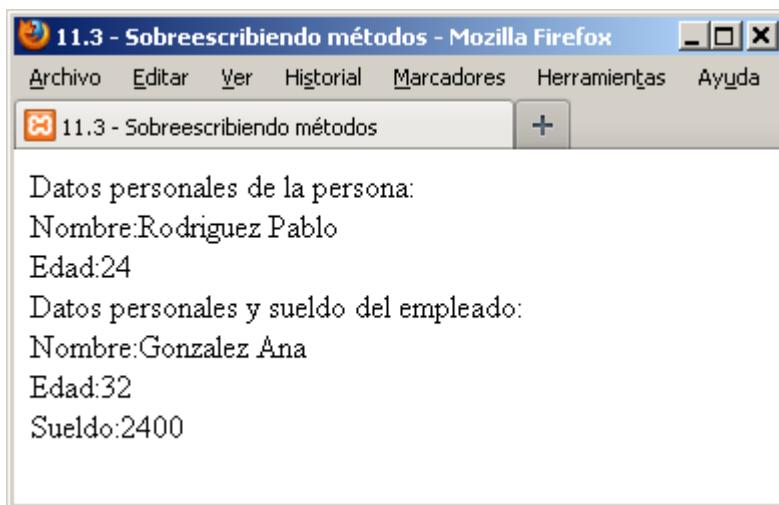
TIPO	Desarrollo
OBJETIVOS	Practicar la sobrescritura de métodos en PHP.
RECURSOS	Editor de texto y navegador web.

ENUNCIADO DE LA ACTIVIDAD

Confeccionar una clase Persona que tenga como atributos el nombre y la edad. Definir como responsabilidades un método que cargue los datos personales y otro que los imprima.

Plantear una segunda clase Empleado que herede de la clase Persona. Añadir un atributo sueldo y los métodos de cargar el sueldo e imprimir todos los datos del empleado (sobreescribir el método imprimir de la clase Persona).

Definir un objeto de la clase Persona y llamar a sus métodos. También crear un objeto de la clase Empleado y llamar a sus métodos.

**COMENTARIOS**

Recordad que cuando se sobreescribe un método de una clase que hereda de una clase superior, esta normalmente debe llamar a la clase padre del siguiente modo:
parent::método_sobreescrito();

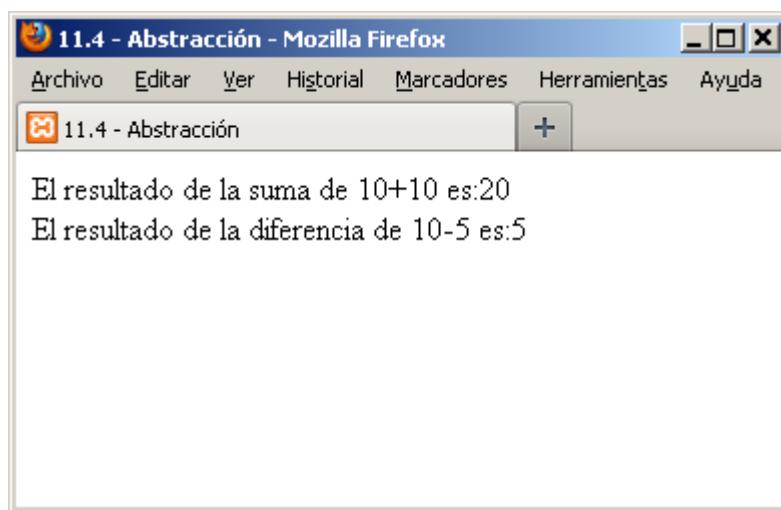
UD11 – ACTIVIDAD 4: Abstracción.

TIPO	Desarrollo
OBJETIVOS	Practicar el uso de métodos y clases abstractas en PHP.
RECURSOS	Editor de texto y navegador web.

ENUNCIADO DE LA ACTIVIDAD

Confeccionar una clase abstracta llamada Operacion que defina como atributos \$valor1, \$valor2, \$resultado y defina como métodos cargar1 (inicializa el atributo \$valor1), cargar2 (inicializa el atributo \$valor2), un método que muestre el contenido de \$resultado y por último definir un método abstracto llamado operar.

Luego definir dos subclases concretas de la clase Operacion. La primera llamada Suma que tiene por objetivo la carga de dos valores, sumarlos y mostrar el resultado. La segunda llamada Resta que tiene por objetivo la carga de dos valores, restarlos y mostrar el resultado de la diferencia.

**COMENTARIOS**

Recordad que cuando se sobreescribe un método de una clase que hereda de una clase superior, esta normalmente debe llamar a la clase padre del siguiente modo:
parent::método_sobreescrito();

12 – Publicación de un sitio web.

El paso siguiente, una vez que nuestra web está terminada, es el de transferir los archivos a un servidor web para que pasen a estar disponibles para cualquier persona que quiera utilizarlos.

El modelo que se sigue es sencillo; para que una página pueda ser visualizada por cualquier persona solemos recurrir a los servidores web, por varios motivos:

- Un servidor web es un ordenador (simplificando un poco) que tiene presencia en Internet, es decir, que tiene una dirección IP asociada, además de un nombre fácilmente identificable asociado a esa dirección IP (del tipo www.elservidor.com). Con esto nos aseguramos de que lo que coloquemos en ese servidor será fácilmente localizable mediante una URL concreta, que se compondrá del nombre del servidor más los nombres de las carpetas que se nos asignen, generalmente.

- Un servidor web está permanentemente encendido, todas las horas del día durante todo el año. Imaginemos que vamos a consultar nuestro periódico favorito y nos encontramos con que no podemos hacerlo, porque alguien ha apagado el servidor que lo aloja. No es un modelo de funcionamiento que tenga sentido, ya que no se sabe a qué hora vendrán nuestros usuarios. Así que los servidores de Internet funcionan a todas horas.

- Un servidor web suele contar con un buen ancho de banda, capaz de dar respuesta a muchas peticiones simultáneas. Así, si nuestro sitio web se vuelve un éxito y tenemos centenares de visitas por minuto, el servidor será capaz de transmitir nuestras páginas a los usuarios correctamente. Si esto lo tuviésemos que hacer con nuestro ordenador, sería inviable y los usuarios no podrían acceder al sitio.

