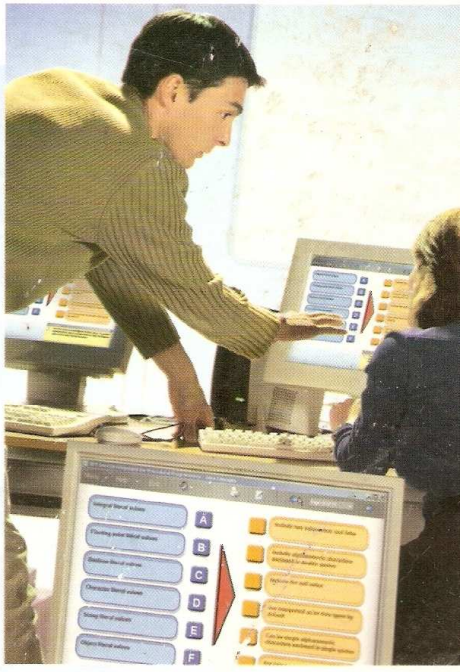
IBM

IBM Capacitación

Programación Web Avanzada

Código del Curso: CY720
Versión: 4.0

Libro 1: PHP Hypertext Preprocessor



Programación Web Avanzada

Código del Curso: CY720
Versión: 4.0

Guía del Estudiante

Libro 1: PHP Hypertext Preprocessor

IBM IT Education Services
Worldwide Certified Material

Información Sobre la Publicación

Esta publicación ha sido producida usando Microsoft Word 2000 y Microsoft PowerPoint 2000 para Windows y software Open Source como Apache 2.0.5.0 para Linux y PHP 5.0.

Marcas Registradas

Trademarks of International Business Machines Corporation

MySQL	Mozilla Firefox
Linux	Apache

Marcas Registradas de otras Compañías

Windows, Microsoft Visual Studio	Microsoft Corporation
Sybase	**Sybase Inc.**

Edición Febrero 2008

Instrucciones Especiales para la Impresión de este Curso:

No elimine páginas en blanco que puedan aparecer al final de cada unidad ó entre unidades. Estas páginas fueron insertadas intencionalmente.

Contenido

Descripción del Curso **5**
Descripción de Unidades ***7***
Volumen 1: Hypertext Preprocessor (PHP) **9**
Unidad 1: Fundamentos de PHP **11**
1. ¿Qué es PHP? 12
2. Características de PHP 15
3. Funcionamiento de PHP 17
4. Arquitectura de PHP 19
5. ¿Cómo Utilizar PHP? 20
Resumen 29
Unidad 1: Examen de Autoevaluación 30
Unidad 1: Respuestas al Examen de Autoevaluación 32
Unidad 2: Elementos del Lenguaje PHP **33**
1. Introducción 34
2. Sintaxis Básica de PHP 34
3. Variables 39
4. Constantes 48
5. Tipos de Datos 50
6. Operadores 57
7. Estructuras de Control 60
Resumen 72
Unidad 2: Examen de Autoevaluación 73
Unidad 2: Respuestas al Examen de Autoevaluación 76
Unidad 3: Funciones y Extensiones PHP **77**
1. Introducción 78
2. Funciones Definidas por el Usuario 78
3. Funciones Definidas por el Lenguaje 82
4. Extensiones en PHP 101
Resumen 103
Unidad 3: Examen de Autoevaluación 104
Unidad 3: Respuestas a Examen de Autoevaluación 106
Unidad 4: Lab. de Elementos del Lenguaje y Funciones **107**
Ejercicios de Laboratorio 108

Unidad 5: Desarrollo de Sitios Web 109
1. Introducción 110
2. Aplicaciones Web 110
3. Variables Predefinidas en PHP 111
4. Uso de Formularios en PHP 115
5. Administración de Sesiones 122
Resumen 142
Unidad 5: Examen de Autoevaluación 143
Unidad 5: Respuestas a Examen de Autoevaluación 146
Unidad 6: Lab. Desarrollo de Sitios Web 147
Ejercicios de Laboratorio 148
Unidad 7: Acceso a Base de Datos usando PHP 149
1. Introducción 150
2. Acceso a Bases de Datos 150
3. Acceso a Bases de Datos con la Extensión MySQL 151
4. Ejecutar Sentencias SQL Usando Funciones MySQL 153
5. Recuperar Registros Usando Funciones MySQL 156
6. Otras Funciones Útiles de la Extensión MySQL 159
Resumen 164
Unidad 7: Examen de Autoevaluación 165
Unidad 7: Respuestas a Examen de Autoevaluación 167
Unidad 8: Lab. Acceso a Bases de Datos 169
Ejercicios de Laboratorio 170
Unidad 9: Introducción a la Programación Orientada a Objetos 171
1. Introducción 172
2. Definir Clases y Objetos 172
3. Herencia entre Clases 182
4. Polimorfismo 186
5. Atributos y Métodos Estáticos 187
6. Clases Abstractas 188
7. Interfaces 191
8. Clonación de Objetos 193
9. Uso de la Función `include()` 195
Resumen 210
Unidad 9: Examen de Autoevaluación 211

Unidad 9: Respuestas al Examen de Autoevaluación 214
Unidad 10: Lab. de Programación Orientada a Objetos 215
Ejercicio de Laboratorio 216
Unidad 11: Extensiones PHP..... 219
1. Introducción 220
2. ¿Qué es XML? 220
3. Extensión de PHP para Trabajar con XML 222
4. PHPLib 234
Resumen 236
Unidad 11: Examen de Autoevaluación 237
Unidad 11: Respuestas a Examen de Autoevaluación 239

Descripción del Curso

Nombre del Curso

Programación Web Avanzada.

Duración

La duración del curso es de 40 horas.

Propósito

Este curso ofrece a los estudiantes una visión general de PHP - Hypertext Preprocessor. Los estudiantes aprenderán los Fundamentos, Elementos, Funciones y Extensiones del Lenguaje PHP. También aprenderán sobre el Desarrollo de sitios Web, Acceso a Base de Datos usando PHP, Programación Orientada a Objetos y Extensión de PHP para trabajar con XML.

El curso incorpora varias sesiones de laboratorios no evaluados, donde los estudiantes tendrán la posibilidad de ganar experiencia en la Programación Web.

Audiencia

Estudiantes, profesionales y gente de negocios.

Prerrequisitos

- Entender los conceptos básicos acerca de HTML.
- Capacidad de crear páginas HTML.
- Entender los conceptos básicos sobre lenguajes script.
- Experiencia práctica trabajando con HTML.

Estas habilidades pueden obtenerse al completar el curso CY710: Programación Web Básica.

Objetivos del Curso

Al final de este curso Ud. será capaz de:

- Discutir qué es PHP, sus características, funcionamiento y arquitectura.
- Describir variables, constantes, tipos de datos y explicar estructuras de control.
- Discutir acerca de subrutina y funciones.
- Definir cookies y explicar cómo administrar las cookies usando PHP.
- Discutir la necesidad de PHP en el desarrollo de aplicaciones web.
- Explicar cómo crear script PHP.
- Explicar cómo trabajar con la Programación Orientada a Objetos en PHP.
- Establecer cómo PHP puede usarse para la administración de sesiones.
- Describir la extensión de PHP para trabajar con XML.
- Establecer Acceso a Base de Datos usando PHP.

Descripción de Unidades

Volumen 1 : Hypertext Preprocessor

Unidad 1: Fundamentos de PHP

En esta unidad, se describe al lenguaje PHP, sus características y evolución. Se discuten los beneficios y funcionalidades de PHP. Se explica cómo funciona un script hecho en PHP. Adicionalmente, se describe la arquitectura de PHP y se crea un script sencillo en este lenguaje.

Unidad 2: Elementos del Lenguaje PHP

En esta unidad, se explica la sintaxis básica de PHP. Se describe el uso de variables y constantes, además del uso de los operadores disponibles en PHP. Se indican los diferentes tipos de datos que posee PHP. Finalmente, se discuten las estructuras de control de PHP.

Unidad 3: Funciones y Extensiones PHP

En esta unidad, se explican y desarrollan las funciones definidas por el usuario. Se describen y utilizan las funciones del lenguaje incorporadas en PHP. Aunado a esto, se indica cómo se incorporan las extensiones en PHP.

Unidad 4: Lab. de Elementos del Lenguaje y Funciones

Esta unidad está diseñada para obtener práctica, escribiendo scripts básicos usando los elementos del lenguaje PHP y creando funciones definidas por el usuario e incorporadas del lenguaje.

Unidad 5: Desarrollo de Sitios Web

En esta unidad, se define el término Aplicación Web. Se discute la utilidad de las variables predefinidas de PHP. Se explica cómo se procesan los formularios en PHP. Se explica la administración de sesiones en PHP.

Unidad 6: Lab. Desarrollo de Sitios Web

Laboratorio que proporciona algunos ejercicios para ayudar a Crear una aplicación web sencilla en PHP. Obtener y manipular los campos de un formulario como entrada de datos. Crear y utilizar cookies. Administrar sesiones en PHP.

Unidad 7: Acceso a Base de Datos usando PHP

En esta unidad, se describe cómo se trabaja con bases de datos en PHP. Se explica cómo conectarse y desconectarse a una base de datos, además de cómo recuperar,

insertar, eliminar y actualizar registros. Se utiliza la extensión de PHP para acceder a bases de datos en MySQL.

Unidad 8: Lab. Acceso a Bases de Datos

Se proporcionan un conjunto de ejercicios para ayudar a los estudiantes a trabajar con los conceptos aprendidos en la unidad anterior, los cuales servirán para conectarse a bases de datos en MySQL a través de PHP. El propósito principal es implementar una aplicación web que pueda agregar, modificar y eliminar datos de una base de datos en MySQL.

Unidad 9: Introducción a la Programación Orientada a Objetos

En esta unidad, se crean Clases y Objetos en PHP, se describe la Herencia en PHP, además de dar a conocer cómo se implementa el Polimorfismo en este lenguaje. Se realiza la creación de Clases Abstractas e Interfaces. Se discute cómo se realiza la clonación de objetos.

Unidad 10: Lab. de Programación Orientada a Objetos

Este laboratorio proporciona algunos ejercicios para ayudar a escribir un script PHP utilizando la Programación Orientada a Objetos. Adicionalmente, permite crear una clase que se encargue de realizar las tareas de Bases de Datos, además de crear clases que sirvan de contenedoras de datos obtenidos de una BD.

Unidad 11: Extensiones PHP

En esta unidad, se conoce en qué consiste XML. Se utiliza la extensión DOM de PHP para trabajar con documentos XML. Entre otros aspectos, se crean scripts PHP que puedan manipular documentos XML. Finalmente se conoce en qué consiste la librería estándar de PHP.

Volumen 1: Hypertext Preprocessor (PHP)

Unidad 1: Fundamentos de PHP

Objetivos de Aprendizaje

Al final de esta unidad, usted será capaz de:

- Describir PHP, sus características y su evolución.
- Discutir los beneficios y funcionalidades de PHP.
- Explicar cómo funciona un script hecho en PHP.
- Describir la arquitectura de PHP.
- Crear un script sencillo en PHP.

1. ¿Qué es PHP?

PHP, acrónimo de "PHP: Hypertext Preprocessor", es un lenguaje interpretado de alto nivel que se ejecuta del lado del servidor. PHP está especialmente diseñado para desarrollos web y puede ser embebido en páginas HTML. La mayor parte de su sintaxis es similar a C y Perl. La meta de este lenguaje es permitir a los desarrolladores web crear páginas dinámicas de una manera rápida y sencilla. PHP es "Open Source" (Código Abierto).

1.1 Origen y Evolución de PHP

En 1995 Rasmus Lerdorf, para controlar los accesos de los visitantes a su currículum online, creó un conjunto de scripts en Perl, al cual llamó 'Personal Home Page Tools'. Enseguida amigos y otros usuarios solicitaron su uso, de modo que en poco tiempo le solicitaron más beneficios. Según se requería más funcionalidad, Rasmus Lerdorf fue escribiendo una implementación en C mucho mayor, que era capaz de comunicarse con bases de datos y permitía a los usuarios desarrollar sencillas aplicaciones web dinámicas.

El autor decidió asociarse a otras personas y creo un nuevo lenguaje llamado PHP/FI (Personal Home Pages / Form Interpreter, traducido como 'Páginas Personales / Intérprete de Formularios'). Lerdorf decidió liberar el código fuente de PHP/FI para que cualquiera pudiese utilizarlo, reparar errores y mejorar el código.

PHP/FI incluía algunas de las funcionalidades básicas de PHP tal y como se conoce hoy. Tenía variables como las de Perl, interpretación automática de variables de formulario y sintaxis embebida HTML.

En 1997, PHP/FI 2.0, la segunda versión de la implementación en C, tuvo un seguimiento estimado de varios miles de usuarios en todo el mundo, con aproximadamente 50.000 dominios que informaron que lo tenían instalado, sumando alrededor del 1% de los dominios de Internet. PHP/FI 2.0 se liberó oficialmente en Noviembre de 1997.

1.1.1 PHP 3

PHP 3.0 fue creado por Andi Gutmans y Zeev Suraski en 1997, quienes lo rescribieron completamente, pues PHP/FI 2.0 tenía pocas posibilidades para desarrollar una aplicación comercial que estaban realizando como parte de un proyecto universitario.

Lerdorf, Gutmans y Zuraski decidieron cooperar y anunciar PHP 3.0 como el sucesor oficial de PHP/FI 2.0, interrumpiendo en su mayor parte el desarrollo que se llevaba a cabo de PHP/FI 2.0.

El equipo de desarrollo decidió que el producto no debía mantener el nombre de "Personal Home Page" y lo rebautizaron como *'PHP: Hypertext Preprocessor'* (Preprocesador de Hipertexto PHP), siendo éste un acrónimo recurrente.

Una de las características de PHP 3.0 era su extensibilidad, además de proveer a los usuarios finales una infraestructura de soporte para muchas bases de datos, protocolos y APIs. Las características de extensibilidad de PHP 3.0 atrajeron a muchos programadores a colaborar con nuevos módulos de extensión. Otra característica introducida en PHP 3.0 fue el soporte de la sintaxis Orientada a Objetos.

PHP 3.0 se liberó oficialmente en Junio de 1998. A finales de 1998, PHP 3.0 estaba instalado en aproximadamente un 10% de los servidores web en Internet.

1.1.2 PHP 4

En 1998, poco después del lanzamiento de PHP 3.0, Andi Gutmans y Zeev Suraski comenzaron a trabajar en el nuevo desarrollo del núcleo de PHP para mejorar la ejecución de aplicaciones complejas y mejorar la modularidad del código base de PHP, puesto que PHP 3.0 no estaba diseñado para el mantenimiento eficiente de aplicaciones complejas.

PHP 4.0 se basó en un nuevo motor, llamado *'Motor Zend'* (combinación de sus nombres, Zeev y Andi) o '*Zend Engine*' y se introdujo por primera vez a mediados de 1999. Actualmente el motor Zend es propiedad de Zend Technologies LTD.

Zend se refiere al motor del lenguaje. El término PHP se refiere al lenguaje como tal. Juntos forman el *Sistema PHP* completo. Zend provee la infraestructura y los servicios a los módulos de PHP e implementa la sintaxis del lenguaje.

El motor Zend es un intérprete que analiza el código de entrada, lo traduce y lo ejecuta. Además proporciona algunas funciones básicas. PHP implementa la mayor parte de la funcionalidad del lenguaje y actúa como interfaz que se comunica con el servidor web.

PHP 4.0 fue oficialmente liberado en Mayo de 2000. Además de la mejora en el desempeño, PHP 4.0 incluía otras características adicionales como el soporte para la mayoría de los servidores web, sesiones HTTP, buffers de salida, formas más seguras de controlar las entradas de usuario y muchas nuevas construcciones de lenguaje.

1.1.3 PHP 5

PHP 5 fue liberado en Julio de 2004. Es principalmente manejado por su núcleo, el motor Zend 2.0. Posee un nuevo modelo de objetos y muchas otras características adicionales.

El manejo de objetos en PHP ha sido completamente restructurado, permitiendo un mejor desempeño. En las versiones anteriores de PHP, los objetos eran manejados como tipos de datos primitivos. En el nuevo enfoque, los objetos pueden ser accedidos por referencia (a diferencia del antiguo modelo donde un objeto se referenciaba por valor cuando se pasaba como parámetro a un método).

Las nuevas características Orientadas a Objetos incluyen:

- Nueva sintaxis para definir el constructor y destructor de una clase.
- Métodos finales (métodos que no pueden ser sobrescritos).
- Clases finales (clases de las cuales ninguna otra clase puede heredar).
- Modificadores de acceso (público, protegido, privado) para atributos y métodos.
- Clases y métodos abstractos.
- Clases y métodos estáticos.
- Interfaces.
- Clonación explícita de objetos, entre otros.

Otras características incluyen:

- Nueva administración de memoria que permite trabajar eficientemente en entornos multihilos.
- Manejo mejorado de excepciones.
- Extensiones para Perl, lo que permite ejecutar scripts Perl, usar su modelo de objetos y funcionalidades nativas del lenguaje.
- Soporte para XML, Web Services y SOAP, entre otros.

1.1.4 ZEND 2.0

A continuación se presentan las nuevas características del motor Zend 2.0, el núcleo de PHP 5, sobre su predecesor Zend 1.0.

- Nuevo modelo de objetos, muy similar a Java.
- Manejo de Excepciones. Implementa sentencias try, catch y throw. Además del lanzamiento de excepciones a nivel de funciones de usuario, se puede configurar el motor de manera que las funciones internas lancen excepciones, en vez de retornar códigos de error.
- Interfaz mejorada con las APIs Orientadas a Objetos de terceros. Mejora capacidad de comunicarse con APIs como la de Java y los componentes COM, lo cual permite una mejor integración con otros modelos de objetos.
- Soporte a la Internacionalización. Mejora el soporte a conjuntos caracteres 'no-Western' (no occidentales) y UNICODE.

De acuerdo a NetCraft (*http://www.netcraft.com*), PHP estaba implantado en más de 1 millón de hosts en Noviembre de 1999, y ya para Octubre del 2003 estaba instalado en al menos 14 millones de hosts.

Hoy, se estima que PHP es usado por cientos de miles de programadores y millones de sitios informan que lo tienen instalado, sumando más del 20% de los dominios en Internet.

Para más información acerca de nuevos proyectos, desarrollos y actualizaciones de PHP y el motor Zend, visite sus sitios oficiales: *www.php.net* y *www.zend.com*.

2. Características de PHP

PHP es un lenguaje de script del lado del servidor, embebido en HTML (server-side HTML – embedded scripting language) y es de plataforma cruzada (cross plataform). Además es un lenguaje de código abierto (open source). Las características principales de PHP se explican a continuación:

- Al ser PHP un *lenguaje scripting del lado del servidor*, se distingue de los lenguajes de script del lado del cliente, tales como JavaScript, porque el código es ejecutado en el servidor y sólo la respuesta se envía al cliente. Esto significa que cuando el resultado llega al cliente no se puede determinar el código subyacente que lo generó.
- El término *embebido en HTML*, quiere decir que se puede incorporar código PHP directamente dentro de código HTML. El código PHP puede ser incrustado dentro de código HTML utilizando etiquetas especiales de inicio y fin (**<?php ... ?>** ó **<?...?>**).
- El hecho que trabaje en *plataformas cruzadas*, se refiere a que se puede utilizar en plataformas bajo distintos sistemas operativos (UNIX, LINUX, Windows (98,NT,2000 y XP), OS/2, Mac Os, etc) y servidores web (Apache, Microsoft Internet Information Server, Netscape, Oreilly Website Pro server, Xitami, OmniHTTPd y otros). Además el código PHP puede trasladarse desde un sistema a otro con muy pocos o ningún cambio.
- El hecho que PHP sea un *lenguaje de código abierto*, significa que su distribución es libre (se puede obtener sin pagar una licencia de software) y además su código fuente es accesible para todos los interesados. Muchas personas piensan que '*open source*', simplemente significa '*gratis*', pero ese es sólo uno de sus beneficios.

Los proyectos Open Source, como es el caso de PHP, cuentan con una gran comunidad organizada de desarrolladores, que ofrece soporte técnico a los usuarios, descubre y repara errores, actualiza continuamente el código con extensiones que expanden las capacidades de PHP.