Los servidores web tienen instalado una aplicación denominada también **servidor de páginas web**, un programa que está permanentemente escuchando para ver si le llegan peticiones. Cuando un usuario desde su ordenador abre el navegador e intenta acceder a una página alojada en ese servidor, la aplicación recibirá la petición, localizará la página y los recursos asociados y se los transferirá al usuario (esto se hace mediante una serie de normas denominadas protocolo http, por eso se suele anteceder las direcciones web de <http://>), que a su vez verá la página en su navegador.

Hay diferentes aplicaciones para servir páginas web. Una de las más extendidas es Apache, que además es software libre, por lo que podríamos llegar a montar nuestro pequeño servidor de pruebas, si fuese necesario, aunque no es el caso.

La transferencia de ficheros.

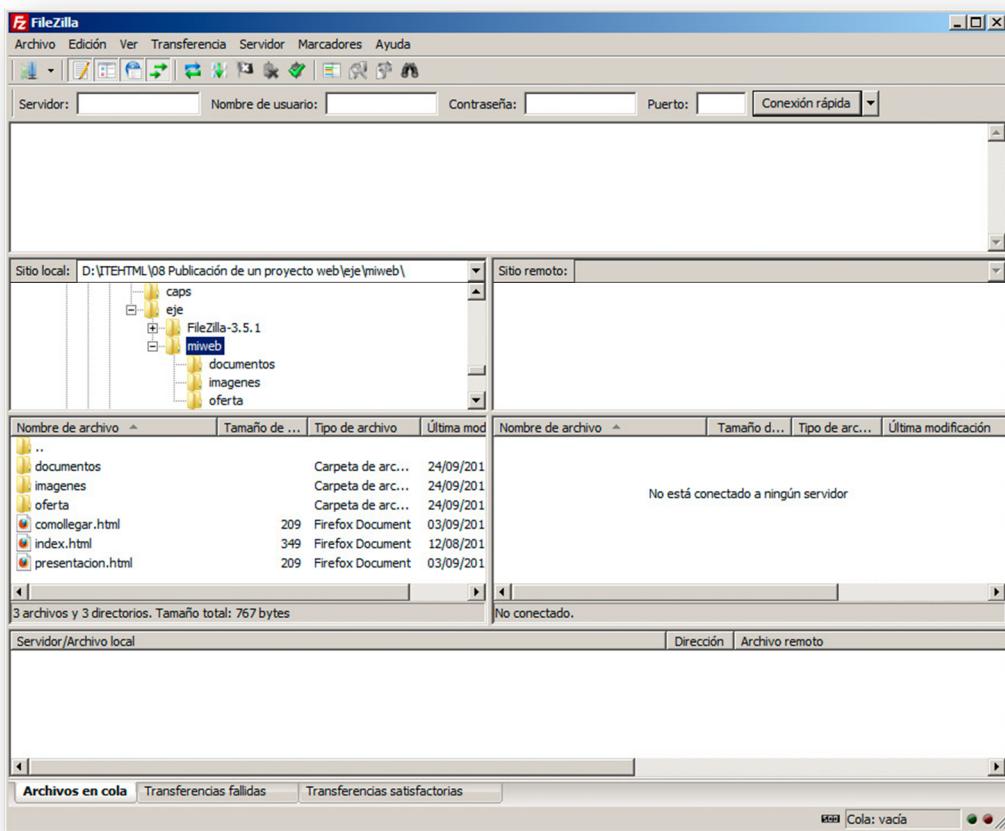
Hasta aquí está claro, pero, ¿cómo envío mis archivos php, html, mis imágenes y mis cosas al servidor de páginas web? Aunque algunos servidores ofrecen métodos sencillos basados en el navegador, el método más extendido es transferir los archivos mediante un conjunto de normas identificadas como **protocolo de transferencia de archivos** (que son las siglas de **FTP**).

Este matiz nos indica que para enviar las páginas web al servidor, normalmente emplearemos una aplicación de transferencia FTP. En nuestro caso emplearemos para

nuestras pruebas **Filezilla** <http://filezilla-project.org/>, que es software libre y multiplataforma, por lo que la podremos utilizar sin coste alguno desde cualquier sistema operativo.

En la página de Filezilla encontraremos también para Windows una versión zip, que ni siquiera requiere instalación; basta con descomprimirla y ejecutarla.

Una vez descargada, procederemos a instalarla con los métodos habituales. Filezilla nos mostrará su pantalla inicial, recogida en la figura:



Tras elegir el idioma, podremos empezar a trabajar con normalidad. El espacio se distribuye de la siguiente manera:

En la parte izquierda encontramos un explorador de archivos que nos muestra el contenido de nuestro ordenador. Con él podremos desplazarnos hasta la carpeta que contiene nuestro sitio web.

La parte derecha nos mostrará las carpetas de nuestro servidor web. Allí aparecerán los archivos que hayamos transferido y podremos editarlos, modificarlos, borrarlos o realizar cualquier operación que necesitemos.

El panel superior presenta los mensajes que emite el servidor. Es información administrativa, que sólo nos interesará si se produce alguna incidencia.

En el panel inferior veremos información sobre las transferencias de archivos que queden pendientes, errores al transmitir algún archivo, etc.

Por tanto el procedimiento se resume en tres pasos:

1. Conectar con el servidor web, llamado normalmente el servidor remoto.
2. Seleccionar los archivos que nos interesan en el panel local, el panel de la izquierda.
3. Transferirlos al servidor, arrastrándolos hasta el panel derecho.

Tras realizar esos pasos, nuestra web estará ya visible desde la dirección http:// correspondiente.

Conectar con el servidor remoto.

Para conectar con un servidor remoto, previamente tenemos que tener una cuenta en ese servidor. Una cuenta es un nombre de usuario y contraseña, que nos darán acceso a una carpeta concreta del servidor.

En muchos casos nuestro proveedor de Internet nos facilita espacio web en sus servidores; en otros, lo hacen nuestras propias empresas de trabajo. Si no sucede así, siempre podremos recurrir a los servidores web gratuitos. Hay multitud de servidores web que facilitan espacios gratuitos con ciertas limitaciones o publicidad para las personas o empresas que quieren crear sus primeras páginas web. No pondremos ningún ejemplo, porque resulta tan sencillo como buscar servidor web gratuito en Internet.

Si buscamos servidores gratuitos en Internet, también encontraremos muchos resultados empleando el término inglés, hosting o host, que son los términos que se emplean para describir el alojamiento o el servidor web.

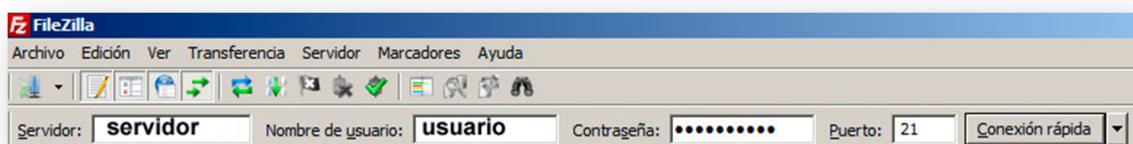
Independientemente del tipo de servidor por el que optemos, al final debemos contar con tres datos imprescindibles:

- **Dirección del servidor FTP:** la URL con la que accederemos al servidor.
- **Nombre de usuario:** el nombre de nuestra cuenta.
- **Contraseña:** la clave para acceder a la cuenta.

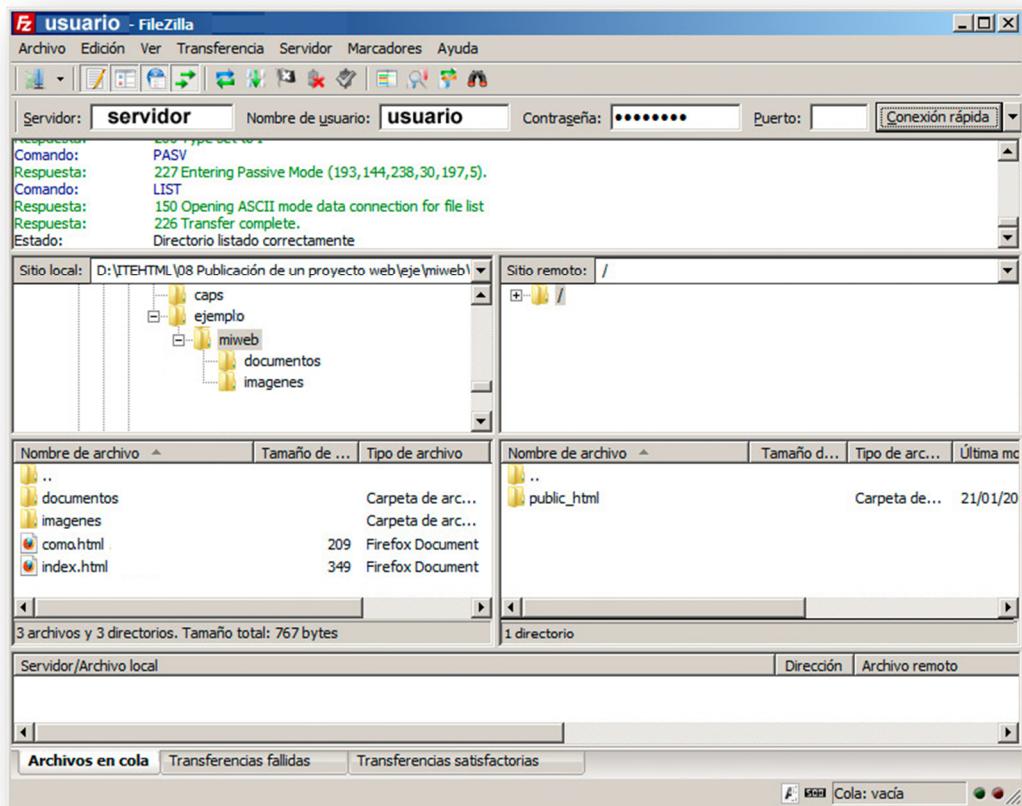
Con esos tres datos seremos capaces de conectarnos a nuestro servidor de ftp en el 99% de los casos. En ocasiones puede que nos indiquen un par de datos más:

- **Carpeta del servidor:** ruta en la que debemos subir los archivos. Casi nunca se indica porque el propio servidor ya nos lleva a la carpeta apropiada.
- **Puerto FTP:** en algunos servidores hay que indicar este número, que es punto en el que el servidor escucha las peticiones que le llegan para realizar transferencias FTP. Normalmente no se indica porque es casi siempre el número 21.

Volviendo a Filezilla, encontramos que la información anterior es la que podemos introducir directamente en la barra superior, recogida en la figura:



Al hacer clic en el botón Conexión rápida, Filezilla comenzará la conexión. Si los datos introducidos son correctos, en el panel de la derecha veremos ya las carpetas del servidor, con los archivos que pueda haber. Si hay algún problema, veremos mensajes de color rojo en el panel superior indicando la respuesta errónea por parte del servidor. La figura muestra una conexión correcta:



Transferir los archivos.

La transferencia de archivos consiste simplemente en arrastrar los archivos situados en la parte izquierda a la parte derecha. A esta transferencia se le suele llamar coloquialmente subir los archivos. Este proceso puede tardar un tiempo, si hemos empleado muchos archivos o muy grandes. El panel inferior nos va mostrando información del proceso.

Se guarda un registro de la transferencia en el panel inferior, indicado por Transferencias fallidas y Transferencias satisfactorias. En fallidas no debería haber ningún archivo, si todo ha ido bien.

Al finalizar, tendremos en el servidor remoto una copia exacta del contenido de nuestro ordenador.

En este proceso sólo necesitaremos tener una precaución: la estructura de los archivos debe mantenerse exactamente igual que la local, es decir, si habíamos distribuido los archivos por carpetas, debemos mantener el mismo sistema.

No debemos olvidar que nuestra página principal debe llamarse index.php (en un servidor Windows), default.php (en un servidor Linux), index.htm o index.html, ya que ésa

será la que se cargue automáticamente por parte del servidor, cuando alguien acceda a nuestro espacio.

En este punto ya podemos acceder al navegador e introducir la dirección correspondiente para ver nuestro espacio web. La dirección nos la facilita también el servicio de almacenamiento que estemos empleando.

La siguiente imagen resumen los pasos del proceso a seguir.



ANEXOS

I – Usabilidad.

Llamamos **usabilidad** a la experiencia que tiene un usuario cuando interactúa con páginas de un web.

Un sitio web al cual le denominamos "usable", es aquel que de una manera clara un usuario entiende el contenido y navega por el web de una forma cómoda y sencilla. Aunque esto no es siempre fácil por problemas de contenido, un diseñador web siempre debe procurar realizar webs claros y fácilmente navegables por el usuario que nos va a visitar, de manera que el usuario disponga de la información que le queremos hacer llegar de una manera clara y sencilla.