Al tener acceso al código fuente, los desarrolladores de PHP pueden incorporar nuevas funcionalidades, cooperar con nuevas extensiones, reportar 'bugs' y proponer su corrección.

Actualmente, un hecho que ha despertado suspicacias dentro de la Comunidad de Software Libre, es que el motor Zend, actual intérprete de PHP, posee un optimizador opcional de carácter comercial, el '*Zend Accelerator*'. Este optimizador almacena código compilado en memoria, eliminando la sobrecarga de analizar e interpretar código fuente por cada solicitud.

2.1 Principales Funcionalidades y Beneficios de PHP

- Posibilidad de usar programación procedimental o programación orientada a objetos. Aunque no todas las características estándares de la programación

orientada a objetos están implementadas en PHP, la versión liberada más recientemente, provee un Modelo de Objetos bastante completo.

- Soporte para una gran cantidad de bases de datos. Las siguientes son algunas de las bases de datos soportadas por PHP: PostgreSQL, MySQL, IBM DB2, Oracle (OCI7 y OCI8), Sybase, Direct MS-SQL, InterBase, Informix, Unix dbm, entre otras.
- Soporte para ODBC (Estándar Abierto de Conexión con Bases de Datos), lo que permite comunicarse con cualquier base de datos que soporte ese estándar.
- Soporte para comunicarse con otros servicios usando protocolos tales como: LDAP, IMAP, SNMP, NNTP, POP3, HTTP, COM (en Windows) y muchos otros.
- PHP puede utilizar objetos Java de forma transparente como objetos PHP.
- La extensión de CORBA puede ser utilizada para acceder a objetos remotos.
- Posee características muy útiles para el procesamiento de texto, desde expresiones regulares extendidas tipo Perl hasta procesadores de documentos XML.
- Soporta los estándares SAX y DOM para procesar y acceder a documentos XML. También se puede utilizar la extensión XSLT para transformar documentos XML.

2.2 ¿Por qué utilizar PHP?

Cuando deseamos que un sitio web posea dinamismo, es necesario que se puedan realizar operaciones para acceder a bases de datos, archivos, manipular correo electrónico, mostrar en el navegador del cliente cambios de información (datos, imágenes, sonido) de acuerdo a sus peticiones.

En el mercado existen tecnologías como los scripts CGI (hechos en C, Perl o Phyton), ASP y JSP, que permiten incorporar dinamismo a sitios web, pero la curva de aprendizaje de Perl o C para trabajar con CGI, VBscript para trabajar con ASP o Java para JSP, es mayor que la necesaria para aprender PHP.

Otra razón es que PHP fue especialmente diseñado para trabajar en el entorno web, esto permite que funcione en forma más rápida y eficiente con HTML.

Algunos lenguajes utilizados para el desarrollo en la Web no son ideales para esta tarea, pues inicialmente no fueron concebidos para ese propósito específico, sino que fueron adaptados posteriormente para trabajar en la Web (tal es el caso de los lenguajes que se usan en el scripting CGI, como C y Perl).

PHP fue pensado desde sus inicios como un lenguaje que se pudiera incrustar directamente en HTML para desarrollar sitios web dinámicos y aplicaciones basadas en la Web. Por lo tanto, ofrece un mejor tiempo de respuesta, mayor seguridad y transparencia para el usuario final en entornos web.

Una gran ventaja de PHP es que no está vinculado a ningún sistema operativo ni servidor web. Aunque fue diseñado inicialmente para funcionar bajo Unix y bajo el

servidor Apache, también funciona bajo Windows en el Microsoft IIS, Netscape Server y otros. Además de proveer todos los beneficios listados en la sección 2.1 de la presente unidad.

Muchas son las tecnologías utilizadas para lograr el procesamiento del lado del servidor (ASP, JSP, CGI, PHP) proporcionando dinamismo a sitios web y fortaleciendo la plataforma para el Comercio Electrónico. PHP actualmente está ganando terreno en estas áreas.

3. Funcionamiento de PHP

PHP se diferencia de los scripts CGI, hechos en Perl o C, porque estos escriben el código HTML a través de comandos, mientras que PHP se puede incorporar directamente dentro de código HTML utilizando etiquetas especiales de inicio y fin (**<?php** ... **?>** ó **<?** ... **?>**).

A continuación se muestra un script que se procesa del lado del servidor que está hecho en PHP.

Ejemplo de un script PHP:

```
1.  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
2.  <HTML>
3.  <HEAD><TITLE>Ejemplo sencillo de PHP</TITLE></HEAD>
4.  <BODY>
5.  <?php
6.      echo "<H1 align="center">Hola a todos!</H1>";
7.  ?>
8.  </BODY>
9.  </HTML>
```

Como se puede observar, es posible combinar el código en PHP con etiquetas HTML dentro de la misma página.

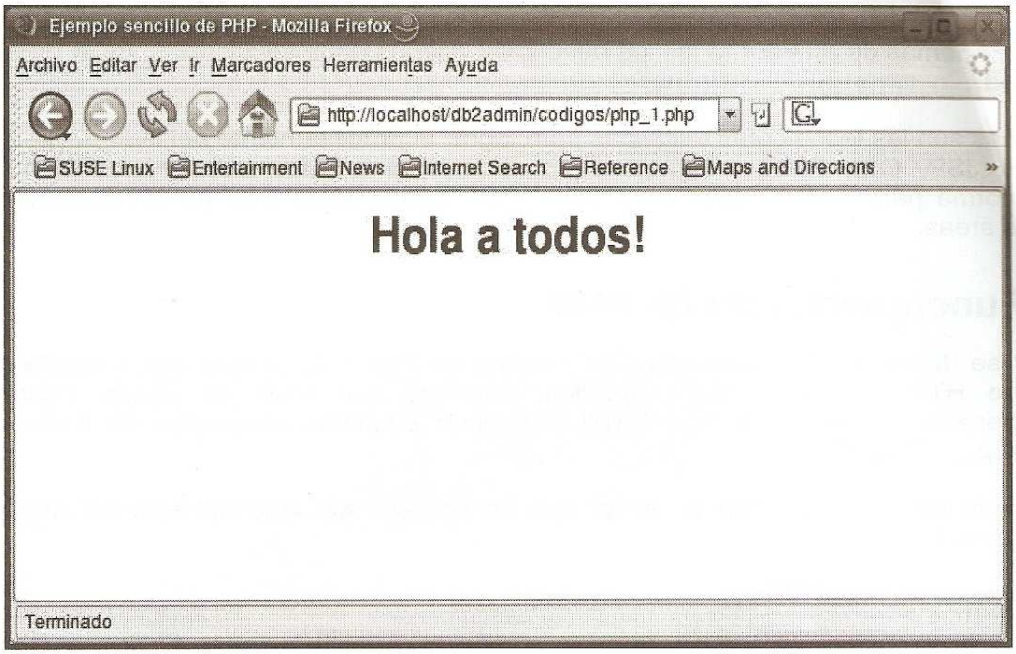


Figura 1.1: Ejemplo Sencillo de PHP

Existen tres formas de utilizar los scripts PHP:

- **Scripts del lado del servidor**: Representa el área principal de desarrollo y uso de PHP. Para ejecutar scripts del lado del servidor se necesita el intérprete PHP (CGI o módulo), un servidor web y un navegador. Es necesario que el servidor web se esté ejecutando con PHP instalado. El resultado del programa PHP se puede obtener a través del navegador, que se comunica con el servidor web.

- **Scripts en la línea de comandos**: Puede crear un script PHP y ejecutarlo sin necesidad de tener un servidor web ni un navegador. Solamente necesita el intérprete PHP.

- **Aplicaciones de interfaz gráfica**: Probablemente, PHP no sea el lenguaje más apropiado para escribir aplicaciones gráficas. Sin embargo, se pueden utilizar algunas características avanzadas en programas clientes, a través de PHP – GTK.

  PHP – GTK es la solución PHP para escribir aplicaciones GUI del lado del cliente, creado por Andrei Zmievski. PHP – GTK está disponible como una extensión de PHP, no incorporada en la distribución principal.

4. Arquitectura de PHP

La Arquitectura de PHP, presentada en la Figura 1.2, se explica brevemente a continuación:

- **El Motor Zend** (Zend Engine) es un componente autocontenido que funciona como el parser del lenguaje. Es un intérprete que analiza el código de entrada, lo traduce y lo ejecuta. Además proporciona algunas funciones básicas del lenguaje.
- **El núcleo PHP** implementa la mayor parte de las funciones del lenguaje.
- **La capa SAPI** o Server Application Programming Interface, (traducido como Interfaz de Programación de Aplicaciones del Servidor), son módulos que proveen una interfaz para interactuar de forma transparente con distintos servidores web y otros servidores (por ejemplo, un servidor de servlets de Java).

  Esta capa provee una abstracción del servidor web y simplifica la tarea de agregar soporte nativo para nuevos servidores. La capa SAPI logra que el funcionamiento de PHP sea independiente del servidor web que se esté utilizando. SAPI incrementa la estabilidad de PHP y además soporta servidores web multihilos. PHP actualmente posee implementaciones SAPI para Apache, Microsoft IIS, Netscape e iPlanet, AOLServer, CGI, Java, entre otros.
- **Las extensiones PHP** son módulos de funciones autocontenidos. Muchas de las funciones, por ejemplo el soporte a MySQL, son provistas por una extensión. Las extensiones pueden ser enlazadas a PHP en tiempo de compilación o pueden cargarse dinámicamente según sea requerido. Muchas extensiones son opcionales.

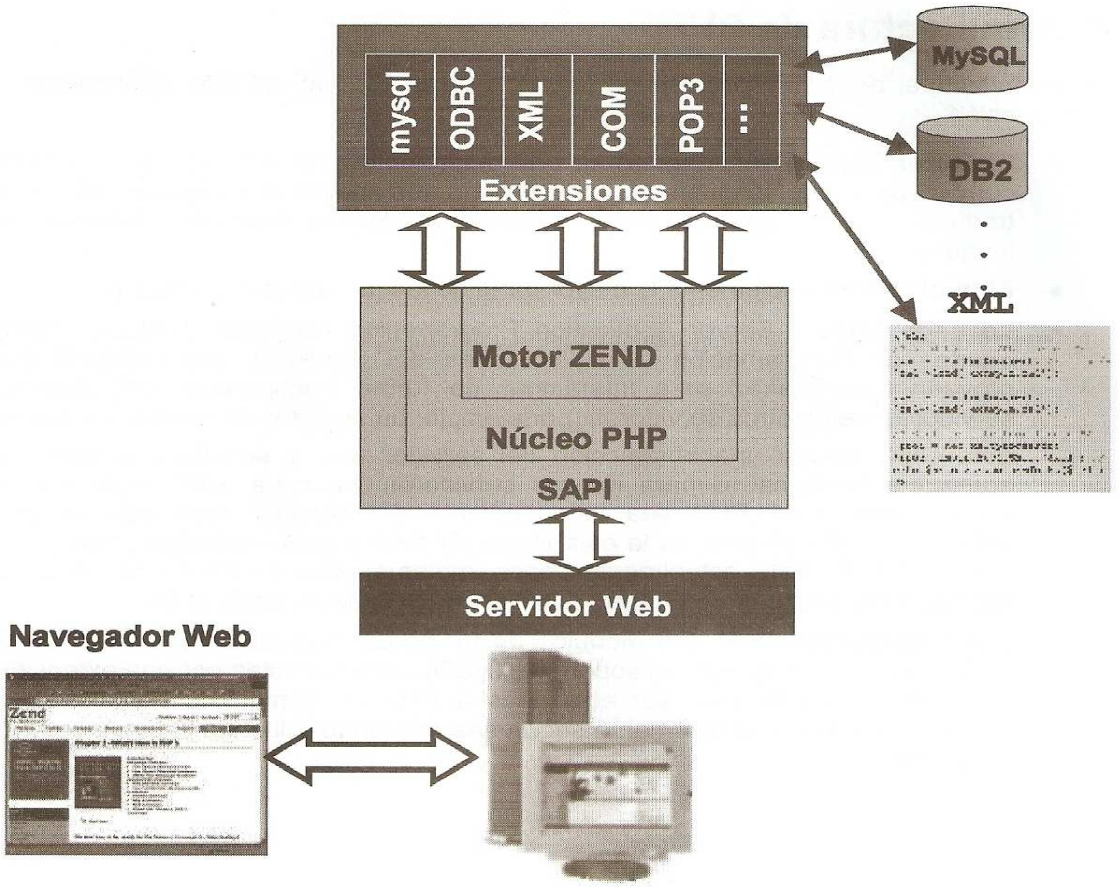


Figura 1.2: Arquitectura de PHP

Una vez establecidos los conceptos, características, funcionamiento y la arquitectura de PHP, se presentan los elementos necesarios para poder crear y ejecutar scripts PHP.

5. ¿Cómo Utilizar PHP?

Para ejecutar archivos php como scripts del lado del servidor, se debe contar con un servidor web que soporte PHP. Además todos los archivos con la extensión **.php** deben ser procesados por el intérprete de PHP.

El servidor web que se utilizará para propósitos de este curso es el Apache HTTP Server versión 2.0.5. El servidor web Apache es un proyecto Open Source (código abierto) y es uno de los servidores más utilizados en Internet. Para mayor información sobre Apache visite su sitio web oficial: http://www.apache.org. Allí también se puede descargar el instalador de Apache.

5.1 Configuración de PHP

PHP se puede configurar para ejecutar scripts del lado del servidor, de dos formas:

- Módulo SAPI.
- Modo CGI.

PHP posee módulos directo de interface o SAPI (Server Application Programming Interface) para muchos servidores web, tales como Apache, Microsoft IIS, Netscape e iPlanet. Si PHP no soporta un módulo SAPI para algún servidor web, siempre se puede utilizar como CGI. Esto significa que el servidor web se debe configurar para que utilice el ejecutable CGI de PHP en el procesamiento de los archivos `.php`.

El uso de PHP como un ejecutable CGI es una opción cuando por alguna razón no se pueda integrar PHP como módulo SAPI del servidor web. Pero se debe ser cuidadoso al utilizar este tipo de instalación. No es recomendable instalar PHP en modo CGI para servidores que estén en producción porque la seguridad puede quedar comprometida. Es preferible utilizar PHP como módulo SAPI.

5.2 Instalación de PHP, bajo Windows

Existen dos formas para instalar PHP bajo Windows:

- A través de un asistente de instalación.
- Manualmente.

Los archivos instalables para ambas formas de instalación, están disponibles en la sección de downloads del sitio oficial de PHP, *http://www.php.net/downloads.php*.

Con el asistente se instala la versión CGI de PHP, para IIS PWS, Xitami, Apache, etc. y configura al servidor web para que lo soporte. En este tipo de instalación no se incluye ninguna de las extensiones externas de PHP

El asistente de instalación para Windows es una forma sencilla de instalar PHP, pero no es la más recomendable porque no soporta la configuración automática de extensiones. Tampoco es recomendable porque instala la versión CGI de PHP, la cual debe ser configurada cuidadosamente para que no comprometa la seguridad del servidor.

En la instalación manual de PHP el usuario lo instala directamente, paso a paso, es decir, debe copiar los recursos necesarios de PHP en el computador, editar archivos de configuración (por ejemplo, el `php.ini`), copiar algunos archivos en ciertas ubicaciones particulares (por ejemplo, en la carpeta `C:\WINNT\system32`) y configurar el servidor web para que lo soporte.

Aunque este tipo de instalación puede resultar un poco complicada, es recomendable utilizarla porque de esta forma PHP se instala como módulo SAPI, lo cual es más

seguro que instalarlo modo CGI. Además se incluyen muchas de las extensiones externas de PHP, y se pueden instalar de forma automática.

5.3 Instalación de PHP, bajo Linux

Ingresar al sistema como el usuario root y en /root descomprimir el archivo php-5.2.0.tar.gz, dicha acción habrá creado una carpeta /root/php-5.2.0/. Realice los siguientes pasos por línea de comandos:

1. cd /root/php-5.2.0
2. ./configure --prefix=/usr/local/php5
   --with-apxs2=/usr/local/apache2/bin/apxs
   --with-libxml-dir=/usr/local/lib
   --with-mysql=/usr/local/lib/mysql
   --with-unixODBC=/usr/local
3. make
4. make install
5. cp php.ini-dist /usr/local/lib/php.ini
6. cd /usr/local/php5/lib/
7. ln -s /usr/local/lib/php.ini php.ini
8. kate /usr/local/apache2/conf/httpd.conf
9. Añadir la siguiente línea al archivo: AddType application/x-httpd-php .php .phtml (ver Figura 1.3).

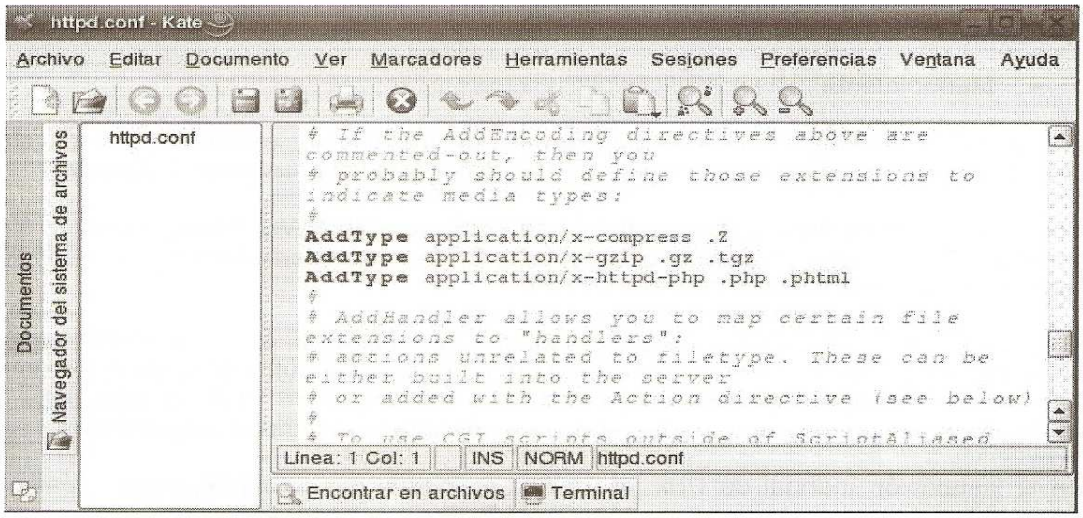


Figura 1.3: Modificar el archivo httpd.conf

10. /usr/local/apache2/bin/apachectl start
11. Crear archivo de nombre info.php haciendo uso del editor de texto de su preferencia, con el siguiente y único contenido: <?php echo phpinfo(); ?> y guardarlo en la ruta /usr/local/apache2/htdocs/ (ver Figura 1.6).