La usabilidad es la combinación de los factores que afectan al usuario cuando interactúa con las páginas web, estos pueden ser entre otros:

Facilidad de aprender. Las páginas web han de proporcionar a un usuario que nunca ha visto antes una página web o interfaz, la manera de navegar por nuestros contenidos y de ofrecerle nuestros productos de una manera rápida y sencilla.

Eficacia del uso. Una vez que un usuario ya ha experimentado o aprendido a utilizar el sistema o página web, cómo rápidamente puede él o ella lograr tareas.

Frecuencia y severidad del error. A menudo los usuarios producen errores mientras navegan por las páginas web, debemos analizar qué tipo de errores se pueden producir y la manera de recuperar a ese cliente al que el sistema le ha fallado.

Navegación por nuestro web. Debemos plantearnos si nuestros visitantes están encontrando la información que nosotros queremos hacerles llegar, o si bien, si estos usuarios están lincando a secciones o páginas que no son tan importantes para nosotros como pueden ser nuestros productos, formularios de contacto, etc.

Las páginas web han de proporcionar a un usuario que nunca ha visto antes una página web o interfaz, la manera de navegar por nuestros contenidos y de ofrecerle nuestros productos de una manera rápida y sencilla.

Consejos de usabilidad

Lo principal es entender a los usuarios que nos visitan:

- Entienda quién es el usuario y analice cualquier característica relevante.
- Entienda las metas y los objetivos del usuario que visita nuestra página.
- Analice las situaciones que se presentan comúnmente como parte de las actividades normales del usuario.
 - Entienda las exigencias y las preferencias del consumidor.
 - Debe resolver metas y requisitos específicos del usuario. Su web debe permitir a los usuarios resolver sus requisitos totales de una manera que sea eficaz y eficiente.
 - Estructure su interfaz y diseño de navegación de una manera que sea intuitiva a las metas del usuario, a las tareas y a los procesos de los usuarios.

- Dé la prioridad a las tareas y a las acciones de los usuarios de modo que estas acciones sean más fácilmente accesibles.
- Haga los objetos y los botones, etc obvios. Esto debe permitir a usuarios aprender y utilizar rápidamente y más eficientemente su sistema.
- Mantenga su diseño simple, no estorbe y no lo distraiga. No confunda ni pierda a sus usuarios y 'no fuerce' a sus usuarios a centrarse en las áreas de la pantalla que no son relevantes.
- Muestre a los usuarios que está haciendo el sistema y qué espera de ellos.

Reglas de usabilidad web

Existen 5 principales reglas que adaptadas a un web, se le puede considerar como un web usable.

Rápido. Un site sólo capta la atención de un usuario durante los primeros 8 segundos que el usuario esta delante de la página web, pasado este tiempo, si el usuario no encuentra la información que esta buscando, cancelará y se ira a otro web. Las páginas deben cargarse en una media de 4 segundos. Lo más que los usuarios esperarán en ver el contendio de una página web es de una media de 10 segundos. La mayoría de los usuarios disponen de moden para su acceso a internet, por lo que nuestras páginas deben de ser lo menos pesadas posibles con el fin de que los usuarios no esperen más tiempo de lo deseado. Si no estos cancelaran la visita.

Simple. Limite la navegación de su web a 6 y 8 páginas como mucho. Los estudios demuestran que es el número máximo que el usuario puede mantener en la memoria a corto plazo. Mantenga una navegación constante. No fuerce a los visitantes a aprender diversos caminos o esquemas para la navegación en diversas partes de su site. No abuse de la utilización de la animación, esto puede abrumar y cansar a la vista.

"Investigable". Los motores de búsqueda buscan el texto real. No prestan ninguna atención a los gráficos (incluso gráficos que parecen texto) y al código de programación (como el Javascript, usado para los menús y otros efectos especiales). Evite estas situaciones si desea que su web este bien posicionada en los buscadores.

Compatible. Los sites necesitan ser compatibles con todos los navegadores y ordenadores. Utilice HTML simple y llano siempre que sea posible, es el más compatible con todos los navegadores.

Actualizado. La manera más rápida para que un web pierda credibilidad es contener la información anticuada. Incluso cosas pequeñas como una fecha del copyright de "2000", etc, pueden dañar la credibilidad del web además de su contenido.

Colores y sus asociaciones naturales

Los colores definen el tipo de web de cara al usuario.

- El rojo se asocia a sangre, y a las sensaciones que son enérgicas, excitando, apasionado o erótico. La mayoría de los colores llevan implicaciones positivas y negativas. El downside del rojo evoca sensaciones agresivas, sugiriendo cólera o violencia.

- El naranja es el color de la carne, o el calor amistoso del fuego del hogar. Las implicaciones positivas de este color sugieren el approachability. El lado negativo puede implicar accesibilidad al punto de sugerir que cualquier persona puede acercarse.
- El amarillo es el color del sol. Este color es optimista, moderno. La energía del amarillo puede llegar a ser abrumadora. Por lo tanto el amarillo no debe ser un color que tienda a dominar en un web.
- El verde, es positivo, sugiere la naturaleza (vida, bosques, plantas), vida, estabilidad.
- El azul seriedad, espiritualidad, elegancia.
- El violeta es el color de la fantasía, de la alegría, del impulso y de estados ideales. En su modo negativo, puede sugerir pesadillas, o locura.

Coloréelo blanco. Utilice el color blanco.

Este color actúa como equilibrio maravilloso entre los colores, hace las páginas agradables a la visión dejando espacio blanco, "da sensación de elegancia", espacio vacío entre los elementos de la página, especialmente si su sitio es rico en contenido (texto).

Cuando se proponga diseñar su página web, intente limitar su gama de colores a 2 o 3 colores importantes como mucho (con variaciones si procede de tonos). Los websites importantes han limitado el uso de colores a 2 colores nada más.

Las fuentes en usabilidad

Las fuentes se utilizan para crear la mayoría de los tipos de elementos del web: títulos, descripciones, links, barras de la navegación, menús, botones, listas, tablas, etc.

En el código HTML, las fuentes se describen en términos de: forma, estilo, tamaño y color. Todos estos atributos se pueden utilizar como cualidades editables dentro de la etiqueta FUENTE. A modo de ejemplo analicemos dos fuentes:

- **Serif:** Estas fuentes tienen accesorios pequeños en las letras. En tipografía estándar, éstas son las letras preferidas para los bloques de texto grandes, puesto que las fuentes de texto Serifs se hacen más fácil de leer en líneas o párrafos largos.
- **Sans-serif** - Versión Imprimible (Pdf): Estas fuentes consisten solamente en "linestrokes" y por lo tanto son más simples en la forma (e.g., Helvetica, Arial, Futura).
- Utilice la "**negrita**" de forma no abusiva, este atributo es altamente visible y por lo tanto puede llegar a ser visualmente intrusiva y desagradable. El uso de "italica" se puede utilizar para definir términos o para acentuar una palabra ocasionalmente, pero no debe ser frecuente el utilizarla, puesto que no es muy legible en la pantalla del ordenador y puede dar sensación de mareo.