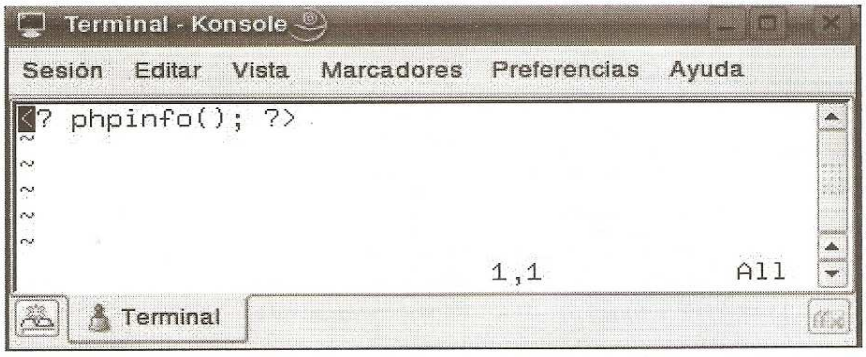


Figura 1.4: Archivo elaborado en Vi

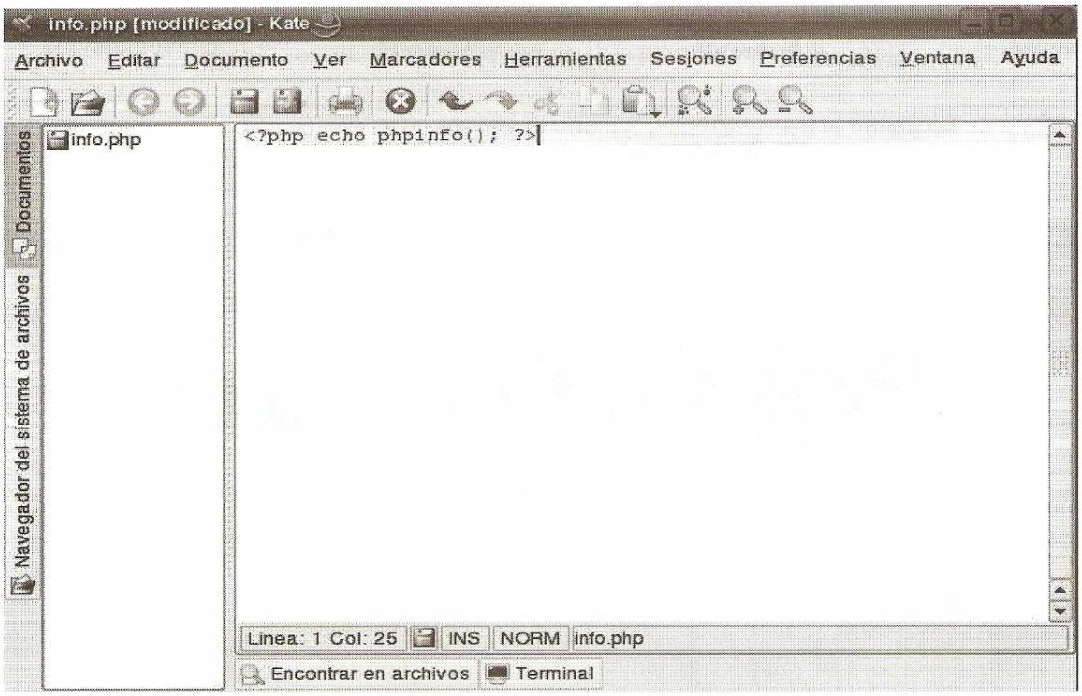


Figura 1.5: Archivo elaborado en Kate

Figura 1.6: Archivo guardado en la dirección especificada

12. Ejecutar el navegador y en la barra de direcciones colocar la ruta http://localhost/info.php, en caso de haber llevado a cabo una instalación exitosa se desplegará una página con información acerca de la instalación de Php.

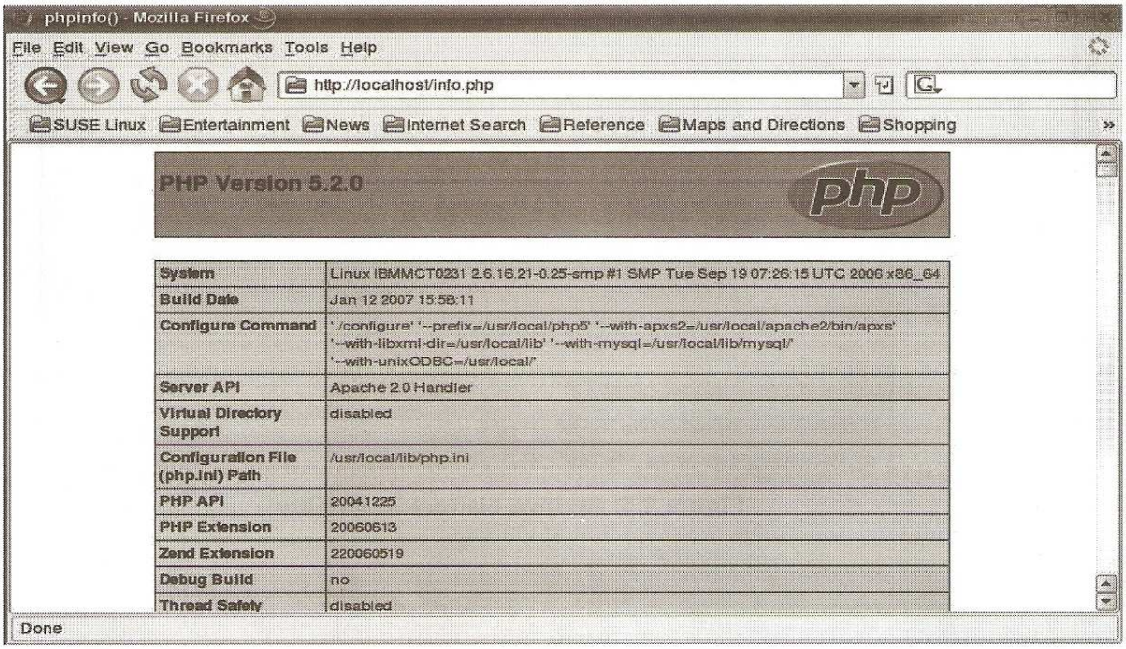


Figura 1.7: Pantalla que se visualiza al instalar Php exitosamente

5.4 ¿Cómo crear un script PHP?

El código PHP puede ser escrito en un editor de texto como NotePad, Notepad++ o WordPad. También puede utilizar otros editores orientados al desarrollo web como Macromedia HomeSite y herramientas WYSIWYG como Microsoft FrontPage y Macromedia Dreamweaver. Los archivos creados deben ser guardados con la extensión `.php`.

Para crear el código PHP se deben colocar unas etiquetas delimitadoras de inicio y fin de código PHP: `<?php ... ?>` . Estas etiquetas le indican al servidor que lo contenido en ese segmento es código PHP y debe ser interpretado como tal. También es común utilizar la forma abreviada de las etiquetas PHP: `<? ... ?>`.

Un ejemplo simple de un script PHP se muestra a continuación. En el código se puede observar etiquetas HTML y código PHP alternativamente, esto muestra que el código PHP puede estar embebido o incrustado dentro de HTML.

Nota: Para trabajar con palabras acentuadas en Firefox bajo Linux, se puede configurar (antes de copiar el código) tanto el editor Kate como el Kwrite de la siguiente manera: Herramientas – Codificación – Europeo Occidental (iso 8859-15).

Ejemplo 1.1

El ejemplo muestra en la ventana del navegador un texto sencillo.

El código PHP comienza aquí...

```
1.  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
2.  <HTML>
3.  <HEAD>
4.  <TITLE> Primer Script PHP - Ejemplo 1.1</TITLE>
5.  </HEAD>
6.  <BODY>
7.  <?php
8.  /* Este es un script PHP para mostrar la sintaxis básica.
9.  Utilizaremos la función print() */
10.  print("<H1>Hola Mundo PHP!</H1>\n");
11.  // La línea anterior muestra "Hola Mundo PHP!" en el
     navegador.
12.  // Ahora saldremos del modo PHP...
13.  ?>
14.  <!-- Aquí estamos en modo HTML -->
15.  <HR>
16.  <?
17.  //Tambien podemos utilizar las etiquetas PHP abreviadas
```

```
18.  echo "En 1995, <B>Rasmus Lerdorf</B> creó<B>PHP</B>, dandole
     el nombre inicial de <I>Personal Home Page Tools</I>. Luego,
     ese nombre fue cambiado a <I>PHP: Hypertext Preprocessor
     </I>\n<BR>.\n";
19.  ?>
20.  </BODY>
21.  </HTML>
```

El código PHP termina aquí

Nota: El uso de \n es para permitir que el código generado esté más ordenado, de manera que cuando se visualice por Ver Código Fuente, no quede todo en la misma línea.

Este código se crea utilizando cualquier editor. El archivo debe ser guardado con la extensión `.php` (por ejemplo, `hola.php`). Este archivo debe colocarse dentro del directorio raíz del servidor web que se esté utilizando o dentro de un directorio que sea accesible para el servidor.

La salida del código se observa en la Figura 1.8.

Nota: Si el Sistema Operativo utilizado es Linux, la ruta del directorio raíz del servidor web es: /home/"nombre_usuario"/www, esta ubicación hace referencia a /usr/local/apache2/htdocs/"usuario"/. Allí se deben guardar todos los archivos `.php`, que se quieran ejecutar. Para que el servidor Apache muestre páginas php, se debe hacer una solicitud al servidor web, colocando en la barra de dirección del navegador el siguiente URL: http://localhost/"usuario", esto mostrará el contenido de la carpeta donde se almacenan los códigos php. En el ejemplo, para ubicar la página sería http://localhost/db2admin/. También se pueden crear subdirectorios dentro del directorio raíz. En este caso se accede al servidor de la siguiente forma: http://localhost/db2admin/codigos.

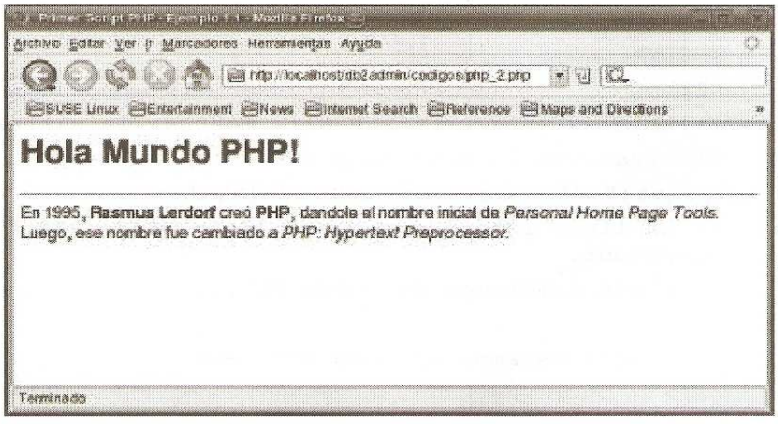


Figura 1.8: Salida del Código del Ejemplo 1.1

Cuando el servidor web recibe una solicitud para un archivo `.php`, pasa la solicitud al intérprete de PHP quien deja el código HTML tal como está, pero cuando llega al código PHP, procesa las instrucciones, generando como resultado código HTML que es enviado de vuelta al navegador que hizo la solicitud.

El código HTML generado por `hola.php`, es el siguiente:

El código HTML comienza aquí...

```
1.  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
2.  <HTML>
3.  <HEAD><TITLE> Primer Script PHP - Ejemplo </TITLE></HEAD>
4.  <BODY>
5.  <H1>Hola Mundo PHP!</H1>
6.  <HR>
7.  <!-- Aquí estamos en modo HTML -->
8.  En 1995, <B>Rasmus Lerdorf</B> creó <B>PHP</B>,     dándole
    el nombre inicial de <I>Personal Home Page Tools</I>.
9.  Luego, ese nombre fue cambiado a <I>PHP: Hypertext
    Preprocessor</I>.<BR>
10. </BODY>
11. </HTML>
```

El código HTML termina aquí

Fin del Ejemplo 1.1

Como se observó en el Ejemplo 1.1, cuando se desea agregar comentarios en código PHP, se utiliza el doble 'backslash' (`//`) o el 'numeral' (#) para comentar una línea, y la combinación de caracteres `/*` y `*/` para marcar el inicio y fin de un bloque de líneas comentadas.

La función `print()` emite o da salida a una cadena de caracteres simple, mientras que la función `echo " ... "`, da salida a una o más cadenas de caracteres.

Las instrucciones en PHP finalizan con el carácter de fin de sentencia `;` (punto y coma), como se pudo observar en el ejemplo anterior.

En un script PHP puede alternarse código HTML y PHP con la frecuencia que se necesite. Pero no se debe dejar todo el trabajo al intérprete de PHP. Es conveniente que el servidor web se dedique a procesar etiquetas HTML y sólo alternar con el interpretador PHP para que haga el procesamiento dinámico que amerite. Un ejemplo de lo que no se debe hacer es lo siguiente:

Ejemplo 1.2

El código PHP comienza aquí...

```
1.  <?php
2.   print("<HTML>\n");
3.   print("<HEAD>\n");
4.   print ("<TITLE> Primer Script PHP - Ejemplo 5.
     1.2</TITLE>\n");
5.   print("</HEAD>\n");
6.   print("<BODY>\n");
7.   print("<H1>Hola Mundo PHP!</H1>\n");
8.   print("En 1995, <B>Rasmus Lerdorf</B> creó <B>PHP</B>,
     dándole el nombre inicial de <I>Personal Home Page
     Tools</I>. Luego, ese nombre fue cambiado a <I>PHP:
     Hypertext Preprocessor </I>\n<BR>.\n ");
9.   print("</BODY>\n");
10.  print("</HTML>\n");
11.  ?>
```

El código PHP termina aquí

Fin del Ejemplo 1.2

El código del Ejemplo 1.2 realiza exactamente lo que produce el código HTML estático, pero obliga a que el servidor web pida más servicios del interpretador PHP, haciendo el procesamiento ineficiente.

En esta unidad se han discutido los fundamentos de PHP. En la próxima unidad se estudiarán los elementos del lenguaje PHP que permitirán crear scripts PHP más complejos.

Resumen

Ahora que ha completado esta unidad, usted debe ser capaz de:

- Describir PHP, sus características y su evolución.
- Discutir los beneficios y funcionalidades de PHP.
- Explicar cómo funciona un script hecho en PHP.
- Describir la arquitectura de PHP.
- Crear un script sencillo en PHP.

Unidad 1: Examen de Autoevaluación

1) ¿Cuáles son características del lenguaje PHP?
   a) Es un lenguaje scripting del lado del servidor
   b) Es un lenguaje propietario
   c) Se puede embeber o incorporar en HTML
   d) Es un lenguaje de código abierto (open source)

2) Los siguientes son componentes de la arquitectura PHP:
   a) HTML, sevidor web y navegador web
   b) Motor Zend, núcleo PHP y capa SAPI
   c) Scripts PHP, HTML y núcleo PHP
   d) Motor Zend, servidor web y navegador web

3) ¿Cuál es la función del motor Zend de PHP?
   a) Provee una interfaz a PHP para que interactúe de forma transparente con distintos servidores web
   b) Es un componente que permite crear aplicaciones de interfaz gráfica del lado del cliente
   c) Es una extensión que permite trabajar con bases de datos
   d) Es un intérprete que analiza el código PHP, lo traduce y lo ejecuta

4) Con PHP no se puede programar orientado a objetos
   a) Verdadero
   b) Falso

5) Cuáles de las siguientes son formas de instalar PHP
   a) Modo cliente
   b) Modo SAPI
   c) Modo servidor
   d) Modo CGI

6) Desde un script PHP no se puede interactuar con una base de datos.
   a) Verdadero
   b) Falso

7) ¿Cuáles de las siguientes son ventajas de PHP?
   a) Se puede utilizar programación procedimental o programación orientada a objetos
   b) Es un lenguaje de código abierto
   c) Se puede utilizar en plataformas bajo distintos sistemas operativos
   d) PHP fue especialmente diseñado para trabajar en el entorno web

8) ¿Cuál de las siguientes etiquetas se pueden utilizar para incorporar código PHP dentro de una página HTML?
   a) <!php !>
   b) <? ?>
   c) /* */
   d) <?php ?>

9) Con PHP se puede desarrollar:
   a) Scripts del lado del servidor
   b) Scripts en la línea de comandos
   c) Aplicaciones de interfaz gráfica
   d) Java Applets

10) Cuando un servidor web recibe una solicitud para un archivo `.php`:
   a) El servidor web procesa directamente el archivo `.php`
   b) El servidor web pasa la solicitud al intérprete PHP
   c) El intérprete PHP procesa las instrucciones
   d) El servidor web genera como resultado código HTML, que es enviado de vuelta al intérprete PHP que hizo la solicitud

Unidad 1: Respuestas al Examen de Autoevaluación

1) a, c y d
2) b
3) d
4) b
5) b y d
6) b
7) a, b, c y d
8) b y d
9) a, b y c
10) b y c

Unidad 2: Elementos del Lenguaje PHP

Objetivos de Aprendizaje

Al final de esta unidad, usted será capaz de:

- Explicar la sintaxis básica de PHP.
- Describir el uso de variables y constantes.
- Describir el uso de los operadores disponibles en PHP.
- Indicar los diferentes tipos de datos que posee PHP.
- Discutir las estructuras de control de PHP.

1. Introducción

En la Unidad 1 – *Fundamentos de PHP,* se discutió acerca de las características básicas de PHP, su arquitectura y funcionamiento. También se discutieron los elementos necesarios para crear y ejecutar scripts PHP del lado del servidor.

En esta unidad, se explicará la sintaxis de PHP y los elementos fundamentales que componen el lenguaje. Se comenzará con la sintaxis básica de PHP.

2. Sintaxis Básica de PHP

Antes de entrar en detalle acerca de los elementos que constituyen el lenguaje PHP, tales como, tipos de datos, variables, estructuras de control y funciones incorporadas, se hará una breve revisión de los principios básicos para construir un script PHP.

2.1 Segmentos de Código PHP

Cuando un servidor web recibe una solicitud por un archivo `.php`, pasa esa solicitud al motor de PHP. Las líneas HTML que contenga el archivo se quedan tal como están, pero cuando llega al segmento PHP el código es interpretado. El intérprete procesa las instrucciones y genera como resultado código HTML, que es incorporado en el archivo que se envía finalmente de vuelta al navegador.

Existen tres formas de delimitar los bloques de código PHP, utilizando los siguientes pares de etiquetas:

- <?php ... ?>
- <script language="php"> ... </script>
- `<? ... ?>`. Forma abreviada de las etiquetas (short-form tags)

Las dos primeras opciones están siempre disponibles, pero la forma abreviada de las etiquetas PHP (`<? ... ?>`), debe configurarse en el archivo `php.ini` para que sean aceptadas por el intérprete. (Se coloca la directiva `short_open_tag` en `"On"`)

Adicionalmente, se puede utilizar el estilo de etiquetas ASP: `<% ... %>`. Utilizar esta opción no garantiza la portabilidad entre los distintos servidores. También debe configurarse el `php.ini` para que sean aceptadas por el intérprete. (Se coloca la directiva `asp_tags` en `"On"`)

Cuando se incorpora código PHP dentro de código XHTML o XML, es obligatorio utilizar la forma de las etiquetas `<?php ... ?>`.

Se puede utilizar cualquier combinación de los formatos de las etiquetas antes mencionadas dentro de código PHP.

Seguidamente, se presenta un ejemplo donde se muestra el uso de estas etiquetas.

Ejemplo 2.1

En el siguiente ejemplo se observará que el código contiene etiquetas HTML y código PHP embebido, utilizando las distintas opciones para las etiquetas de inicio y fin de PHP.

El código PHP comienza aquí...

```
1.  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
2.  <HTML>
3.  <HEAD>
4.  <TITLE> Etiquetas PHP - Ejemplo 2.1 </TITLE>
5.  </HEAD>
6.  <BODY>
7.  <?php
8.   echo ("Esta es la forma recomendada para delimitar código
    PHP <BR>\n");
9.  ?>
10. <P> Aquí estamos en modo HTML </P>
11. <?
12.  echo ("Esta es la forma abreviada de las etiquetas
    PHP<br>\n");
13. ?>
14. <SCRIPT language="php">
15.  echo("En este segmento utilizamos la etiqueta
    SCRIPT<BR>\n")
16. </SCRIPT>
17. <%= "Opcionalmente puedo utilizar las etiquetas ASP, pero
    esta opción debe estar configurada en el php.ini<BR>\n " %>
18. </BODY>
19. </HTML>
```

El código PHP termina aquí...

Fin del Ejemplo 2.1

Es recomendable utilizar el formato de etiquetas PHP completo (`<?php ... ?>`), pues se garantiza la compatibilidad entre las diferentes versiones de PHP para distintos servidores web. Además no es necesario configurar ninguna opción en el `php.ini` para que sean aceptadas por el intérprete.

2.2 Finalización de Sentencias

Las instrucciones PHP finalizan con punto y coma (`;`), a menos que la instrucción sea la última antes de la etiqueta de fin de PHP.

```
<?php print("hola a todos!"); ?>
```

es equivalente a:

```
<?php print("hola a todos!") ?>
```

Tampoco se coloca punto y coma después de la etiqueta de inicio PHP (`<?php` ó `<?`) ni después de la etiqueta de fin (`?>`).

2.3 Uso de las Mayúsculas y Minúsculas

Generalmente, un lenguaje de programación es sensible a mayúsculas y minúsculas ('case sensitive') o no lo es ('case insensitive'). Pero en PHP el comportamiento es muy particular. A continuación se describen los siguientes casos:

- Las variables y constantes definidas por el usuario son sensibles a mayúsculas y minúsculas.
- Las variables y constantes predefinidas de PHP son sensibles a mayúsculas y minúsculas.
- Las funciones definidas por el usuario no son sensibles a mayúsculas y minúsculas.
- Las funciones incorporadas de PHP no son sensibles a mayúsculas y minúsculas.
- Es recomendable mantener la uniformidad al nombrar los diferentes elementos y así evitar confusiones. Más adelante en esta unidad, se indicarán las reglas y convenciones para la nomenclatura de variables, constantes y funciones en PHP.

2.4 Comentarios

Existen tres formas de colocar comentarios dentro del código PHP.

- Utilizar el doble 'backslash' (`//`). Sirve para comentar una sola línea.
- Utilizar la combinación de de caracteres `/*` y `*/`. Sirve para comentar varias líneas, formando un bloque de comentarios.
- Utilizar el carácter 'numeral' (#). Sirve para comentar una única línea.

2.5 Mostrar la Salida en el Navegador

Básicamente, para mostrar la salida al navegador se tienen tres posibilidades:

- **Utilizar la función `print()`**. Con esta función, incorporada en PHP, se puede escribir una cadena resultado. Su sintaxis es `print(S)`.

```
1.  <?php
2.  print("<STRONG>Muestro la salida con PRINT</STRONG><BR>");
3.  ?>
```

- **Utilizar la función echo ()**. Trabaja de forma similar a la función `print()`, pero usando esta función, se pueden escribir varias cadenas a la vez. Su sintaxis es:
  `echo (S1[ ,S2 ... ]).`

Para escribir una sola cadena:

```
1.  <?php
2.  echo("<STRONG>Escribo el resultado con ECHO
    </STRONG><BR>\n");
3.  ?>
```

Para escribir varias cadenas.

```
4.  <?php
5.  echo "1era cadena ","2da cadena ","3era cadena";
6.  ?>
```

Nótese que cuando se imprimen múltiples cadenas no se utilizan los paréntesis.

- **Utilizar las etiquetas <?= ?>**. Esta es una forma abreviada de escribir una cadena de salida. Se utiliza de la siguiente manera:
  `<?="Esto es equivalente a utilizar echo o print"?>`

Es importante destacar que el uso de estas funciones no está limitado solamente a cadenas de caracteres. Con `print()`, `echo()` ó `<?= ?>`, se puede escribir cualquier expresión, de la siguiente forma:

- <?php print(EXPR); ?>
- <?php echo(EXPR); ?>.
- <?php echo EXPR1,EXPR2...; ?>
- <?=EXPR ?>.

Donde `EXPR` representa una expresión, que puede ser:

- Una variable que contenga un valor de cualquier tipo de dato.
- Una constante que contenga un valor de cualquier tipo de dato.
- El resultado de la evaluación de una función.
- El resultado de la concatenación de dos o más expresiones.

A continuación se presenta un ejemplo donde se pone en práctica todo lo mencionado hasta ahora.

Ejemplo 2.2

Este script PHP proporciona un ejemplo de lo que se ha contemplado en los apartados 2.1 al 2.5. Simplemente imprime algunos mensajes, incorpora comentarios y muestra el uso de las distintas etiquetas PHP.

El código PHP comienza aquí...

```
1.  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
2.  <HTML>
3.  <HEAD>
4.  <TITLE> Sintaxis Básica de PHP - Ejemplo 2.2 </TITLE>
5.  </HEAD>
6.  <BODY>
7.  <? #Inicio del código PHP. Estoy utilizando el tipo de
    comentario 'Shell' . ?>
8.  <?="<H1>Sintaxis Básica de PHP</H1>"?>
9.  <?="<HR>"?>
10. <?php
11. echo ("Esta instrucción finaliza con punto y coma
    pues le sigue otra instrucción...<BR>\n");
12. EcHo "Una FuNcIóN no es sensible a "," MAYÚSCULAS y
    minúsculas...","  Aquí finaliza el primer segmento. <BR>\n"
13. //No es necesario finalizar con ; porque es la última
    instrucción de este segmento.
14. ?>
15. <HR><!-- Estoy en modo HTML-->
16. <?
17.         /*Comienzo del segundo segmento de código.
18. (Puedo escribir varias líneas de comentario).
19. Ahora usaremos la función print.*/
20.  PRINT("Mostramos un número... <BR>\n");
21.  $num=250;
22.  echo "NÚMERO: ",$num;
23. ?>
24. <HR>
25. <SCRIPT language="php">
26.  echo("En este segmento utilizamos la etiqueta
     SCRIPT<BR>\n")
27. </SCRIPT>
28. <HR>
29. <?="<B>FIN!!</B>\n";?>
30. </BODY>
31. </HTML>
```

El código PHP termina aquí...

El resultado de la ejecución de `ejemplo-2.2.php`, se muestra en la Figura 2.1.

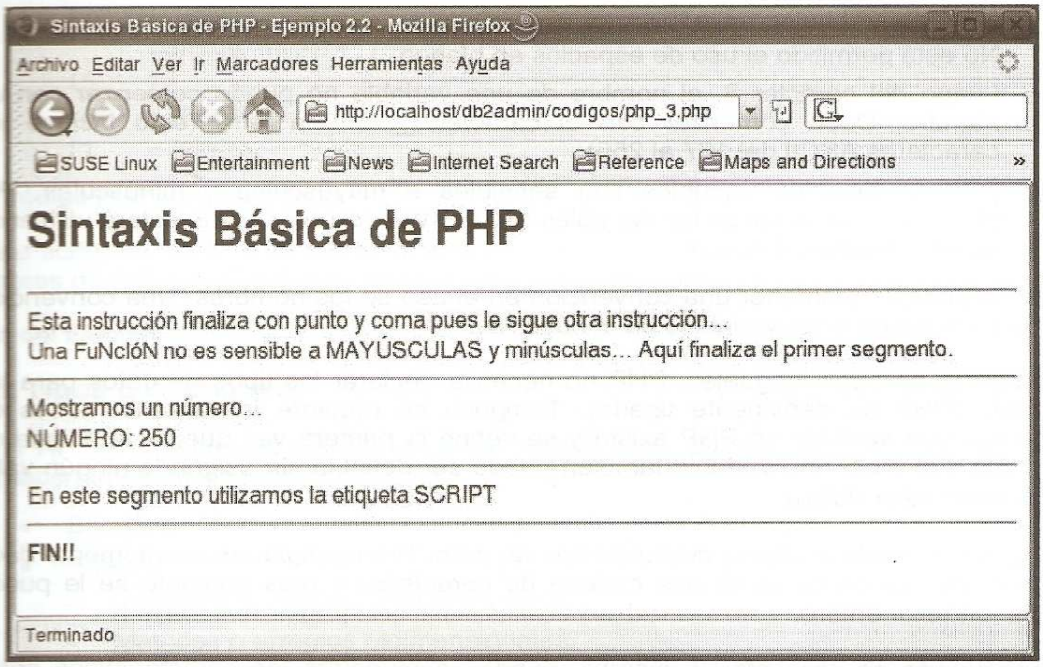


Figura 2.1: Resultado de la ejecución del ejemplo-2.2.php

Fin del Ejemplo 2.2

Una vez discutidos los aspectos fundamentales de la sintaxis de PHP, se continuará con los elementos que componen el lenguaje, los cuales permitirán realizar programas en PHP más completos. Se iniciará con las variables y constantes en PHP.

3. Variables

Una variable en PHP es un contenedor de datos temporal, es decir, sólo existe y mantiene su valor mientras se ejecuta el script. Los datos que contienen las variables pueden ser modificados, guardados en archivos o bases de datos o enviados para que sean mostrados en el navegador; por ejemplo mediante el uso de la función `print()`.

3.1 Convenciones para nombrar variables

Algunas convenciones para nombrar variables en PHP, son las siguientes:

- En PHP el nombre de las variables deben comenzar con el signo $ seguido con una letra en los rangos (A – Z , a – z) o el carácter 'underscore' (_), luego puede tener una combinación de letras en los rangos (A – Z, a – z), números (0 – 9), el carácter especial 'under score' (_) y los caracteres ASCII del 127 al 255 (extendidos).
- No está permitido el uso de espacios en blanco.
- Luego del carácter $, el nombre de una variable no puede comenzar con un número. Debe comenzar con el carácter '_' o una letra (esto incluye los caracteres ASCII del 127 al 255).
- Los nombres de variables son sensibles a mayúsculas y minúsculas. Por ejemplo, si se emplean las variables `$VAR`, `$Var`, `$var` y `$vAr`, estará utilizando cuatro variables distintas.

Es recomendable mantener una convención en el uso de los nombres. Una convención común es nombrar a las variables en minúsculas.

A diferencia de otros lenguajes, PHP no necesita declarar los tipos de datos para las variables (PHP es débilmente tipado). Tampoco se requiere inicializarlas antes de utilizarlas. Una variable en PHP existe y se define la primera vez que se le asigna un valor. Sin embargo, una variable también puede ser definida sin asignarle ningún valor (toma como valor `NULL`).

Una variable puede contener cualquier tipo de dato. Por ejemplo, en un momento dado una variable puede contener una cadena de caracteres y más adelante se le puede asignar un valor entero.

A continuación se presenta un ejemplo donde se muestran las características antes mencionadas.

Ejemplo 2.3

Este script PHP muestra cómo pueden nombrarse variables de forma correcta e incorrecta.

El código PHP comienza aquí...

```
1.  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
2.  <HTML>
3.  <HEAD>
4.  <TITLE> Sintaxis Básica de PHP - Ejemplo 2.3 </TITLE></HEAD>
5.  <BODY>
6.  <?php
7.  // * Usos válidos de variables *
8.    $var1 = "Variable var1";
9.    $VAR1 = "Variable VAR1, distinta a var1";
10.   $_var1 = "Variable _var1";
```

```
11.    $var¿ñé1 = "Variable con caracteres ASCII
       extendidos";
12.    // * Usos no válidos de variables *
13.    var4 = "Se debe utilizar el carácter $ al inicio";
14.    $5var = "El nombre de una variable no puede comenzar
       con un número";
15.  ?>
16.  </BODY>
17.  </HTML>
```

El código PHP termina aquí...

Este script no se ejecuta correctamente, pues se incluyen líneas de código que tienen errores de sintaxis. El ejemplo es sólo para efectos ilustrativos.

Fin del Ejemplo 2.3

3.2 Tipos de Variables

En PHP las variables se pueden clasificar en dos tipos, de acuerdo a los valores que se almacenan en ellas:

- Escalares.
- Arreglos:
  - Vectores o arreglos unidimensionales.
  - Matrices o arreglos bidimensionales.
  - Arreglos n-dimensionales.

3.3 Variables Escalares

Una variable escalar se usa para almacenar un valor individual. PHP es un lenguaje débilmente tipado, esto quiere decir que no se necesita declarar explícitamente un tipo de dato para una variable. Además una variable puede almacenar datos de cualquier tipo. En el Ejemplo 2.3 se pueden observar ejemplos de variables escalares (`$var1`, `$VAR1`, `$_var1`) que almacenan cadenas de caracteres.

En el Ejemplo 2.3 se utilizaron nombres arbitrarios para las variables, pero es importante recordar que una convención para los nombres de variables es colocarlos en minúsculas. Si el nombre de la variable está compuesto de varias palabras, la primera letra de cada palabra debe estar en mayúscula. Observe los siguientes ejemplos:

```
$edad
$edadEstudiante
```

3.3.1. Asignación de Valores

Como se ha podido observar en los ejemplos presentados hasta ahora, se utiliza el operador '=' (igualdad), para asignarle valores a las variables. En el siguiente código, se muestra cómo asignar valores a variables escalares.

```
1.  <?php
2.      $var; /* Se declara una variable sin ningún valor.
3.      Su contenido es NULL */
4.      $var0 = NULL; // Explícitamente se le asigna
    NULL
5.      $var1 = "Variable 1"; // Se le asigna una cadena
6.      $var2 = 2005;// Se le asigna un número entero
7.      $var3 = 2150.75;// Se le asigna un punto
    flotante
8.      $var4 =(25*9+18); // Se le asigna el resultado de una
    expresión
9.      $var5=$var4; // Se le asigna el contenido de $var4
10. ?>
```

3.3.2. Asignación de Valores por Referencia

A partir de PHP 4 se tiene otra forma de asignar valores a las variables: *La asignación por referencia*. Esto significa que la variable a la que se le asigna la referencia se convierte en "un alias de" o "apunta a" la variable original asignada. Los cambios a la nueva variable afectan a la original y viceversa. Esto también significa que no se produce una copia de valores, por lo tanto la asignación ocurre más rápido. Para asignar una variable por referencia se antepone al nombre de la variable asignada el carácter 'ampersand' (&).

```
1.  <?php
2.      $origen = "Valor original <BR>";
3.      $referencia = &$origen;
4.      $referencia = "Nuevo valor <BR>";
5.      echo "Variable \$referencia: ",$referencia;
6.      echo "Variable \$origen: ", $origen;
7.          //En ambos casos se imprime 'Nuevo valor'
8.  ?>
```

3.4 Variables Arreglos

Los arreglos (arrays) son colecciones de números, cadenas, otros arreglos o una combinación de éstos, ensamblados en una sola variable, permitiendo guardar múltiples valores, a diferencia de las variables escalares. Los elementos de datos individuales en un arreglo pueden ser de cualquier tipo, como enteros, flotantes o caracteres.

En PHP, al igual que en Perl, y a diferencia de otros lenguajes de programación, los arreglos no son uniformes, es decir, un mismo arreglo puede contener valores de distinto tipo en cada una de sus posiciones.

En PHP los arreglos pueden trabajar como tablas hash (arreglos asociativos) o como arreglos indexados (vectores). Los arreglos pueden ser unidimensionales o multidimensionales.

3.4.1 Acceso a los Elementos de un Arreglo

Los elementos de un arreglo se pueden acceder a través de un índice para los arreglos indexados, y a través de una clave, en el caso de los arreglos asociativos o hashes.

- Acceso a un arreglo unidimensional:

```
$var1 = $vector[ i] ;
$vector[ j] = $var1;
//i y j representan las posiciones de los elementos en el
arreglo
$var1 = $hash[ "clave1"] ;
$hash[ "clave2"] = $var1;
// "clave1" y "clave 2" representan las posiciones de los
// elementos en el arreglo asociativo
```

- Acceso a un arreglo bidimensional:

```
$matriz[ i][ j] = $var2;
//i representa la fila y j la columna donde está el
elemento
```

- Acceso a un arreglo multidimensional.

```
$multi[ i][ j][ k][ l] = $var4;
// Aquí tenemos un arreglo de cuatro dimensiones
```

3.4.2 Creación de Arreglos

Para crear un arreglo se pueden utilizar algunas de las opciones mostradas a continuación.

- Crear un arreglo por asignación de valores indicando un índice:

```
<? php
# Crear un arreglo unidimensional por asignación de valores
    $arreglo[ 0] = "primer valor";
    $arreglo[ 1] = 2;
    $arreglo[ 2] = 3.5;
    $arrreglo[ ] = "último valor";
# Crear un arreglo bidimensional por asignación de valores
```

```
        $matriz[ 0][ 0]  = 100;
        $matriz[ 0][ 1]  = 200;
        $matriz[ 1][ 0]  = 10.75;
        $matriz[ 1][ 1]  = 20.75;
    ?>
```

El índice de los arreglos comienza en cero (0) y cada posición del arreglo puede contener cualquier valor, es decir, el contenido de un arreglo no está limitado a un sólo tipo de dato.

Cuando se asigna un valor a un arreglo usando corchetes vacíos (`$arrreglo[ ] =$valor`), el valor se agrega al final del arreglo.

También se puede crear arreglos asociativos. Esto se logra utilizando una clave en vez de un índice.

- Crear un arreglo por asignación de valores indicando una clave

```
1.  <?php
2.  # Crear un arreglo asociativo unidimensional por asignación
    de valores #
3.      $arreglo[ "primero"]  = "primer valor";
4.      $arreglo[ "segundo"]  = "segundo valor";
5.      $arreglo[ "tercero"]  = "tercer valor";
6.      echo $arreglo[ "segundo"] ; //Se imprime 'segundo valor'
7.      echo "<BR>";
8.  # Crear un arreglo asociativo bidimensional por asignación
    de valores #
9.      $matriz[ "cero"][ "cero"]  = 100;
10.     $matriz[ "cero"][ "uno"]  = 200;
11.     $matriz[ "uno"][ "cero"]  = 300;
12.     $matriz[ "uno"][ "uno"]  = 400;
13.     echo $matriz[ "uno"][ "cero"] ; // Se imprime '300'
14. ?>
```

También se puede mezclar índices numéricos y asociativos. Por ejemplo:

```
$matriz[ 1][ "cero"]  = 100;
$var1 = $matriz[ 1][ "uno"] ;
```

Nota: Para poder recuperar el valor utilizando índices numéricos y asociativos, previamente los índices del arreglo deben haber sido definidos de la misma forma. Si los índices no están definidos de la misma forma que cuando se creó el arreglo, al tratar de recuperar el elemento con esos índices se genera un error.

- Crear un arreglo utilizando la función **`array()`** e índices numéricos.

```
1.  <?php
```

```
2.    # Crear un arreglo unidimensional con array() #
3.      $arreglo1 = array(10,20,30,40,50);
4.      $arreglo2 = array("carro","moto","patín");
5.
6.    # Otra forma de crear un arreglo unidimensional con array()
      #
7.      $arreglo = array(0=>10,1=>20,2=>30,3=>40,4=>50);
8.      echo $arreglo[ 3] ; //Imprime '40'
9.      echo "<BR>";
10.    # Crea un arreglo bidimensional con array()#
11.     $matriz = array(
      0=>array(0=>"00",1=>"01",2=>"02"),
12.     1 => array(0=>"10",1=>"11",2=>"12"),
13.     2 => array(0=>"20",1=>"21",2=>"22"));
14.     echo $matriz[ 1][ 2] ; //Imprime '12'
15.   ?>
16.
```

- Crear un arreglo utilizando la función `array()` e índices asociativos.

```
1.    <?php
2.    # Creando un arreglo unidimensional con array() #
3.          $colores = array("am"=>"amarillo",
4.         "az"=>"azul",
5.         "vd"=>"verde",
6.         "rj"=>"rojo",
7.          "ng"=>"negro");
8.          echo $colores[ "vd"] ; //Imprime 'verde'
9.          echo "<BR>";
10.   # Creando un arreglo bidimensional con array() #
11.             $frutas=array(
12.   "pera"=>array("color"=>"amarillo",
13.           "sabor"=>"dulce" ),
14.   "manzana"=>array("color"=>"rojo",
15.        "sabor"=>"dulce"),
16.   "limon"=>array("color"=>"verde",
17.                  "sabor"=>"ácido"));
18.   echo $frutas[ "limon"][ "sabor"] ; //Imprime 'ácido'
19.   ?>
20.
```

- Si se utiliza la función array sin parámetros se crea una arreglo vacío.

```
1.    <?php
```

```
2.   # Creando un arreglo vacío #
3.   $colores = array();
4.   $colores[] = "amarillo"; # Ahora tiene un elemento
5.   ?>
```

Existen muchas funciones en PHP que permiten trabajar con arreglos. Estas funciones serán discutidas en la Unidad 3 – *Funciones y Extensiones PHP*.

A continuación se discutirá el alcance de las variables en PHP.

3.5 Alcance de las Variables

El alcance de una variable es el contexto dentro del cual la variable está definida y donde la variable puede ser utilizada. La mayor parte de las variables PHP sólo tienen alcance global. Ese alcance también comprende los archivos incluidos.

```
1.   <?php
2.   $var = 1;
3.   include "archivo.php";
4.   ?>
```

La variable `$var` estará disponible dentro del script incluido `"archivo.php"`.

Nota: En el ejemplo, la sentencia **`include()`**, sirve para incluir y evaluar un archivo especificado.