Tamaño y color de fuente

El tamaño de fuente se puede especificar en código HTML en términos relativos o absolutos. Evite los tamaños de fuente absolutos, es decir, en vez de especificar una fuente como SIZE="14pt ", utilice SIZE="+1" o SIZE="200%" .

Mejor todavía, utilice las etiquetas del estilo de HTML (e.g., H2), puesto que estas etiquetas aplican estándares de tipografía para variar el tamaño del texto con incrementos equitativos.

El color de la fuente se refiere, por supuesto, al color usado para dibujar el texto. El color se debe elegir cuidadosamente, para maximizar la legibilidad contra su color de fondo. Si el fondo de su web es claro - que recomendamos - ésto significa un color negro o una fuente en un color oscuro para facilitar su legibilidad frente al usuario.

Si el color de fondo es oscuro, el texto será más difícil de leer, si no utilizamos un color para la fuente que contraste con este color de fondo; el color debe ser blanco, amarillo pálido, naranja pálido. Finalmente, no utilice demasiada variación de fuentes (Arial, Verdana, Times, etc), esto distraerá y corre el riesgo de que los usuarios no asimilen completamente su contenido.

Links, el camino más eficiente

Los link son el soporte de navegación más común de los website, recuerde que si los usuarios se sienten demasiado confusos, abandonarán su sitio web. Por lo tanto, es importante proporcionar ayudas estructurales claras a la navegación.

Apariencia de los links

Una web usa links para proporcionar el acceso rápido a otras partes del sitio:

- Para señalar e indicar a los usuarios las páginas con más información sobre el texto o el gráfico mencionado en el link.
- Para señalar a los usuarios una alternativa, en caso de que la página actual que están viendo no sea lo que él este buscando (éstos se expresan típicamente como "y además", o también, etc..).
- Los usuarios esperan que los links tengan un aspecto fiable.

Donde y como usar los links

El texto de los links se debe elegir cuidadosamente, puesto que su color y el subrayar automáticamente hacen de los links la manera más fiable de navegación.

Los links pueden ser extremadamente eficaces cuando destacan visualmente las palabras o los conceptos más importantes en la página, al mismo tiempo que proporcionan el acceso a más información sobre ellos. Evite las frases sin sentido como "haz click aquí" o "para ver la información sobre"; en su lugar, indíquele al usuario una frase corta que indique la información que se esconde "al otro lado".

Para mejorar la legibilidad de los links, subraye las palabras que realmente importan, no el título entero de un documento. Una atención especial es necesaria cuando los links aparecen en una lista. Agrupe los acoplamientos en categorías. Dentro de cada grupo, organice los links según su importancia frente al usuario.

Los links basados en nombres de gente, deben conducir a las biografías cortas o a sus propias webs, no a un email. La utilidad de un link es que ayudará al usuario a

encontrar más información sobre el tema que esté buscando. Si usted desea incluir un link de email, muestre el email y no el nombre de la persona.

Hojas de estilos en la usabilidad

Las hojas de estilos en cascada (CSS) constituyen una de las grandes esperanzas para recuperar el ideal que tiene la web de separar la presentación del contenido.

Utilice una sola hoja de estilos para todas las páginas de su sitio, o unas pocas páginas css coordinadas si posee páginas con necesidades muy distintas: documentación técnica frente a páginas de marketing. Una de las principales ventajas de las hojas de estilos es la de asegurar la continuidad visual a medida que el usuario navega por el sitio.

Los sitios web deben tener la misma cohesión cuando todas las páginas del mismo se vinculen con la misma hoja de estilos. Existen dos formas implementar las hojas de estilos:

- Una hoja de estilos incrustada se incluye como parte de la página web en forma de líneas de código html adicionales.
- Una hoja de estilos vinculada se mantiene en un archivo separado, y cada página web que desee usar ese estilo posee un vínculo de hipertexto en su encabezado que señala o hace referencia a esa hoja de estilos.

El hacer referencia a un archivo externo css, le proporcionará la ventaja de actualizar sólo el contenido de esta página css que se propagará al resto de páginas las cuales hagan referencia a esta página css.

Sí utiliza una sola hoja de estilos para todo el sitio web, este archivo se descargará una sola vez, con lo que ahorraremos en tiempo de carga al pasar de una página a otra.

Tamaño de una página web

El tamaño (peso) de las páginas es crítico: la velocidad con la cual las páginas pueden ser descargadas y ser mostradas.

El tiempo de cuanto tardan las peticiones de un usuario en llegar a su pantalla ha sido el tema de muchas pruebas con usuarios en los últimos años. En general, el tiempo de carga debe ser menos de 10 segundos para tener la atención del usuario; si no, el usuario cancelará la sesión. Los estudios del peso de carga de la página también han probado que los webs que son más rápidos, consiguen más tráfico.

¿Cómo deben ser las páginas?

Asumimos que la mayoría de los usuarios tendrán acceso al web con un módem cable, fibra o ADSL. Esta tabla indica los tiempos y tamaños de las páginas para alcanzar ambos 1 segundos y de 10 segundos tiempos de descarga

Esto significa que los Web deben de ser de un tamaño menor de 150 KB donde sea posible. Además, cualquier archivo más grande de 250 KB debe ser separado hacia fuera y ser identificado al usuario como un archivo grande, preferiblemente con una indicación de aproximadamente como es de grande.

El tamaño total de la página incluye el tamaño de todos los gráficos utilizados, puesto que tendrán que ser descargados, también, a menos que se estén reutilizando (véase el uso de imágenes).

Uso de imágenes en la web

Uno de los factores que más dan que hablar es la inclusión de imágenes de todas las clases - las fotografías, diagramas, ilustraciones, multimedia etc. Los usuarios culpan a las imágenes de muchos los problemas de carga y de utilidad que pueblan la red. Las imágenes atraen y distraen a usuarios por igual, es importante limitar los gráficos. La reducción de imágenes es una herramienta para asegurarse de que su web están en un tamaño apropiado (véase el tamaño de la página).

Reutilización de imágenes. Cuando usted utiliza una imagen, considere el reutilizarla en otras páginas. Los navegadores de hoy pueden guardar imágenes, lo que significa que no tiene que ser descargada otra vez la próxima vez una página se cargue.

La reutilización también tiene la ventaja de dar al usuario un sentido de la familiaridad con su sitio. Dentro de un web, reutilice flechas, botones o iconos, intente que sus imágenes pesen lo menos posible, el usuario se lo agradecerá.

Animaciones en la web. Las animaciones son particularmente molestas y pueden conducir realmente a usuarios lejos de un web o de un sitio. No utilice la animación a menos que agregue verdad al significado de la información.

La animación puede ser una herramienta valiosa. El movimiento gratuito en las páginas del web, molesta a usuarios y no tiene cuenta el hecho de que muchos usuarios tienen viejas versiones de navegadores o no tienen los más nuevos sistemas de reproducción disponibles, (contrario a lo que creen algunos diseñadores, los usuarios no descargarán un plug-in nuevo, porque su página lo requiera).

El formato más eficiente para sus imágenes. Utilice siempre el formato más eficiente para sus imágenes. Para las fotografías, utilice el JPEG con tanta compresión como sea posible. Otros tipos de imágenes se deben almacenar usando formato del GIF. Usar un formato inadecuado puede doblar o triplicar el tamaño del archivo y del tiempo de la transferencia desde el servidor.

Reduzca la resolución de la imagen tanto como sea posible. La reducción en la resolución mejora la transferencia directa de la imagen. Una imagen debe ser muy de alta

de resolución, si se piensa para imprimir solamente (puesto que las impresoras tienen una resolución mucho más alta).

Reduzca el número de colores. En lo posible, reduzca el número de colores. Muchos programas gráficos le permiten "posterizar" las fotos, de tal modo reduciendo los requisitos del color. Si usted utiliza muchas imágenes, utilice una herramienta de compresión de imagen para reducir el tamaño de los archivos.

Incluya en la imágenes las cualidades en HTML que mejorarán el funcionamiento. Las cualidades explícitas del tamaño aceleran la exhibición del web, puesto que el navegador no tiene que realmente cargar la imagen para determinar cuánto espacio debe dejar para esta imagen. Utilice la ANCHURA y ALTURA para todas las imágenes. También agregue el ALT (descripción del gráfico) a cada imagen.

Los menús y el usuario

Muchos o casi todos los webs utilizan un menú, localizado típicamente en la parte superior de las páginas, este menú nos proporciona funcionalidad a la navegación. El menú puede incluir simplemente un número de links.

En general, los menús son la mejor manera de ayudar a usuarios a no perder de vista el contenido de nuestro web; puede también ser utilizado conjuntamente con otras ayudas de la navegación, tales como barras de navegación, para los sitios que son demasiado profundos o liosos.

Si los menús son necesarios, dispóngalos en la parte superior, ésta es la localización acostumbrada para los menús y lo que mejor reconoce el usuario.

Asegurese de que su menú funciona correctamente. Hay menús que aparecen y desaparecen espectacularmente mientras que los textos del menú no aparecen.

Títulos del menú

Los títulos del menú deben ser cortos, puesto que el menú se suele limitar en su anchura, haga los títulos lo más distintos como sean posibles, así no liara al usuario. Recuerde que el propósito de los menús es dar acceso a los usuarios al resto de páginas así como a su información y no todo lo contrario.

El menú se debe ajustar a un formato definido, usando letras mayúsculas y minúsculas según proceda, en el mismo menú no se debe utilizar un párrafo con una font de letra Arial y en el mismo menú pero en otro parrafo una font de letra Times. Utilice un separador (línea horizontal) para distinguir una categoría de otra. Esto mejorará las opciones para el usuario de encontrar la información requerida.

Intente no utilizar los menús de varios niveles (de cascada). Los estudios con usuarios demuestran que los menús en cascada frustran a los usuarios. Estos menús fuerzan al usuario a memorizar el contenido de los menús y de cada submenú, con lo que cada vez les pondremos mas difícil el encontrar nuestros productos en nuestro site.

II – Accesibilidad.

La **accesibilidad** es un derecho que posibilita a la persona a permanecer en lugar de forma autónoma y confortable. La accesibilidad web se refiere a la capacidad de acceso a la Web y a sus contenidos por todas las personas independientemente de la discapacidad (física, intelectual o técnica) que presenten o de las que se deriven del contexto de uso (tecnológicas o ambientales). Esta cualidad está íntimamente relacionada con la usabilidad.



La Organización Mundial de la Salud (OMS) recoje en sus informes un total de 600 millones de personas con discapacidad. El acceso de estas personas a la tecnología debe tenerse en cuenta en la construcción de una sociedad igualitaria.

Cuando los sitios web están diseñados pensando en la accesibilidad, todos los usuarios pueden acceder en condiciones de igualdad a los contenidos. Por ejemplo, cuando un sitio tiene un código HTML semánticamente correcto, se proporciona un texto equivalente alternativo a las imágenes y a los enlaces se les da un nombre significativo, esto permite a los usuarios ciegos utilizar lectores de pantalla o líneas Braille para acceder a los contenidos. Cuando los videos disponen de subtítulos, los usuarios con dificultades auditivas podrán entenderlos plenamente. Si los contenidos están escritos en un lenguaje sencillo e ilustrados con diagramas y animaciones, los usuarios con dislexia o problemas de aprendizaje están en mejores condiciones de entenderlos.

Si el tamaño del texto es lo suficientemente grande, los usuarios con problemas visuales puedan leerlo sin dificultad. De igual modo, el tamaño de los botones o las áreas activas adecuado puede facilitar su uso a los usuarios que no pueden controlar el ratón con precisión. Si se evitan las acciones que dependan de un dispositivo concreto (pulsar una tecla, hacer clic con el ratón) el usuario podrá escoger el dispositivo que más le convenga.

Limitaciones.

Las limitaciones en la accesibilidad de los sitios Web pueden ser:

- **Visuales:** En sus distintos grados, desde la baja visión a la ceguera total, además de problemas para distinguir colores (Daltonismo).
- **Motrices:** Dificultad o la imposibilidad de usar las manos, incluidos temblores, lentitud muscular, etc, debido a enfermedades como el Parkinson, distrofia muscular, parálisis cerebral, amputaciones...
- **Auditivas:** Sordera o deficiencias auditivas.
- **Cognitivas:** Dificultades de aprendizaje (dislexia, discalculia, etc) o discapacidades cognitivas que afecten a la memoria, la atención, las habilidades lógicas, etc.