3.5.1 Variables Globales y Locales

Dentro de las funciones definidas por el usuario se tiene el *alcance local*. Es decir, cualquier variable que se use dentro de una función está limitada al contexto local de la función.

```
1.   <?php
2.   $var = 1; /* variable con alcance global */
3.   function miFuncion(){
4.   $var = 10;
5.   echo $var; /* Se hace referencia a la variable
6.     local $var */
7.   echo "<BR>";
8.   }
9.   miFuncion(); /* Se imprime 10*/
10.  echo $var; /* Se imprime 1*/
11.  ?>
```

A diferencia de lenguaje C, donde las variables globales están disponibles automáticamente dentro de la función, a menos que sean expresamente sobrescritas por una definición local, en PHP las variables globales no están disponibles

explícitamente dentro de la función. Esto es una forma de prevención, porque se pueden cambiar los valores de las variables globales inadvertidamente.

En PHP, las variables globales deben ser declaradas explícitamente como globales dentro de la función si van a ser utilizadas. Observe un ejemplo:

```
1.  <?php
2.  $glob1 = 10; $glob2 = 20;
3.  function sumaGlob(){
4.   global $glob1, $glob2;
5.   $glob2 = $glob1 + $glob2;
6.  }
7.  sumaGlob();
8.  echo $glob2; // Se imprime 30
9.  ?>
```

Al declarar `$glob1` y `$glob2` como globales dentro de la función, todas las referencias a tales variables se referirán a las globales.

Un segundo método para acceder a las variables globales es usando la variable predefinida de PHP `$GLOBALS`. El ejemplo anterior se puede escribir así:

```
1.  <?php
2.  $glob1 = 10;
3.  $glob2 = 20;
4.  function sumaGlob(){
5.  $GLOBALS[ "glob2"] = $GLOBALS[ "glob1"] + $GLOBALS[ "glob2"];
6.  }
7.  sumaGlob();
8.  echo $glob2; //Se imprime 30
9.  ?>
```

La variable predefinida `$GLOBALS` es un arreglo asociativo que usa el nombre de la variable global como clave y el contenido de dicha variable como el valor del elemento del arreglo. `$GLOBALS` existe en cualquier alcance. Esto pasa porque `$GLOBALS` es una variable predefinida de PHP *superglobal*. Este concepto se estudiará en la Unidad 5 – *Desarrollo de Aplicaciones Web*.

2.5.2 Variables Estáticas

Una variable estática (static) existe sólo en el alcance local de la función, pero no pierde su valor cuando la ejecución sale del alcance local. Observe el siguiente ejemplo:

```
1.  <?php
2.  function prueba(){
3.      static $estatica = 0;
```

```
4.        echo $estatica;
5.        $estatica++;
6.        print $estatica; }
7.    prueba(); // se imprime 1
8.    echo "<BR>";
9.    prueba(); // se imprime 2
10.   echo "<BR>";
11.   prueba(); // se imprime 3
12.   ?>
```

En este ejemplo, en cada llamada a la función `prueba()` la variable `$estatica`, contendrá el valor que alcanzó cuando salió de la invocación anterior.

3.6 Variables Predefinidas del Lenguaje

PHP proporciona una gran cantidad de variables predefinidas que se utilizan para almacenar valores específicos cuando se ejecutan los scripts PHP. Estas variables pueden ser accedidas dentro del script, lo cual resulta muy útil en el desarrollo de programas. Sin embargo, el número, propósito y nombre de esas variables varía dependiendo del servidor que se esté ejecutando, versión, configuración de dicho servidor y otros factores. Por lo tanto, muchas de estas variables no se encuentran documentadas. Observe algunos ejemplos de variables predefinidas de PHP:

- La variable predefinida `$GLOBALS`, es un arreglo asociativo que contiene una referencia a cada variable global disponible dentro del script.
- `$_SERVER` guarda las variables definidas por el servidor web.
- `$_ENV` guarda las variables de entorno del script PHP que se esté ejecutando.

Aunque la revisión de todas las variables predefinidas de PHP, está fuera del alcance de este curso, una explicación detallada acerca de algunas variables predefinidas la encontrará en la Unidad 5 – *Desarrollo de Aplicaciones Web*.

4. Constantes

Una constante es un identificador que hace referencia a un valor simple. Como el nombre sugiere, este valor no puede variar durante la ejecución del script. En PHP una constante es sensible a mayúsculas y minúsculas, al igual que las variables. Por convención, las constantes suelen declararse en mayúsculas.

Estas son algunas de las reglas para nombrar constantes:

- No son precedidas por un símbolo de dólar ($).
- El nombre de las constantes debe comenzar con una letra en los rangos (A – Z, a – z), o el carácter 'underscore' (_), seguido por una combinación de letras, números (0 – 9), 'underscore' (_) y los caracteres ASCII del 127 al 255.

- Sólo pueden ser definidas usando la función **`define()`**, nunca por simple asignación.
- Pueden ser definidas y accedidas sin tener en cuenta las reglas de alcance. El alcance de una constante es global, es decir, una vez definida se puede acceder desde cualquier punto del script.
- No pueden ser redefinidas o eliminadas después de establecerse.
- Sólo pueden contener valores escalares.

El siguiente código muestra cómo definir una constante en PHP.

```
1.  <?php
2.  define("MI_CONSTANTE", "Hola Mundo PHP!");
3.  echo MI_CONSTANTE; // Se imprime 'Hola Mundo PHP!'
4.  ?>
```

La función **`defined()`**, se utiliza para determinar si una constante ya está definida. Retorna `TRUE`, si está definida y `False` en caso contrario.

```
1.  <?php
2.  define("MI_CONSTANTE", "Hola Mundo PHP!");
3.  $t = defined("MI_CONSTANTE"); //retorna true
4.  $f = defined("OTRA_CONSTANTE"); // retorna false
5.  ?>
```

4.1 Constantes Predefinidas en PHP

PHP ofrece un largo número de constantes predefinidas para cualquier script en ejecución. Sin embargo, muchas de estas constantes, son creadas por diferentes extensiones y sólo estarán presentes si dichas extensiones están disponibles. Las constantes predefinidas guardan información acerca de la configuración de PHP o de las extensiones.

Las constantes predefinidas, pueden ser:

- **Constantes Base Predefinidas:** Son constantes definidas en el núcleo de PHP, el motor Zend y los módulos SAPI. Por ejemplo, la constante `PHP_VERSION` indica la versión de PHP que está instalada y `PHP_OS` indica el sistema operativo para el cual se instaló PHP.
- **Constantes Estándar Predefinidas:** Son constantes definidas en PHP por defecto. Se utilizan para indicar opciones en algunas instrucciones. Por ejemplo, la constante `SORT_ASC`, se utiliza con la función `array_multisort()` para indicar que el arreglo se ordene ascendentemente. `M_SQRT2` es una constante que tiene el valor de la raíz cuadrada de dos.

La revisión de todas las constantes predefinidas en PHP, está fuera del alcance de este curso.

5. Tipos de Datos

En PHP el tipo de dato para una variable no se asigna explícitamente. Por el contrario, es asignado por PHP en tiempo de ejecución, de acuerdo al valor que almacene la variable. Sin embargo, si se quiere obligar a que una variable se convierta a un tipo concreto, se puede utilizar la función **`settype()`**. Observe el siguiente segmento de código:

```
1.  <?php
2.      $var1 = "PHP es débilmente tipado";
3.      $result = settype ($var1, "string");
4.  /* Retorna true si se puede establecer el tipo, false en
    caso contrario. */
5.  ?>
```

La sintaxis para `settype()` es la siguiente: **`settype ($var, $tipo)`**.

Donde `$tipo` puede contener los siguientes valores: `"string"`, `"integer"`, `"double"`, `"array"` y `"object"`.

5.1 Números Enteros

Las variables adquieren el tipo entero, si los valores con los que se inicializan las mismas son enteros. Por ejemplo:

```
1.  <?php
2.      $entero = 1234; //número entero positivo base 10
3.      $entero = -123; //número entero negativo
4.      $entero = 0123; //número octal (equivalente al 83
    base          10)
5.      $entero = 0x12; //número hexadecimal (equivalente al 18)
6.  ?>
```

5.2 Números Punto Flotante

Las variables adquieren el tipo punto flotante si los valores con que se inicializan son números reales. Por ejemplo:

```
1.  <?php
2.  $flotante = 250.75; /* El carácter que indica la parte
    decimal es el punto (.) */
3.  $flotante = 3.4e3; /* Se utiliza notación científica
    para asignarle el valor a la variable */
4.  ?>
```

5.3 Lógicos o Booleanos

Las variables adquieren el tipo booleano o lógico si el valor con que se inicializan es `true` o `false`. Por ejemplo:

```
1.  <?php
2.      $bool = true; // $bool tiene el valor true
3.      $bool = false; // Ahora se le asigna el valor false
4.  ?>
```

5.4 Cadenas

Las variables adquieren el tipo cadena si el valor asignado a la variable está encerrado entre comillas dobles (" ") o simples (' '). Las cadenas pueden contener cualquier valor.

```
1.  <?php
2.      $cadena1 = "Valor de cadena con comillas dobles";
3.      echo "<BR>";
4.      $cadena2 = 'Valor de cadena con comillas simples';
5.  ?>
```

Si la cadena se delimita con comillas dobles (" "), las variables que estén dentro de la cadena serán expandidas o interpoladas.

Si la cadena se delimita con comillas simples (' '), las variables que estén dentro de la cadena no serán expandidas o interpoladas.

5.4.1 Interpolación de Cadena

La interpolación significa reemplazar un símbolo o variable con su contenido. En PHP, la interpolación de cadena significa que siempre que una variable escalar sea colocada dentro de una cadena con comillas dobles, será interpolada en la cadena, es decir, se mostrará su contenido, no el nombre de la variable. En otras palabras, cuando las comillas dobles (" ") se usan, PHP sabe que tiene que sustituir el valor de la variable cuando lo imprima. Observe el siguiente segmento de código.

```
1.  <?php
2.          $mensaje = "Bienvenido a IBM";
3.          print "$mensaje";
4.  ?>
```

La salida será la siguiente:

```
Bienvenido a IBM
```

El backslash (\ ) se utiliza, del mismo modo que en C o Perl, como un carácter de escape. Si el backslash precede al $ dentro de las comillas dobles, será interpretado como un '$' en el literal y el valor de la variable no será sustituido. Observe el siguiente segmento de código.

```
1.  <?php
2.          $mensaje = "Bienvenido a IBM";
```

```
3.              print "\$mensaje";
4.    ?>
```

La salida será la siguiente:

```
$mensaje
```

Las secuencias de escape funcionan de la siguiente manera:

Si se coloca el backslash (\ ) delante de un carácter que tiene un significado o función especial en el lenguaje, el carácter en cuestión pierde sus atributos y es interpretado literalmente. Por ejemplo, si se coloca "\$", el $ ya no será interpretado como prefijo de variable sino como el carácter "$".

Por el contrario, si se antecede el backslash (\ ) delante de un carácter que no tiene un significado particular en el lenguaje, el mencionado carácter adquiere una nueva característica. Por ejemplo, si se antecede el backslash a la letra "n", ésta será interpretada como el carácter de nueva línea "\n".

En la Tabla 2.1 se muestran algunas secuencias de escape y su significado.

Secuencia	Significado
\n	Nueva línea
\r	Retorno de carro
\t	Tabulación horizontal
\\	Barra invertida (backslash)
\$	Signo dólar
\"	Comilla doble
\'	Comilla simple

Tabla 2.1: Secuencias de Escape en PHP

5.4.2 No Interpolación de Cadena

Cuando una variable es colocada dentro de comillas simples, no será interpolada en la cadena. Esto significa que la cadena será interpretada literalmente. Entiéndase esto con un ejemplo.

```
1.    <?php
2.              $mensaje = "Bienvenido a IBM";
3.              print '$mensaje';
```

```
4.    ?>
```

La salida será la siguiente:

```
$mensaje
```

Observe otro ejemplo del uso de la barra invertida '\'.

Suponga que se usan comillas simples para definir una cadena, pero la cadena tiene una comilla simple dentro de ella, por ejemplo:

```
$mensaje = 'why don't you understand?';
```

PHP no será capaz de interpretar esta cadena correctamente debido a que la comilla en "don't" señalará el fin de la cadena. Para superar esto se tiene que anteceder la comilla con un backslash '\'.

```
1.    <?php
2.          $mensaje = 'Why don\'t you understand?';
3.          print "$mensaje";
4.    ?>
```

La salida será la siguiente:

```
Why don't you understand?
```

5.5 Conversión de Tipos

En PHP el tipo de una variable se determina por el contexto en el que se usa esa variable. Esto significa que si se le asigna un valor de cadena a la variable `$var`, `$var` se convierte en `string`. Si luego se le asigna un valor entero, se convierte en una variable de tipo `integer`.

Un ejemplo de conversión de tipo automática en PHP es el operador suma '+' (y también el resto de los operadores aritméticos). Si cualquiera de los operandos es un punto flotante, entonces todos los operandos se evalúan como punto flotantes y el resultado será un punto flotante. En caso contrario, los operandos se interpretarán como enteros y el resultado será también un entero.

En caso de que alguno de los operadores sea una cadena, si la cadena representa un número, entonces se tomará como un número (entero o punto flotante) y luego se realizará la operación. Observe el siguiente segmento de código.

```
1.    <?php
2.        # Conversión automática de tipos #
3.        $var1 = "100";
4.        $var2 = "25.75";
5.        echo $var1 + $var2; //Imprime 125.75
6.        echo "<BR>";
```

```
7.        echo $var1 * $var2; //Imprime 2575
8.        echo "<BR>";
9.        $var3 = "casa";
10.       $var4 = 20;
11.       echo $var3 + $var4; //Imprime 20
12.       echo "<BR>";
13.       echo $var3 * $var4; //Imprime 0
14.   ?>
```

5.5.1 Conversión de Cadenas a Valores Numéricos

Cuando se hacen conversiones de cadenas a valores numéricos, la cadena se evalúa como un punto flotante si contiene números y alguno de los caracteres '.', 'e' ó 'E' (exponente). En caso de que contenga sólo números, se evalúa como un entero.

El valor numérico se determina por la porción inicial de la cadena. Si la cadena comienza con datos de numéricos, este será el valor usado. En caso contrario, el valor será 0 (cero).

Los datos numéricos válidos son el signo + ó – (opcional), seguido por uno o más dígitos (puede contener un punto decimal), seguido por un exponente (opcional). El exponente es una 'e' o una 'E' seguida por uno o más dígitos.

Con la función de PHP **`gettype($variable)`** se puede conocer el tipo de dato, que almacena una variable en un momento dado.

Con la función de PHP **`var_dump($variable)`** además de mostrar el tipo de dato de la variable, muestra el valor que contiene.

Observe un ejemplo de conversión de tipos.

Ejemplo 2.4

En este script PHP, se le asigna valores de diferentes tipos a una serie de variables. Luego se hacen distintas operaciones con estas variables y se imprime el resultado. La función **`gettype()`** muestra el tipo de dato de las variables y de los resultados obtenidos.

El código PHP comienza aquí...

```
1.    <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
2.    <HTML>
3.    <HEAD><TITLE> Conversión de Tipos en PHP - Ejemplo 2.4
      </TITLE> </HEAD>
4.    <BODY>
5.    <H1> Conversión de Tipos en PHP</H1>
6.    <HR>
```

```
7.  <?php
8.          $cadena1="250";
9.          $cadena2="solo letras";
10.         $cadena3="10.075e2 y letras";
11.         $flotante=2.5;
12.         $entero=20;
13. echo "El tipo de \$cadena1 es: ",gettype($cadena1),"<BR>\n";
14. echo "El tipo de \$cadena2 es: ",gettype($cadena2),"<BR>\n";
15. echo "El tipo de \$cadena3 es:
    ",gettype($cadena3),"<BR><BR>\n";
16. echo "\$cadena1*\$cadena2 => ", $cadena1*$cadena2, " / Tipo:
    ", gettype($cadena1*$cadena2), "<BR>\n";
17. echo "\$cadena1+\$cadena2 => ",$cadena1+$cadena2," / Tipo:
    ", gettype($cadena1+$cadena2), "<BR>\n";
18. echo "\$cadena1+\$cadena3 => ", $cadena1+$cadena3," / Tipo:
    ", gettype($cadena1+$cadena3), "<BR>\n";
19. echo "\$cadena2+\$cadena3 => ", $cadena2+$cadena3," / Tipo:
    ", gettype($cadena2+$cadena3), "<BR>\n";
20. echo "\$flotante+\$entero => ", $flotante+$entero," / Tipo:
    ", gettype($flotante+$entero), "<BR>\n";
21. echo "\$flotante*\$cadena3 => ", $flotante+$cadena3," /
    Tipo: ", gettype($flotante+$cadena3), "<BR>\n";
22. echo "\$entero*\$cadena1 => ", $entero*$cadena1," / Tipo: ",
    gettype($entero*$cadena1), "<BR>\n";
23. echo "\$flotante*\$cadena2 => ", $flotante*$cadena2," /
    Tipo: ", gettype($flotante+$cadena2), "<BR>\n";
24. ?>
25. </BODY>
26. </HTML>
```

El código PHP termina aquí

El resultado de ejecutar el script `ejemplo-2.4.php`, se muestra en la Figura 2.2.

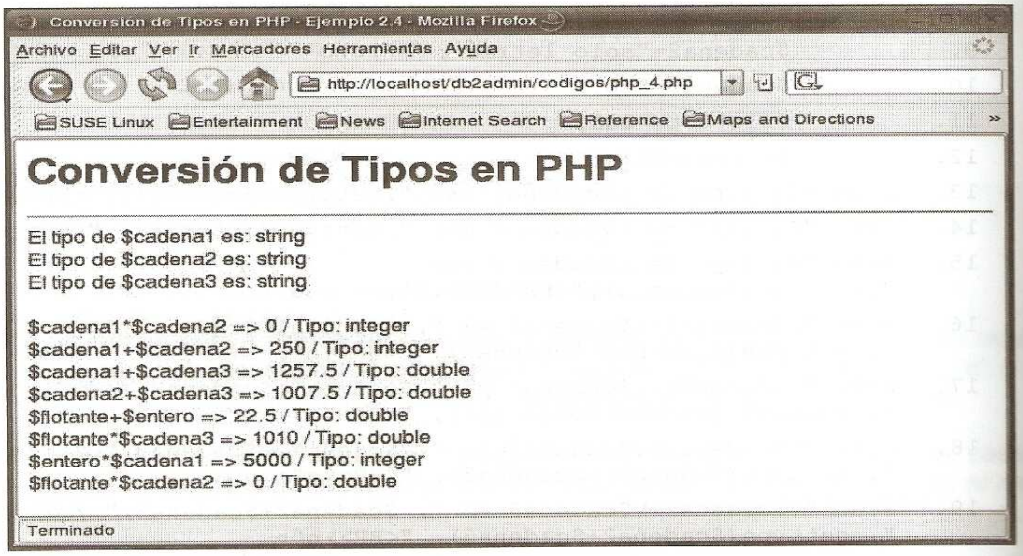


Figura 2.2. Resultado de la ejecución de `ejemplo-2.4.php`

Fin del Ejemplo 2.4

5.5.2 Casting de Variables

Otra forma de hacer conversión de tipos es hacer 'casting' o 'moldear' a una variable. Para hacer el '*casting*' de variables en PHP el nombre del tipo deseado se escribe entre paréntesis antes de la variable a la que se pretende que adquiera un tipo específico.

```
$entero = 10; // $entero es un entero
$flotante = (double)$entero; //$flotante es un punto
flotante
$cadena = (string)"Hola!"; //$cadena es un string
```

Los '*casting*' de tipos permitidos se muestran en la Tabla 2.2.

Tipo	Conversión
`(int), (integer)`	Convierte a entero (integer)
`(real), (double), (float)`	Convierte a doble (double)
`(string)`	Convierte a cadena (string)
`(boolean)`	Convierte a booleano (bool)

(array)	Convierte a array (array)
(object)	Convierte a objeto (object)

Tabla 2.2: Casting de Tipos

PHP también ofrece un conjunto de funciones que permiten probar si una variable es de cierto tipo. Estas funciones se listan en la Tabla 2.3

Función	Descripción
is_array($var)	Retorna true si el argumento es un arreglo
is_bool($var)	Retorna true si el argumento es un booleano
is_double($var)	Retorna true si el argumento es un punto flotante
is_object($var)	Retorna true si el argumento es un objeto
is_string($var)	Retorna true si el argumento es una cadena
is_null($var)	Retorna true si el argumento es nulo

Tabla 2.3: Funciones para Probar el Tipo de Dato

6. Operadores

Los operadores son símbolos reservados del lenguaje de programación para indicar que se va a realizar alguna operación aritmética, lógica, de comparación, de incremento, decremento o combinaciones de las anteriores.