Características de un sitio accesible.

- **Transformable:** La información y los servicios deben ser accesibles para todos y deben poder ser utilizados con todos los dispositivos de navegación.
- **Comprensible:** Contenidos claros y simples.
- **Navegable:** Mecanismos sencillos de navegación.

Pautas de accesibilidad Web.

El máximo organismo dentro de la jerarquía de Internet que se encarga de promover la accesibilidad es el World Wide Web Consortium (W3C), en especial su grupo de trabajo Web Accessibility Initiative (WAI). En 1999 el WAI publicó la versión 1.0 de sus pautas de accesibilidad Web. Con el paso del tiempo se han convertido en un referente internacionalmente aceptado. En diciembre del 2008 las WCAG 2.0 fueron aprobadas como recomendación oficial. Estas pautas se dividen en tres bloques:

- Pautas de Accesibilidad al Contenido en la Web (**WCAG**). Están dirigidas a los webmasters e indican cómo hacer que los contenidos del sitio Web sean accesibles.
- Pautas de Accesibilidad para Herramientas de Autor (**ATAG**). Están dirigidas a los desarrolladores del software que usan los webmasters, para que estos programas faciliten la creación de sitios accesibles.
- Pautas de Accesibilidad para Agentes de Usuario (**UAAG**). Están dirigidas a los desarrolladores de Agentes de usuario (navegadores y similares), para que estos programas faciliten a todos los usuarios el acceso a los sitios Web.

Beneficios.

Los principales beneficios⁴ que ofrece la accesibilidad web.

- **Aumenta el número de potenciales visitantes de la página web:** esta es una razón muy importante para una empresa que pretenda captar nuevos clientes. Cuando una página web es accesible no presenta barreras que dificulten su acceso, independientemente de las condiciones del usuario. Una página web que cumple los estándares es más probable que se visualice correctamente en cualquier dispositivo con cualquier navegador.
- **Disminuye los costes de desarrollo y mantenimiento:** aunque inicialmente aprender a hacer una página web accesible supone un coste (igual que supone un coste aprender a utilizar cualquier tecnología nueva), una vez se tienen los conocimientos, el coste de desarrollar y mantener una página web accesible es menor que frente a una no accesible, ya que una página web accesible es una página bien hecha, menos propensa a contener errores y más sencilla de actualizar.
- **Reduce el tiempo de carga de las páginas web y la carga del servidor web:** al separar el contenido de la información sobre la presentación de una página web mediante CSS se logra reducir el tamaño de las páginas web y, por tanto, se reduce el tiempo de carga de las páginas web.
- **Aumenta la usabilidad de la página web:** esto también implica indirectamente, que la página podrá ser visualizada desde cualquier navegador.
- **Demostramos que nos implicamos socialmente.**

Requisitos del nivel A de accesibilidad.

Los requisitos de accesibilidad que exige el nivel A son los siguientes:

- Proporcionar un texto alternativo para todas las imágenes, objetos y otros elementos no textuales (mediante los atributos alt y longdesc).
- Asegurar que toda la información que utilice el color como elemento informativo pueda ser entendida por las personas o dispositivos que no pueden distinguir los colores.
- Marcar claramente (mediante los atributos lang) las variaciones del idioma del texto o de los elementos textuales (<caption>) respecto del idioma principal de la página.
- El documento debe poder leerse completamente cuando no se utilicen hojas de estilos.
- La información equivalente para los contenidos dinámicos debe adaptarse a los cambios de los contenidos dinámicos.
- Ningún elemento debe parpadear en la pantalla.
- El contenido del sitio se debe escribir con un lenguaje sencillo y limpio.
- Si se utilizan mapas de imagen. Proporcionar un enlace textual por cada una de las regiones del mapa de imagen. Utilizar mapas de imagen en el cliente, en vez de mapas de imagen de servidor.
- Si se utilizan tablas. Utilizar cabeceras de fila y de columna. Si la tabla tiene varios niveles de cabeceras, utilizar las agrupaciones disponibles (<thead>, <tfoot>).
- Si se utilizan scripts. Asegurar que la página también se pueda utilizar cuando no se ejecutan los applets y los scripts. Si no es posible, proporcionar informaciones equivalentes o páginas alternativas que sean accesibles.
- Si se utilizan contenidos multimedia (audio y vídeo). Incluir una descripción textual del contenido multimedia.
- Para los contenidos basados en vídeo o animaciones, sincronizar las alternativas textuales con la presentación.
- Si no se pueden cumplir los anteriores requisitos, proporcionar una página alternativa con la mayor cantidad posible de contenidos y que cumpla con los requisitos anteriores.

La lista completa con todos los requisitos de los tres niveles de accesibilidad se puede consultar en <http://www.w3.org/TR/WCAG10/full-checklist.html>

Herramientas en español de comprobación de la accesibilidad.

- TAW: <http://www.tawdis.net/>
- HERA: <http://www.sidar.org/hera/>
- eXaminator: <http://examinator.ws/>
- INTAV: <http://www.inteco.es/checkAccessibility/Accesibilidad/>

III – Otras tecnologías.

Para la creación de una aplicación web completa, es necesario además de la parte del diseño, que hemos aprendido a lo largo del libro con HTML y CSS, otras tecnologías que nos ofrezcan posibilidades como una mayor interacción en local, la manipulación de datos o la aplicación de reglas de negocio.

Estas tecnologías las podemos dividir en dos grandes grupos dependiendo del lugar donde se desarrollen. Aquellas que se despliegan del lado del navegador web y las que se desarrollan en el lado del servidor. Aunque la línea que separa estos dos lados sea cada vez más fina, esta división es todavía válida.

Tecnologías locales o del lado del cliente.

Estas tecnologías buscan facilitar una mayor y mejor interactividad ofreciendo al usuario una experiencia más grata en el uso de la aplicación web. Destaca fundamentalmente el lenguaje Javascript.

JavaScript es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.

JavaScript se diseñó con una sintaxis similar al C, aunque adopta nombres y convenciones del lenguaje de programación Java. Sin embargo Java y JavaScript no están relacionados y tienen semánticas y propósitos diferentes.

Todos los navegadores modernos interpretan el código JavaScript integrado en las páginas web. Para interactuar con una página web se provee al lenguaje JavaScript de una implementación del Document Object Model (DOM).



Tecnologías del lado del servidor.

Estas tecnologías se utilizan para el manejo del negocio de la aplicación y el acceso a los datos. Entre ellas destacan los siguientes lenguajes según su popularidad:

Java es un lenguaje de programación de alto nivel orientado a objetos, desarrollado por James Gosling en 1995. El lenguaje en sí mismo toma mucha de su sintaxis de C, Cobol y Visual Basic, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria. La memoria es gestionada mediante un recolector de basura.

Las aplicaciones Java están típicamente compiladas en un bytecode, aunque la compilación en código máquina nativo también



es posible. En el tiempo de ejecución, el bytecode es normalmente interpretado o compilado a código nativo para la ejecución, aunque la ejecución directa por hardware del bytecode por un procesador Java también es posible.

La implementación original y de referencia del compilador, la máquina virtual y las bibliotecas de clases de Java fueron desarrollados por Sun Microsystems en 1995. Desde entonces, Sun ha controlado las especificaciones, el desarrollo y evolución del lenguaje a través del Java Community Process, si bien otros han desarrollado también implementaciones alternativas de estas tecnologías de Sun, algunas incluso bajo licencias de software libre.

Entre diciembre de 2006 y mayo de 2007, Sun Microsystems liberó la mayor parte de sus tecnologías Java bajo la licencia GNU GPL, de acuerdo con las especificaciones del Java Community Process, de tal forma que prácticamente todo el Java de Sun es ahora software libre aunque la biblioteca de clases de páginas web comprendidas en las librerías de objecction de objetos para ser compilados como aplicaciones comprimidas no estan totalmente acopladas de acuerdo con Sun que dice que se requiere un interprete para ejecutar los programas de Java.



Python es un Lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis muy limpia y que favorezca un código legible.

Python fue creado a finales de los ochenta2 por Guido van Rossum en el Centro para las Matemáticas y la Informática (CWI, Centrum Wiskunde & Informatica), en los Países Bajos, como un sucesor del lenguaje de programación ABC, capaz de manejar excepciones e interactuar con el sistema operativo Amoeba.

Se trata de un lenguaje de programación multiparadigma ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico, es fuertemente tipado y multiplataforma.

Es administrado por la Python Software Foundation. Posee una licencia de código abierto, denominada Python Software Foundation License,¹ que es compatible con la Licencia pública general de GNU a partir de la versión 2.1.1, e incompatible en ciertas versiones anteriores.

Ruby es un lenguaje de programación interpretado, reflexivo y orientado a objetos, creado por el programador japonés Yukihiro "Matz" Matsumoto, quien comenzó a trabajar en Ruby en 1993, y lo presentó públicamente en 1995. Combina una sintaxis inspirada en Python y Perl con características de programación orientada a objetos similares a Smalltalk. Comparte también funcionalidad con otros lenguajes de programación como Lisp, Lua, Dylan y CLU. Ruby es un lenguaje de programación interpretado en una sola pasada y su implementación oficial es distribuida bajo una licencia de software libre.



Bibliografía

A continuación se detalla la biografía utilizada para la redacción de la presente obra. El 100% del material de consulta utilizado ha sido sitios web especializados en PHP y en programación.

Libros y revistas.

- **The PHP Anthology.** Harry Fuecks: Seguramente, el mejor libro sobre PHP escrito nunca. Abarca casi todos los temas importantes a la hora de crear un sitio Web profesional. Todos los *scripts* están cuidadosamente pensados y orientados a objetos. El autor es uno de los propulsores de los patrones de diseño.
- **PHP5 and MySQL Bible.** Tim Converse y Joyce Park: Completo libro que abarca numerosos temas. Especializado en generar bases de datos con MySQL y PHP 5.
- **PHP5 for Dummies.** Janet Valade. Un buen libro para aprender y profundizar en varias áreas. Desde luego el término *Dummies* no le hace justicia, porque a veces se hace algo complicado de seguir.
- **PHP 5 Power Programming.** Andi Gutmans. De uno de los creadores de PHP. Muy bien enfocado y con múltiples temas.
- **Beginning PHP 5.** Equipo Wrox. Libro extenso sobre PHP 5. Cubre todo lo que se puede conocer sobre PHP 5. En Octubre saldrá la segunda parte, llamada Professional PHP 5.
- **PHP Solutions:** Revista polaca traducida al español. Contiene muy buenos artículos sobre desarrollo actual con PHP.
- **PHP Magazine.** Revista alemana de gran alcance.
- **PHP Architect.** Revista canadiense, donde habitualmente escriben desabolladores como Harry Fuecks, Marco Tabini o Andi Gutmans. La misma editora de esta revista ha sacado el libro de estudio para presentarse al examen de Certificación de PHP.

Páginas web.

Se ha intentado en todo momento utilizar sitios web oficiales o con una gran presencia en Internet. Si en cualquier momento alguno de los anteriores enlaces fallase, se ruega al lector que se ponga en contacto con el editor para solventar el problema.

- Php.net – (<http://www.php.net>) Web oficial de php.
- Zend – (<http://www zend.com>) La empresa que hay detrás de php. Ofrece cursos y seminarios *on-line*.
- Phpbuilder – (<http://www.phpbuilder.com>) Comunidad de usuarios con código libre.
- Phppatterns – (<http://www.phppatterns.com>) La Web de Harry Fuecks.
- Codewalkers – (<http://www.codewalkers.com>) Web con mucho código libre.
- Hot Scripts PHP – (<http://www.hotscripts.com/category/scripts/php/scripts-programs/>) El más completo directorio de scripts PHP.
- PHP Scripts – (<http://www.scripts.com/php-scripts/>) Directorio de scripts de php.
- PHP Resource Index – (<http://php.resourceindex.com/>) Interesante directorio de scripts.
- GScriptS.net – (<http://gscripts.net/>) Directorio de scripts php gratuitos.
- PHP Resource Index – (<http://php.resourceindex.com/>) Interesante directorio de php scripts.
- Hot php scripts – (<http://www.hot-php-scripts.com/>) Otro directorio de scripts php.
- PHP JUnk Yard – (<http://www.phpjunkyard.com/>) Otro directorio de scripts php.
- Best php scripts – (<http://best-php-scripts.com/>) Extenso directorio de scripts php.
- Phpsolmag – (<http://www.phpmag.net>) Web de la revista PHP Magazine.
- Phparch – (<http://www.phparch.com>) Web de la revista PHP Architect.
- Sinuh – (<http://www.sinuh.org>) Comunidad GnuLinux de Extremadura.