6.1 Operador de Asignación

Como ya se ha visto en los ejemplos mostrados, el operador de asignación en PHP es el signo de igualdad '='. Cuando se coloca `$var1 = $var2`, se le está asignando el valor de `$var2` a `$var1`.

6.2 Operador de Concatenación

El operador de concatenación en PHP es el punto '.'. Sirve para unir dos valores, no para sumarlos. Por ejemplo:

```
1.  <?php
2.  $var1=25; $var2=30;
3.  echo $var1.$var2; //Imprime 2530, no 55.
4.  $var3="Hola ";
5.  echo $var3. "Mundo PHP!"; //Muestra 'Hola Mundo PHP!'
```

```
6.    ?>
```

6.3 Operadores Aritméticos

Se utilizan para realizar operaciones matemáticas. Los operadores aritméticos de PHP, se muestran en la Tabla 2.4.

Operador	Descripción	Ejemplo
+	Adición o suma. Suma de $a más $b	`$c = $a + $b`
-	Substracción o resta. Diferencia entre $a y $b	`$c = $a - $b`
*	Multiplicación. Producto de $a por $b	`$c = $a * $b`
/	División. Cociente de $a entre $b	`$c = $a / $b`
%	Resto o Módulo. Resto de $a entre $b	`$c = $a % $b`

Tabla 2.4: Operadores Aritméticos

6.4 Operadores de Comparación

Se utilizan para comparar dos valores. Los valores pueden ser de cualquier tipo. Un uso común de estos operadores es establecer una condición en las estructuras de control, como ciclos y condicionales. Los operadores de comparación de PHP se muestran en la Tabla 2.5.

Operador	Descripción	Ejemplo
==	Igual. Se evalúa `TRUE` si $a es igual a $b	`$a == $b`
===	Idéntico. Se evalúa `TRUE` si $a es igual a $b y son del mismo tipo	`$a === $b`
!=	Diferente. Se evalúa `TRUE` si $a es distinto a $b	`$a != $b`
<>	Diferente. Se evalúa `TRUE` si $a es distinto a $b	`$a <> $b`
!==	No Idéntico. Se evalúa `TRUE` si $a no es igual a $b o no son del mismo tipo	`$a !== $b`
>	Mayor que. Se evalúa `TRUE` si $a es mayor que $b	`$a > $b`
<	Menor que. Se evalúa `TRUE` si $a es menor que $b	`$a < $b`
>=	Mayor o igual que. Se evalúa `TRUE` si $a es mayor o igual que $b	`$a >= $b`
<=	Menor o igual que. Se evalúa `TRUE` si $a es menor o igual que $b	`$a <= $b`

Tabla 2.5: Operadores de Comparación

6.5 Operadores Lógicos

Se utilizan para evaluar una, dos o más expresiones, de acuerdo a la veracidad o falsedad de esas expresiones, se evalúa `true` o `false`. La Tabla 2.6 muestra los operadores lógicos de PHP.

Operador	Descripción	Ejemplo
&&	'Y' lógico. Se evalúa TRUE si tanto $a como $b son TRUE.	$a && $b
and	'Y' lógico. Se evalúa TRUE si tanto $a como $b son TRUE.	$a and $b
||	'O' lógico. Se evalúa TRUE al menos una de las dos, $a o $b, es TRUE.	$a || $b
or	'O' lógico. Se evalúa TRUE al menos una de las dos, $a o $b, es TRUE.	$a or $b
!	NO. Se evalúa TRUE si $a es FALSE. Niega el valor de $a.	!$a
xor	'O' exclusivo. Se evalúa TRUE si alguna de las dos, $a o $b, es TRUE, pero no ambas.	$a xor $b

Tabla 2.6: Operadores Lógicos

Los operadores lógicos se pueden combinar en expresiones complejas con los operadores de comparación. Básicamente, se emplean para establecer las condiciones en estructuras de control, como ciclos y condicionales. Por ejemplo:

```
$var1=100; $var2=300; $var3=400;
(($var1>$var2) || ($var3>$var2)) //  Retorna TRUE.
(($var1>$var2) && ($var3>$var2)) // Retorna FALSE
```

El uso de paréntesis (()) ayuda a establecer la precedencia de los operadores, es decir, agrupando expresiones complejas se puede determinar en qué orden se evaluarán las sub-expresiones que la componen.

6.6 Operadores de Incremento / Decremento

Se utilizan para aumentar o disminuir en 1 el valor de una variable. La Tabla 2.7 muestra los operadores de incremento y decremento de PHP.

Operador	Descripción	Ejemplo
++$var	Pre-incremento. Aumenta el valor $a en 1 y luego retorna ese valor.	$var1=5; $var2=++$var1; //$var2=6 y $var1=6
$var++	Post-incremento. Retorna el valor de $a y luego aumenta su valor en 1.	$var1=5; $var2=$var1++; //$var2=5 y $var1=6
--$var	Pre-decremento. Reduce el valor de $a en 1 y luego retorna ese valor.	$var1=5; $var2=--$var1; //$var2=4 y $var1=4
$var--	Post-decremento. Retorna el valor de $a y luego reduce su valor en 1.	$var1=5; $var2=$var1--; //$var2=5 y $var1=4

Tabla 2.7: Operadores de Incremento y Decremento

6.7 Operadores Combinados

Se utilizan para operar el valor de una variable con otro valor y dejarlo en la misma variable. La Tabla 2.8 muestra los operadores de combinados de PHP.

Operador	Descripción	Ejemplo

+=	Suma y asigna	`$var1=20;`
		`$var1+=10;`
		`//$var1 = 30`

-=	Resta y asigna	`$var1=20;`
		`$var1-=10;`
		`//$var1 = 10`

*=	Multiplica y asigna	`$var1=20;`
		`$var1*=10;`
		`//$var1=200`

/=	Divide y asigna	`$var1=20;`
		`$var1/=10;`
		`//$var1=2`

%=	Calcula el módulo y asigna	`$var1=20;`
		`$var1%=10;`
		`//$var1=0`

.=	Concatena y asigna	`$var1="Hola ";`
		`$var1.= "Mundo!";`
		`//$var1= 'Hola Mundo!'`

Tabla 2.8: Operadores Combinados

7. Estructuras de Control

Cuando se está programando, es necesario en muchas ocasiones la repetición de acciones sucesivas o la elección de una determinada secuencia, y no de otra, dependiendo de las condiciones específicas de la ejecución. Un ejemplo, puede ser un script que ejecute un conjunto de acciones diferentes en función a la fecha, el mes o el día de la semana actual.

Este tipo de acciones pueden ser llevadas a cabo gracias a un conjunto de estructuras de control presentes en la mayoría de los lenguajes. Aquí se describirán algunas estructuras de control de PHP que resultan de evidente utilidad.

7.1 Condicional if

La construcción `if` permite la ejecución condicional de un conjunto de instrucciones. Si se cumple la condición se ejecutan las instrucciones, en caso contrario se continúa con el resto del código que está fuera del bloque if. La sintaxis es la siguiente:

```
if (expr)
    sentencia;
```

o también:

```
if (expr) {
    sentencia1;
    sentencia2;
        ...
  }
```

Cuando se ejecuta una sola sentencia dentro del bloque `if`, no es necesario colocar los caracteres de inicio y fin de bloque, es decir, las llaves ({ }).

7.2 Condicional if ... else

La construcción `if...else` permite la ejecución condicional de un conjunto de instrucciones. Si se cumple la condición se ejecutan las instrucciones que están dentro del bloque `if`, en caso contrario se ejecutan las instrucciones que están dentro del bloque `else`. La sintaxis es la siguiente:

```
if (expr)
    sentenciaX;
else
        sentenciaY;
```

o también:

```
if (expr) {
    sentenciaX1;
    sentenciaX2;
    ...
  } else {
    sentenciaY1;
    sentenciaY2;
    ...
  }
```

7.3 Condicional if ... elseif

En la construcción `if...elseif`, cuando la condición del bloque `if`, se evalúa como `false`, se verifica la siguiente condición que se encuentra en el bloque `elseif`. Si tampoco se cumple se comprueba el siguiente bloque `elseif`, y así sucesivamente. Las instrucciones en el bloque `else` se ejecutan sólo si ninguna de las condiciones en el bloque `if` o los bloques `elseif` se cumplen. La sintaxis es la siguiente:

```
if (expr1)
        sentencias1;
  } elseif (expr2) {
```

```
    sentencias;
} elseif (expr3) {
      sentencias;
} ...
    else {
   sentencias;
  }
```

Observe el Ejemplo 2.5, donde se hace uso del condicional `if...elseif`.

Ejemplo 2.5

Este script PHP muestra el uso del condicional `if ... elseif`. Imprime un mensaje distinto dependiendo del idioma del navegador.

El código PHP comienza aquí...

```
1.  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
2.  <HTML>
3.  <HEAD>
4.  <TITLE>Condicional if... elseif - Ejemplo 2.5 </TITLE>
5.  </HEAD>
6.  <BODY>
7.  <H1>Detector de Idioma</H1>
8.  <HR>
9.  <?
10. $espanol="Hola Mundo!";
11. $ingles="Hello World!";
12. $frances="Bonjour Monde!";
13. //Leer el idioma del navegador
14. $idioma = substr($_SERVER[ 'HTTP_ACCEPT_LANGUAGE'],0,2);
15. if ($idioma == "es")
16. {
17. print "<H2>".$espanol."</H2>";
18. }
19. elseif ($idioma == "fr")
20. {
21. print "<H2>".$frances."</H2>";
22. }
23. else
24. {
25. print "<H2>".$ingles."</H2>";
```

```
26.  } ?>
27.  </BODY>
28.  </HTML>
```

El código PHP termina aquí

La salida del script `ejemplo-2.5.php` se muestra en la Figura 2.3.

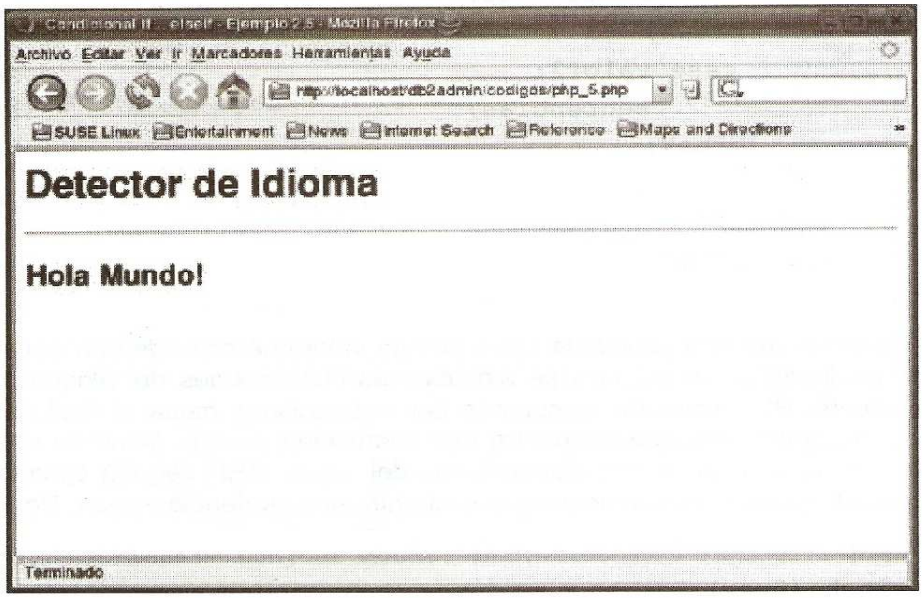


Figura 2.3: Resultado de la Ejecución de `ejemplo-2.5.php`

Se define una variable (`$idioma`) y, haciendo uso de la función **`substr()`**, se obtienen las dos primeras letras del código correspondiente al idioma aceptado por el navegador, que se almacena en la variable predefinida del servidor `$_SERVER['HTTP_ACCEPT_LANGUAGE']`. Con el condicional `if...elseif` se comprueba el idioma del navegador, español (es), francés (fr) o cualquier otro idioma que no sea ninguno de estos y dependiendo del caso se imprime un mensaje.

Nota: `$_SERVER` es una variable predefinida en PHP y representa las variables del servidor. `$_SERVER` es un arreglo asociativo que contiene información tal como cabeceras, rutas y ubicaciones de scripts.

Fin del Ejemplo 2.5

7.4 Sentencia switch

La sentencia `switch` es muy parecida a la sentencia `if...elseif`, pero compara una misma variable (o expresión) con varios valores diferentes y ejecuta un conjunto de instrucciones distinto dependiendo del valor de la variable. La sintaxis es la siguiente:

```
switch (expr) {
        case valor1:
             sentencias;
        break;
        case valor2:
             sentencias;
        break;
    ...
    default:
    sentencias;
}
```

Cuando se encuentra una sentencia `case` con un valor que coincide con el valor de la expresión evaluada en el `switch` se ejecutan las instrucciones del bloque del `case` correspondiente. PHP continúa ejecutando las instrucciones hasta el final del bloque `switch` o la primera vez que encuentre una instrucción `break`. Si no se coloca una sentencia `break` al final de las instrucciones del `case`, PHP seguirá ejecutando las sentencias del siguiente case o hasta que encuentre una sentencia `break`. Por ejemplo:

```
<?php
switch ($var) {
    case 'A':
        print "var es igual a A";
    case 'B':
        print "var es igual a B";
    case 'C':
        print "var es igual a C";
} ?>
```

Si `$var` es igual a 'A', se imprimirán los tres mensajes. Si `$var` es igual a 'B' se imprimirán los dos últimos mensajes y si es igual a 'C' se imprimirá el último mensaje.

Las sentencias en la sección `default`, se ejecutan sólo si no hubo coincidencia en ninguna de las sentencias `case`.

El valor del `case` puede ser cualquier expresión que se evalúe como un tipo simple, es decir, entero, punto flotante y cadena de texto. No se pueden usar en el `case` valores que representen arreglos ni objetos.

Ahora observe el mismo ejemplo que se presentó anteriormente pero haciendo uso de la sentencia `switch`.

Ejemplo 2.6

Este script PHP muestra el uso de la sentencia `switch`. Imprime un mensaje distinto dependiendo del idioma del navegador.

El código PHP comienza aquí...

```
1.  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
2.  <HTML>
3.  <HEAD>
4.  <TITLE>Sentencia switch - Ejemplo 2.6 </TITLE></HEAD>
5.  <BODY>
6.  <H1>Detector de Idioma</H1>
7.  <HR>
8.  <?
9.  $espanol="Hola Mundo!";
10. $ingles="Hello World!";
11. $frances="Bonjour Monde!";
12. //Leemos del navegador cuál es su idioma
13. $idioma = substr($_SERVER[ 'HTTP_ACCEPT_LANGUAGE'],0,2);
14.   switch ($idioma){
15. case "es":
16. print "<H2>".$espanol."</H2>";
17. break;
18. case  "fr":
19. print "<H2>".$frances."</H2>";
20. break;
21. default:
22. print "<H2>".$ingles."</H2>";
23.   }
24. ?>
25. </BODY>
26. </HTML>
```

El código PHP termina aquí

El script `ejemplo-2.6.php` da el mismo resultado que el script `ejemplo-2.5.php`. El resultado se puede observar en la Figura 2.3.

Fin del Ejemplo 2.6

Los ciclos son utilizados para ejecutar un código en forma repetitiva hasta que se cumpla o se mantenga una condición. Es por ello, que los programas pueden aprovecharse de este principio para realizar una determinada secuencia de instrucciones un cierto número de veces.

7.5 Ciclo while

Es el ciclo más utilizado y el más sencillo. Se utiliza para ejecutar las instrucciones contenidas en su interior siempre y cuando la condición definida sea verdadera. La sintaxis es la siguiente:

```
while (expr) {
    sentencias;
}
```

7.6 Ciclo do... while

Este tipo de ciclo no difiere mucho del anterior. La diferencia con respecto al ciclo `while` es que el `do...while` evalúa la condición al final, es decir, que aunque la condición sea `false` desde el principio, las instrucciones que están en su interior se ejecutan al menos una vez. El bucle `while` primero evalúa la condición y después hace el procesamiento, en cambio el ciclo `do...while` primero hace el procesamiento y después evalúa la condición. La sintaxis es la siguiente:

```
do {
   sentencias;
} while (expr);
```

7.7 Ciclo for

El ciclo `for`, como para los casos anteriores, se encarga de ejecutar las instrucciones que se encuentren en el bloque del ciclo (entre llaves). La diferencia con los anteriores radica en cómo se plantea la condición del ciclo. La sintaxis es la siguiente:

```
for (expr_inicial;expr_evaluación;expr_cierre) {
   sentencias;
}
```

La expresión `expr_inicial` se ejecuta incondicionalmente una sola vez cuando se ingresa en el ciclo.

Al inicio de cada iteración, se evalúa `expr_evaluación` . Si se evalúa como `true`, el bucle continúa y las sentencias anidadas se ejecutan. Si se evalúa como `false`, la ejecución del ciclo finaliza.

Al final de cada iteración, se ejecuta la expresión `expr_cierre`.

Observe el siguiente segmento de código:

```
for ($i = 1; $i <= 10; $i++) {
     print $i;
} //se imprimen los números del 1 al 10.
```

Cada una de las expresiones del ciclo `for (expr_inicial; expr_evaluación; expr_cierre)`, puede estar vacía, pero queda de parte del programador controlar la ejecución del ciclo con otras sentencias, para que no se quede iterando indefinidamente.

Ahora considere el Ejemplo 2.7 donde se muestra el uso de los ciclos `while`, `do...while` y `for`.

Ejemplo 2.7

El siguiente script PHP muestra el uso de los ciclos `while`, `do...while` y `for`. Genera tres menús de selección que contiene días, meses y años respectivamente.

El código PHP comienza aquí...

```
1.  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
2.  <HTML>
3.  <HEAD>
4.  <TITLE>Ciclos while, do..while y for - Ejemplo
    2.7</TITLE></HEAD>
5.  <BODY>
6.  <H1>Selección de día, mes y año... </H1>
7.  <HR>
8.  <FORM name="f1">
9.  <?php
10. $dia=date("d"); // * Retorna el día actual
11. print("<H2>Seleccione un día del mes:</H2>\n");
12.
13. // * Se crea el menú desplegable de los días del mes
14. echo("<SELECT name=\"dia\">\n\t<OPTION>Elija
    uno...</OPTION>\n");
15.
16. while ($dia<= 31)     // * Se usa un ciclo while
```

```
17.  {    print("\t<OPTION value=\"$dia\">$dia</OPTION>\n");
18.     $dia++;
19.  }
20.  echo("</SELECT >\n");
21.
22.  $mes=date("m"); // * Retorna el mes actual, en número.
23.  print("<H2>Seleccione un mes del año:</H2>\n");
24.
25.  // * Se crea el menú desplegable de los meses del año
26.  echo("<SELECT name=\"mes\">\n\t<OPTION>Elija
     uno...</OPTION>\n");
27.
28.  do // * Se usa un ciclo do...while
29.  {
30.     print("\t<OPTION value=\"$mes\">$mes</OPTION>\n");
31.     $mes++;
32.  } while ($mes<= 12);
33.  echo("</SELECT >\n");
34.
35.  $año=date("Y"); // * Retorna el año actual
36.  print("<H2>Seleccione un año:</H2>\n");
37.
38.  // * Se crea el menú desplegable de los años
39.  echo("<SELECT name=\"an\">\n\t<OPTION>Elija
     uno...</OPTION>\n");
40.  $finAño = $año + 30;
41.  for ($año; $año<= $finAño; $año++) // * Se usa un ciclo for
42.  {
43.     print("\t<OPTION value=\"$año\">$año</OPTION>\n");
44.     $año++;
45.  }
46.  echo("</SELECT >\n");
47.  ?>
48.  </FORM>
49.  </BODY>
50.  </HTML>
```

El código PHP termina aquí

La salida del script `ejemplo-2.7.php` se muestra en la Figura 2.4.

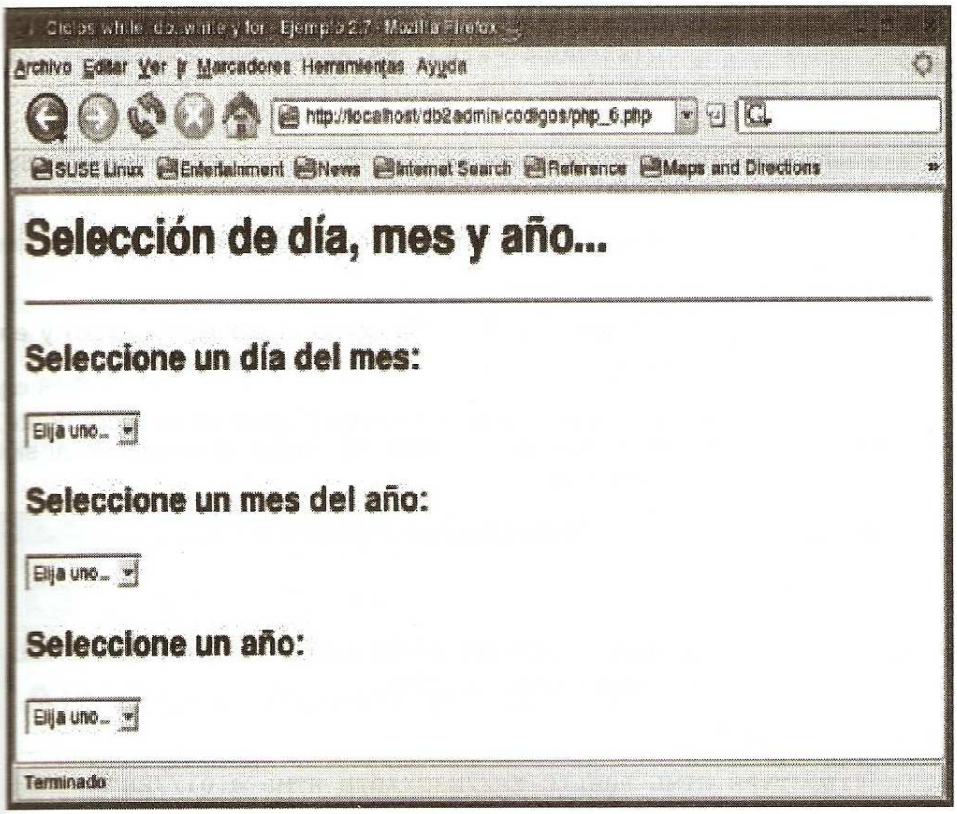


Figura 2.4: Resultado de la Ejecución de `ejemplo-2.7.php`

En el ejemplo, la función de PHP **`date()`**, se utiliza para retornar una fecha completa, ya sea el día, mes o año, de acuerdo al parámetro recibido. La opción `"d"` indica que se obtendrá el día actual en formato numérico. La opción `"m"` indica que se obtendrá el mes actual en formato numérico. La opción `"Y"` indica que se obtendrá el año actual en formato de cuatro cifras.

Fin del Ejemplo 2.7

7.8 Ciclo foreach

Este ciclo está implementado en las versiones de PHP a partir de la versión 4. Ayuda a recorrer los valores de un arreglo, lo cual puede resultar muy útil para efectuar un acceso rápido al mismo. `foreach` funciona solamente con arreglos y generará un error si se intenta utilizar con otro tipo de datos o variables no inicializadas. La sintaxis es la siguiente:

```
foreach($arreglo as $valor) {
        sentencias;
```

```
}
```

o también:

```
foreach($arreglo as $clave => $valor) {
        sentencias;
}
```

En la primera opción se toma cada uno de los elementos del arreglo y en cada iteración se deja en la variable `$valor`.

En la segunda opción, se toma tanto el elemento como la clave (o índice) y en cada iteración se dejan en las variables `$valor` y `$clave` respectivamente.

Nótese que no se requiere especificar una condición de parada ni un incremento, pues el ciclo se detendrá cuando se finalice el recorrido del arreglo y el acceso al siguiente elemento se hace de forma automática.

Observe el uso del ciclo `foreach` con un ejemplo.

Ejemplo 2.8

El siguiente script PHP muestra el uso del ciclo `foreach`. Genera una tabla que contiene algunos países y sus respectivas monedas.

El código PHP comienza aquí...

```
1.  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
2.  <HTML>
3.  <HEAD>
4.  <TITLE>Ciclo foreach - Ejemplo 2.8</TITLE>
5.  </HEAD>
6.  <BODY>
7.  <H1>Algunas monedas del mundo...</H1>
8.  <HR><BR>
9.  <?
10. //Se crea un arreglo asociativo que contiene las monedas
11. $monedas=array("Europa"=>"Euro",
12.     "USA"=>"Dolar",
13.     "Japón"=>"Yen",
14.     "Venezuela"=>"Bolívar",
15.     "México"=>"Peso");
16. //Se crea la tabla donde se muestra la información
17. echo "<TABLE bordercolor=blue border=1 align=center>\n",
18. "<TR>\n","<TD>PAIS</TD><TD>MONEDA</TD>\n","</TR>\n";
```

```
19.
20.   foreach ($monedas as $pais=>$moneda)
21.   {
22.   //Se agrega una nueva fila por cada moneda en el arreglo
23.   echo "<TR>\n","<TD>$pais</TD><TD>$moneda</TD>\n","</TR>\n";
24.   }
25.   echo "</TABLE>";
26.   ?>
27.   </BODY>
28.   </HTML>
```

El código PHP termina aquí

El resultado de la ejecución del script `ejemplo-2.8.php` se muestra en la Figura 2.5.

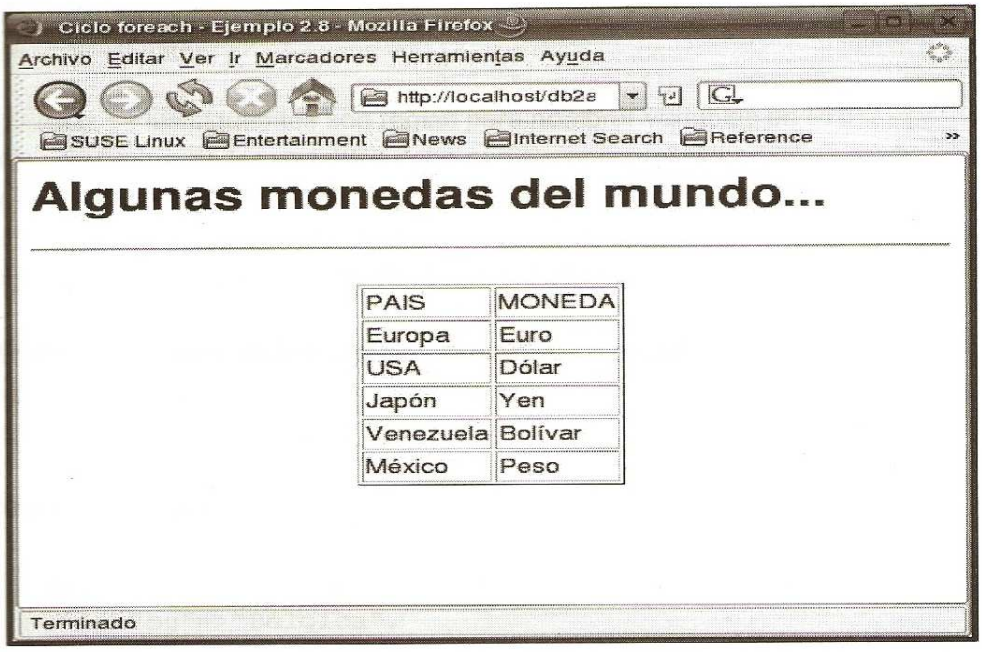


Figura 2.5: Resultado de la Ejecución de `ejemplo-2.8.php`

Ya se han revisado la mayoría de las estructuras de control que ofrece PHP. En la próxima unidad se estudiará un concepto fundamental para la programación: *Las Funciones*.

Resumen

Ahora que ha completado esta unidad, usted debe ser capaz de:

- Explicar la sintaxis básica de PHP.
- Describir el uso de variables y constantes.
- Describir el uso de los operadores disponibles en PHP.
- Indicar los diferentes tipos de datos que posee PHP.
- Discutir las estructuras de control de PHP.

Unidad 2: Examen de Autoevaluación

1) Utilizar la siguiente sintaxis: `<script language="php"> ... </script>`, es una forma incorrecta de delimitar los bloques de código PHP
   a) Verdadero
   b) Falso

2) ¿Cuáles de las siguientes sentencias son ciertas?
   a) Las variables y constantes definidas por el usuario son sensibles a mayúsculas y minúsculas
   b) Las variables y constantes predefinidas de PHP no son sensibles a mayúsculas y minúsculas
   c) Las funciones definidas por el usuario son sensibles a mayúsculas y minúsculas
   d) Las funciones incorporadas de PHP no son sensibles a mayúsculas y minúsculas

3) ¿Cuáles de las siguientes son reglas para nombrar variables en PHP?
   a) El nombre de las variables debe comenzar con el signo '@'
   b) El nombre de las variables puede contener espacios en blanco
   c) El nombre de las variables deben comenzar con el signo '$'
   d) Luego del prefijo de variable '$', el nombre debe comenzar con una letra

4) En PHP se pueden definir arreglos asociativos o hashes
   a) Verdadero
   b) Falso

5) Al ejecutar el siguiente código PHP, ¿Qué se obtiene como resultado?

```
<?php
$paises = array("al"=>"Alemania",
      "bg"=>"Bélgica",
      "ca"=>"Canadá",
      "pr"=>"Puerto Rico",
      "ve"=>"Venezuela"
      );
?>
```

   a) Se crea un arreglo asociativo bidimensional de 5 filas y 5 columnas
   b) Se crea un arreglo asociativo unidimensional de 5 posiciones

c) Se crea un arreglo asociativo bidimensional de 5 filas y 2 columnas
d) Ocurre un error porque los arreglos no se pueden crear de esa forma

6) ¿Cómo se pueden acceder las variables globales definidas en un script dentro de una función?
   a) Se pueden acceder directamente, colocando simplemente el nombre de la variable
   b) Se deben declarar explícitamente como globales dentro de la función, utilizando la cláusula **`global`**, para poder utilizarlas
   c) Se pueden acceder utilizando la variable predefinida de PHP **`$GLOBALS`**
   d) Se pueden acceder utilizando la variable predefinida de PHP **`$_SERVER`**

7) ¿Cuáles de las siguientes son estructuras de control en PHP?
   a) Switch($var) ... case("valor")
   b) if (condicionX)... elseif (condicionY)
   c) do ... while (condicionX)
   d) foreach($arreglo as $valor)

8) Al ejecutar el siguiente código PHP, ¿Qué se imprime en pantalla?

```
1.  <?php
2.  $var='B'
3.  switch ($var) {
4.      case 'A':
5.          print "Ud. eligió la opción A";
6.      case 'B':
7.          print "Ud. eligió la opción B";
8.      case 'C':
9.          print "Ud. eligió la opción C";
10. } ?>
```

   a) Ud. eligió la opción B
      Ud. eligió la opción C

   b) Ud. eligió la opción B

   c) Ud. eligió la opción A
      Ud. eligió la opción B
      Ud eligió la opción C

d) No se imprime nada

9) ¿Cuál de las siguientes es la forma correcta de declarar una constante en PHP?

a) $PI = 3.1416
b) PI = 3.1416
c) define("PI", 3.1416)
d) $var = defined("PI")

10) La función `gettype($variable)` permite conocer el tipo de dato, que almacena una variable en un momento dado, además muestra el valor que contiene la variable.

a) Verdadero
b) Falso

Unidad 2: Respuestas al Examen de Autoevaluación

1) b
2) a y d
3) c y d
4) a
5) b
6) b y c
7) a, b, c y d
8) a
9) c
10) b

Unidad 3: Funciones y Extensiones PHP

Objetivos de Aprendizaje

Al final de esta unidad, usted será capaz de:

- Explicar y desarrollar las funciones definidas por el usuario.
- Describir y utilizar las funciones del lenguaje incorporadas en PHP.
- Indicar cómo se incorporan las extensiones en PHP.

1. Introducción

En la *Unidad 2 – Elementos del Lenguaje PHP*, se presentaron las bases para desarrollar programas usando PHP. Se discutieron conceptos tales como variables, constantes y se revisaron la mayoría de las estructuras de control, condicionales y ciclos. En la presente unidad se estudiará un concepto fundamental para la programación: *Las Funciones*.

Las Funciones son un conjunto de instrucciones escritas en un lenguaje de programación que permite realizar varias operaciones con datos suministrados en forma de *argumentos* a la función. Una función agrupa un conjunto de instrucciones, que pueden ser utilizadas en cualquier punto del programa, sólo invocándolas a través del nombre de la función. En PHP existe una gran división de las funciones que son:

- Funciones definidas por el usuario.
- Funciones propias de PHP.

2. Funciones Definidas por el Usuario

Las funciones definidas por el usuario son aquellas desarrolladas por el programador y quedan como parte de su aplicación. Las funciones se definen dentro del código PHP. La siguiente es la sintaxis para crear una función:

```
function miFuncion ($arg_1, $arg_2, ..., $arg_n)
{
    sentencias;
    return valorDeRetorno;
}
```

Las funciones son invocadas dentro del código PHP colocando el nombre y entre paréntesis los argumentos que recibe. Si no recibe argumentos se colocan paréntesis vacíos.

```
miFuncion ($valor_1, $valor_2, ..., $valor_n);
miFuncion();
```

Cuando se invoca una función, puede ser asignada a una variable.

```
$resultado = miFuncion();
```

En PHP3 las funciones deben definirse antes de ser referenciadas. A partir de PHP4 no es necesario hacer esto, excepto cuando una función es definida condicionalmente.

2.1 Funciones Condicionales

Cuando una función es definida condicionalmente su definición debe ser procesada antes que sea llamada. Observe el siguiente segmento de código:

```
1.  <?php
2.  $condicion = true;
3.  /*Aquí todavía no se puede llamar a funcionCondicional()
    porque aún no existe. Pero podemos llamar a la función
    ultima(), aunque esté definida al final del script. */
4.  ultima();
5.  if ($condicion) {
6.    function funcionCondicional() {
7.      echo "Si la condición se cumple, se define la función!";
8.    }
9.  }
10. /* Ahora podemos llamar a funcionCondicional() sólo si
    $condicion se cumple */
11. if ($condicion)
12. funcioncondicional();
13. /* Definición de la función ultima()*/
14. function ultima()
15. { echo "Esta función existe desde el inicio del programa!";
    }
16. ?>
```

Las funciones condicionales sólo se pueden utilizar si la condición se cumple, porque sólo en ese caso la función existe. En caso contrario no se puede invocar porque la función no existe.

2.2 Funciones Anidadas

En PHP se pueden declarar funciones dentro de funciones. Observe el siguiente segmento de código.

```
1.  <?php
2.  function primera() {
3.          function segunda()  {
4.                  echo "No existo hasta que primera() sea
    llamada";
5.          }
6.  }
7.  /* Aquí todavía no podemos invocar a segunda()*/
8.  primera();
9.  /*  Ahora si podemos invocar a segunda()*/
10. segunda();
11. ?>
```

Una función anidada no puede ser llamada sino hasta que se invoque primero la función padre, es decir, la función dentro de la cual ha sido declarada.

Los nombres de las funciones no son sensibles a mayúsculas y minúsculas.

2.3 Retorno de Funciones

Una función retorna valores usando la sentencia **return**. Se puede retornar cualquier tipo de valor, incluyendo arreglos y objetos. Observe el siguiente ejemplo:

```
1.  <?php
2.  function listaAutos()
3.  {
4.  // * Se retorna un arreglo
5.  return array ("Volskwagen","BMW", "Mercedes Benz");
6.  }
7.  list ($auto1, $auto2, $auto3) = listaAutos();
8.  ?>
```

2.4 Paso de Parámetros a las Funciones

El comportamiento de los argumentos que reciben las funciones varía de acuerdo a cómo sean pasados. En PHP se pueden pasar parámetros por valor, por referencia y por defecto.

2.4.1 Parámetros por Valor

Por defecto, los parámetros de una función se pasan por valor, es decir, si el valor del parámetro se modifica dentro de la función, no cambia fuera de ella.

```
1.  <?php
2.  function mostrar($param) {
3.      $param = 'Valor modificado';
4.      echo $param;     // Imprime: 'Valor modificado'
5.  }
6.  $param = 'Valor actual';
7.  mostrar($param); // En la llamada, se imprime 'Valor
    modificado'
8.  echo $param;     // Imprime: 'Valor actual'
?>
```

2.4.2 Parámetros por Referencia

Cuando un parámetro se pasa por referencia, si el valor del parámetro cambia dentro de la función, este cambio se ve reflejado fuera de ella. Para pasar un parámetro por referencia se antepone un ampersand (&) al nombre del parámetro en la definición de la función.

```
1.  <?php
```

```
2.  function agregar(&$parRef) {
3.      $parRef .= 'modificado!';
4.  }
5.  $var = 'Valor inicial ';
6.  agregar($var);
7.  echo $var;    // Muestra: 'Valor inicial modificado!'
8.  ?>
```

2.4.3 Parámetros por Defecto

En este caso la función recibe un valor por defecto si en la llamada a la función no se especifica ninguno. El valor por defecto debe ser constante, no una variable. Se debe colocar de último en la secuencia de parámetros que no sean definidos por defecto.

```
1.  <?php
2.  function sumar30($num1, $num2 = 30){
3.      return "La suma es: ".($num1 + $num2);
4.  }
5.  echo sumar30(20);    // Muestra: 50
6.  echo sumar30(50,50);    // Muestra: 100
7.  ?>
```

2.4.4 Número de Parámetros Variables

A partir de PHP4 las funciones pueden recibir un número variable de parámetros en las funciones definidas por los usuarios. Existen funciones incorporadas en PHP que permiten conocer cuántos parámetros han sido pasados a la función y obtener cada uno de ellos.

- **`func_num_args()`:** Retorna el número de parámetros pasados a la función definida por el usuario. `func_num_args()` generará un 'warning' si es llamada fuera de la definición de la función.

```
1.  <?php
2.  function miFuncion() {
3.     $numargs = func_num_args();
4.     echo "Número de parámetros: $numargs\n";
5.  }
6.  miFuncion(10,30,20);  // Muestra: 'Número de parámetros: 3'
7.  ?>
```

- **`func_get_arg(pos)`**: Retorna el parámetro que está en la posición '`pos`' en la lista de parámetros de una función definida por el usuario. Los parámetros de la función se cuentan comenzando por la posición cero (0). Generará un 'warning' si es llamada fuera de la definición de la función.

```
1.  <?php
```

```
2.    function miFuncion() {
3.       $numargs = func_num_args();
4.       if ($numargs >= 2) {
5.          echo "El 2do parámetro: ".func_get_arg(1);
6.       }
7.    }
8.    miFuncion(10,20,30); //Muestra 'El 2do parámetro es: 20'
9.    ?>
```

- **`func_get_args()`**: Devuelve un arreglo en el cual cada elemento corresponde a un elemento de la lista de parámetros de la función. Generará un 'warning' si es llamada fuera de la definición de la función.

```
1.    <?php
2.    function miFuncion() {
3.       $numargs = func_num_args();
4.       $parametros = func_get_args();
5.       for ($i = 0; $i < $numargs; $i++ ) {
6.          echo "Parámetro $i: ".$parametros[ $i] ."<br>\n";
7.       }
8.    }
9.    miFuncion(10, 20, 30); /* Muestra    Parámetro 1: 10
10.                                  Parámetro 2: 20
11.                                  Parámetro 3: 30 */
12.   ?>
```

3. Funciones Definidas por el Lenguaje

PHP tiene incorporadas muchas funciones en su núcleo. También existen funciones definidas en extensiones específicas de PHP y no funcionarán si la extensión no está cargada en PHP. Por ejemplo, para usar la función `mysql_connect()` se necesita tener la extensión de soporte para MySQL e incorporarla en PHP. Por otra parte, existen nuevas funciones en el núcleo de PHP que se incluyen con cada versión de PHP que se libera.

Usando las funciones **`phpinfo()`** o **`get_loaded_extensions()`** se puede conocer qué extensiones están cargadas en PHP. Muchas extensiones se encuentran activadas por defecto y otras requieren ser activadas explícitamente.

Para conocer si una función está definida se puede utilizar la función incorporada de PHP `function_exists(nombreFuncion)`.

```
1.    <?php
2.    if (function_exists(miFuncion))
3.    print "La función está definida!";
```

```
4.  else
5.  print "La función no está definida!";
6.  ?>
```

Para conocer cuales funciones posee una extensión cargada en PHP, se puede utilizar la función **`get_extension_funcs(nombre_modulo)`**, que retorna el nombre de todas las funciones que están cargadas en un módulo de extensión.

```
1.  <?php
2.  //Imprime el nombre de las funciones del módulo de XML
3.  print_r(get_extension_funcs("xml"));
4.  //Imprime el nombre de las funciones del módulo de mySQL
5.  print_r(get_extension_funcs("mysql"));
6.  ?>
```

Por la gran cantidad de funciones que posee PHP, no se hará referencia a todas ellas, sino a las más representativas. Para una información detallada sobre la sintaxis y el uso de las funciones PHP, consulte el Manual de referencia de PHP, disponible en: *http://www.php.net/docs.php*.

A continuación se presentan algunas funciones para trabajar con cadenas.

3.1 Funciones de Cadena

Permiten la manipulación de cadenas. Algunas de estas funciones ya han sido utilizadas en algunos de los ejemplos: `print()` y `echo()`, que permiten escribir una cadena de salida.

3.1.1 Función `printf (formato[,argumentos])`

Esta función permite dar formato a uno o más valores que se especifiquen como argumentos. La cadena de salida se puede formatear como un número entero, punto flotante (indicando el número de decimales que se quiere mostrar), caracteres ASCII, etc. Los distintos formatos se indican en la Tabla 3.1.

Símbolo	Descripción
%	Carácter que permite la definición del formato a utilizar.
b	El argumento es tratado como un número entero, presenta el resultado como un número binario.
c	El argumento es tratado como un número entero, presenta el resultado como el valor del carácter ASCII.
d	El argumento es tratado como un número entero, presenta el resultado como un número decimal con signo.

u	El argumento es tratado como un número entero, presenta el resultado como un número decimal sin signo.
f	El argumento es tratado como un número real (double), presenta el resultado como número punto flotante.
o	El argumento es tratado como un número entero, presenta el resultado como un número en base octal.
s	El argumento es tratado como un string (cadena), presenta el resultado como un string(cadena).
x	El argumento es tratado como un número entero, presenta el resultado como un número en base hexadecimal (con letras minúsculas).
X	El argumento es tratado como un número entero, presenta el resultado como un número en base hexadecimal (con letras mayúsculas).

Tabla 3.1: Tabla de Caracteres para Formatos de Impresión

3.1.2 Función `sprintf (formato[,argumentos])`

Funciona como la función `printf()`, con la diferencia que `printf()` imprime el resultado, mientras que `sprintf()` sólo cambia el formato de la variable y debe guardarse el valor en una nueva variable para poder utilizarla.

Observe un ejemplo donde se emplea `printf()` y `sprintf()`.

Ejemplo 3.1

El siguiente script PHP muestra el uso de las funciones `printf y sprintf`. Genera una tabla que contiene algunos países y sus respectivas monedas.

El código PHP comienza aquí...

```
1.  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
2.  <HTML>
3.  <HEAD>
4.  <TITLE>Funciones printf() y sprintf() - Ejemplo 3.1 </TITLE>
5.  </HEAD>
6.  <BODY>
7.  <H1>Funciones de Formato de Cadenas</H1>
8.  <HR>
9.  <?php
10. $pago = 1200.1666667;
11. print("<B>Valor de la variable \$pago sin formato
    $pago</B><BR>\n");
```

```
12.
13.  print("<BR><H2>Función printf()</H2>\n");
14.  print("<B> printf(\"%01.3f\", \$pago) imprime: </B>");
15.  printf("<B>%01.3f</B><BR>",$pago);
16.  print("<B> printf(\"%06d\", \$pago) imprime: </B>");
17.  printf("<B>%06d</B> <BR>",$pago);
18.
19.  print("<BR><BR><H2>Función sprintf()</H2>");
20.  $pago1=sprintf("%01.3f",$pago);
21.  print("<B> sprintf(\"%01.3f\", \$pago) cambia a:
     $pago1<BR></B>");
22.  $pago2=sprintf("%06d",$pago);
23.  print("<B> sprintf(\"%06d\", \$pago) cambia a: $pago2</B>");
24.  ?>
25.  </BODY>
26.  </HTML>
```

El código PHP termina aquí

El resultado de la ejecución del script `ejemplo-3.1.php` se muestra en la Figura 3.1.

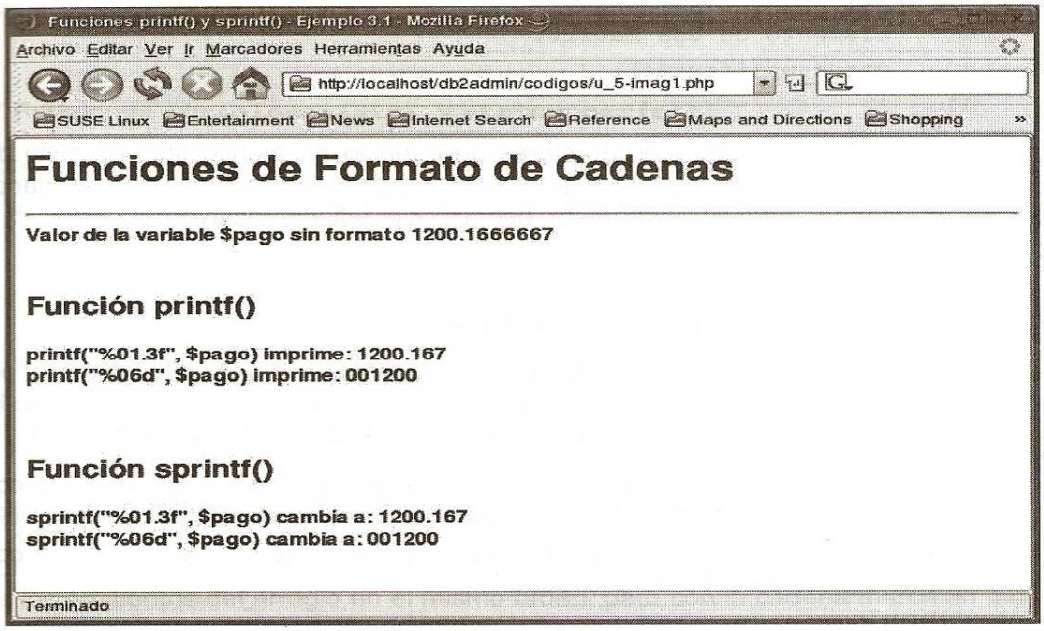


Figura 3.1: Resultado de la Ejecución de `ejemplo-3.1.php`

En el uso de `printf()` y `sprintf()` con el formato `%1.3f`, se indica que el número se formatea como un punto flotante con tres dígitos de precisión. El formato `%06d` tratará al número como un entero sin signo de seis dígitos.

Fin del Ejemplo 3.1

3.1.3 Funciones `trim(str)`, `ltrim(str)`, `rtrim(str)` y `chop(str)`

La función **`trim()`** elimina los caracteres de espacio en blanco al inicio y al final de una cadena de caracteres.

```
1.  <?
2.  function limpia_valores(&$valor)
3.  {
4.    $valor=trim($valor);
5.  }
6.  $frutas = array('   manzana','platano   ', '   arandano
    ');
7.  var_dump($frutas);  //Función que muestra el tipo de dato de
    la variable y el valor que contiene.
8.  echo "<br>";
9.  array_walk($frutas, 'limpia_valores');  //array_walk aplica
    una función del usuario, en este caso limpia_valores, a cada
    elemento de una matriz.
10. var_dump($frutas);
11. ?>
```

La función **`ltrim()`** elimina los caracteres de espacio en blanco sólo al inicio de la cadena. La función `rtrim()` y `chop()` elimina los caracteres de espacio en blanco al final de la cadena. Los caracteres de espacio en blanco incluyen: `"\n"`, `"\r"`, `"\t"`, `"\v"` y `"\0"`.

3.1.4 Función `substr(str,comienzo,cantidad)`

Retorna parte o un substring de la cadena `str`. El parámetro `comienzo`, indica el punto de partida y `cantidad` indica el número de caracteres que se desean obtener. Existen algunas condiciones que se deben tomar en cuenta cuando se utiliza esta función:

- Si el carácter de partida es un número positivo, la función empezará a contar desde la parte izquierda de la cadena.
- Si el carácter de partida es un número negativo, la función empezará a contar desde la parte derecha de la cadena.

Observe el siguiente segmento de código:

```
1.  <?
2.  $subcadena1 = substr("Caracas", 0, 3);
```

```
3.  /* Inicia en la posición 0 a partir de la izquierda de la
    cadena y se toman 3 caracteres */
4.  print $subcadena1. "<BR>"; // Imprime 'Car'
5.  $subcadena2 = substr("Caracas", -4, 4);
6.  /* Inicia en la cuarta posición a partir de la derecha de la
    cadena  y se toman 4 caracteres */
7.  print $subcadena2."<BR>"; // Imprime 'acas'
8.  ?>
```

3.1.5 Función substr_count(cadena, subcadena)

Retorna el número de veces que subcadena aparece en cadena.

```
1.  <?
2.  print substr_count("Tres tristes tigres", "es"); //Imprime 3
3.  ?>
```

3.1.6 Función strlen(str)

Devuelve la longitud de la cadena de caracteres.

```
1.  <?
2.  $longitud = strlen("Metropolis");
3.  print $longitud; // Imprime 10
4.  ?>
```

3.1.7 Funciones strtolower(str) y strtoupper(str)

La función `strtolower(str)`, devuelve la cadena `str` en minúsculas y la función `strtoupper(str)`, retorna la cadena `str` en mayúsculas.

```
1.  <?
2.  $minúsculas = strtolower("VeNEzuEla")
3.  $mayúsculas = strtoupper("VeNEzuEla")
4.  print $minúsculas; // Imprime 'venezuela'
5.  echo "<BR>";
6.  print $mayúsculas; // Imprime 'VENEZUELA'
7.  ?>
```

3.1.8 Funciones implode(marca,arreglo) y explode(marca,cadena,lim)

La función `implode()` devuelve una cadena que contiene una representación de todos los elementos del arreglo en el mismo orden, pero con la cadena marca en medio de los mismos.

```
1.  <?
2.  $miArreglo = array("carro","moto","patín");
```

```
3.  $miCadena = implode ("*", $miArreglo);
4.  print $miCadena; // Imprime 'carro*moto*patín'
5.  ?>
```

La función `explode()` devuelve un arreglo de cadenas, cada una de las cuales es una subcadena de la cadena original. Esas subcadenas están delimitadas dentro de la subcadena con una cadena que funciona como delimitador o marca. Si se especifica limite, el arreglo contendrá un máximo de limite elementos con el último conteniendo el resto de la cadena.

```
1.  <?
2.  $miCadena = "manzana-naranja-pera-limón";
3.  $miArreglo = explode("-", $miCadena);
4.  /* Se crea un arreglo de 4 elementos */
5.  print $miArreglo[ 1] ; // Imprime 'naranja'
6.  ?>
```

3.1.9 Funciones `htmlentities(str)` y `urlencode(str)`

La función `htmlentities(str)` codifica los caracteres de la cadena str en caracteres HTML estándar. Por ejemplo, los caracteres '<' y '>' son codificados como < y > respectivamente. El uso de esta función permite que todos los caracteres que tengan una entidad equivalente en HTML sean cambiados a esas entidades.

```
1.  <?
2.  $cadenaHTML = htmlentities("<La niña se cayó>");
3.  echo $cadenaHTML;
4.  /* $cadenaHTML = '&lt;La ni&ntilde;a se cay&oacute;&gt;'
    sino se utiliza la función htmlentities(), la cadena
    debe escribirse de esta manera para poder
    visualizarse.*/
5.  ?>
```

La ventaja de utilizar esta función es que algunos caracteres reservados en HTML podrán escribirse literalmente, sin necesidad de usar sus respectivas Referencias de Entidad.

La función `urlencode(str)` permite codificar los caracteres de la cadena str en caracteres aceptados en la cadena de una dirección URL.

```
1.  <?
2.  $nombre=urlencode("María Pérez");
3.  // $nombre = Mar%EDa+P%E9rez
4.  echo "<BR>";
5.  $fecha_nac=urlencode("10/12/1980");
6.  // $fecha_nac = 10%2F12%2F1980
7.  ?>
```

3.2 Funciones de Fecha y Hora

Estas funciones permiten obtener la fecha y hora del servidor en el que se están ejecutando los scripts PHP. Se pueden usar estas funciones para dar formato a las fechas y horas en muchas maneras diferentes.

3.2.1 Función `date(formato[,timestamp])`

Devuelve una cadena formateada de acuerdo con el `formato` especificado, utilizando la marca de tiempo `timestamp` indicada o la hora local actual si no hay parámetro. En la Tabla 3.2 se describen algunos de los formatos posibles.

Formato	Descripción	Ejemplo
a	am o pm en minúscula	7:00 am
A	am o pm en mayúscula	7:00 PM
d	Día del mes, numérico de dos dígitos	01, 02 ... 09
D	Nombre del día en idioma ingles, tres letras	Mon... Sun
F	Nombre completo del mes en idioma ingles	January... December
h	Hora en el formato de 12 horas	01 ... 12
H	Hora en el formato de 24 horas	00 ... 23
i	Minutos en dos dígitos	00 ... 59
j	Día del mes sin cero inicial	1, 2 ... 9
l	Nombre completo del día de la semana en inglés	Monday ... Sunday
m	Número del mes de dos dígitos, con 0 inicial	01, 02 ... 09
M	Nombre abreviado del mes en ingles	Jan, Feb ... Dec
S	Sufijo ordinal del día del mes	st, nd, rd, etc. 1st,2nd,3rd
U	Número de segundos transcurridos desde el 1 de enero de 1970, a las 00:00:00 horas	45896321
y	Año en el formato de dos dígitos	98,99,00,01, etc
Y	Año en el formato de cuatro dígitos	1998, 1999, 2000, 2001, etc.
z	Día del año como el número de días a partir del primero de enero del año	190,192, ..., 365 o 366 si es año bisiesto

Tabla 3.2: Formatos de la Función date()

Observe el siguiente segmento de código:

```
1.  <?
2.  $fechaHoy = date("M j, Y");
3.  print("La fecha de hoy es: $fechaHoy");
4.  ?>
```

El resultado es: `'La fecha de hoy es = Mar 22, 2007'`

3.2.2 Función `time()`

Devuelve una marca de tiempo (timestamp UNIX), que es el número de segundos transcurridos desde el 1 de enero de 1970 a las 00:00:00 horas.

```
1.  <?
2.  $timeStamp = time();
3.  print("El tiempo transcurrido en segundos es: $timeStamp");
4.  ?>
```

3.3 Funciones Matemáticas

Estas funciones manipulan valores dentro del rango definido para los tipos integer y float del servidor (en la actualidad. estos tipos corresponden con los tipos long y double de C).

3.3.1 Función `round(num,precision)`

Permite redondear los valores numéricos a la precisión solicitada, teniendo en cuenta las siguientes condiciones:

- Si la parte decimal es inferior a 0.50 se redondea al entero inmediato inferior.
- Sí la parte decimal es superior a 0.50 se redondea al entero inmediato superior.

Para visualizar el efecto revise el siguiente código:

```
1.  <?
2.  $pago = round(1200000.1666667);
3.  $bono = round(30000.58);
4.  print("El pago es:  $pago");
5.  echo "<BR>";
6.  print("El bono es: $bono");
7.  ?>
```

Cuando se ejecute en el servidor el resultado debe ser el siguiente:

```
El pago es: 1200000
El pago es: 30001
```

Pero si se desea considerar los decimales, después del punto se debe especificar con el parámetro `precision`.

```
1.  <?
2.  $pago = round(1200000.1332 , 1);
3.  $bono = round(30000.58 , 1);
4.  print("El pago es: $pago");
5.  echo "<BR>";
6.  print("El bono es: $bono");
```

```
7.  ?>
```

El resultado será el siguiente:

```
El pago es: 1200000.1
El pago es: 30000.6
```

La función `round()` se divide en dos funciones: `ceil()` que redondea la parte entera al entero inmediato superior y la función `floor()` que redondea la parte decimal al entero inmediato inferior.

3.3.2 Función `abs(valor)`

Permite hallar el valor absoluto de un número.

```
1.  <?
2.  $positivo = 25;
3.  $negativo = -25;
4.  $pos = abs($positivo);
5.  $neg = abs($negativo);
6.  print("Valor absoluto de \$positivo: $pos");
7.  echo "<BR>";
8.  print("Valor absoluto de \$negativo: $neg");
9.  ?>
```

El resultado es el siguiente:

```
Valor absoluto de $positivo:  25
Valor absoluto de $negativo: 25
```

3.3.3 Funciones `rand(lim_min,lim_max)` y `srand(expr)`

La función `rand()` devuelve un valor pseudoaleatorio que se encuentre entre los valores enteros límite mínimo (`lim_min`) y límite máximo (`lim_max`).

La función `srand()` posee como entrada una expresión `expr` que proporciona la semilla para la generación de un número pseudo-aleatorio. La expresión más comúnmente utilizada es un tiempo a nivel de microsegundos, utilizando la siguiente función: `(double) microtime() * 1000000`.

Estas funciones permiten la generación de números aleatorios. Se debe ejecutar primero la función `srand()` y luego la función `rand()`, para garantizar una buena aleatoriedad.

```
1.  <?
2.  srand((double) microtime() * 1000000);
3.  $numeroaleatorio = rand ();
```

```
4.  print("\$numeroaleatorio = $numeroaleatorio");
5.  ?>
```

3.4 Funciones de Arreglos

Estas funciones permiten trabajar y manipular arreglos unidimensionales y multidimensionales, de diferentes maneras. Son útiles porque existen funciones específicas de manejo de bases de datos que le asignan valores a arreglos, con el resultado devuelto por alguna consulta a la base de datos y muchas otras funciones devuelven arreglos como resultado.

3.4.1 Funciones `asort(arr)`, `arsort(arr)`, `ksort(arr)`, `sort(arr)`, `rsort(arr)`

Estas funciones permiten ordenar los elementos de un arreglo, de distintas formas.

- La función **`asort()`** ordena un arreglo ascendentemente de modo que los índices mantengan su correlación con los elementos a los que están asociados. Ordena a partir de los valores.

Se utiliza principalmente para arreglos asociativos, pero también se puede utilizar con arreglos indexados. Observe un ejemplo donde se hace uso de la función `asort()`.

Ejemplo 3.2

El siguiente script imprime los valores de un arreglo tal como están almacenados y luego los imprime una vez ordenados utilizando la función **`asort()`**.

El código PHP comienza aquí...

```
1.  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
2.  <HTML>
3.  <HEAD>
4.  <TITLE>Funciones de Arreglos - Ejemplo 3.2 </TITLE>
5.  </HEAD>
6.  <BODY bgcolor="#FFFFFF">
7.  <?
8.  /* Se declara un arreglo con valores */
9.  $carros = array ("d"=>"BMW", "b"=>"Mercedes Benz",
    "a"=>"Volskwagen", "c"=>"Lexus");
10. print "<H2>Antes de ordenar el arreglo:</H2>\n";
11. for (reset ($carros); $clave = key($carros); next($carros))
    {
12.     echo $carros[ $clave] ."<BR>\n";
13. }
14. ?>
```

```
15.  <HR>
16.  <?
17.  print "<H2>Después de ordenar el arreglo:</H2>\n";
18.  asort ($carros);
19.  for (reset ($carros); $clave = key($carros); next($carros))
     {
20.      echo $carros[ $clave] ."<BR>\n";
21.  }
22.  ?>
23.  </BODY>
24.  </HTML>
```

El código PHP termina aquí

El resultado de la ejecución de `ejemplo-3.2.php` se muestra en la Figura 3.2.

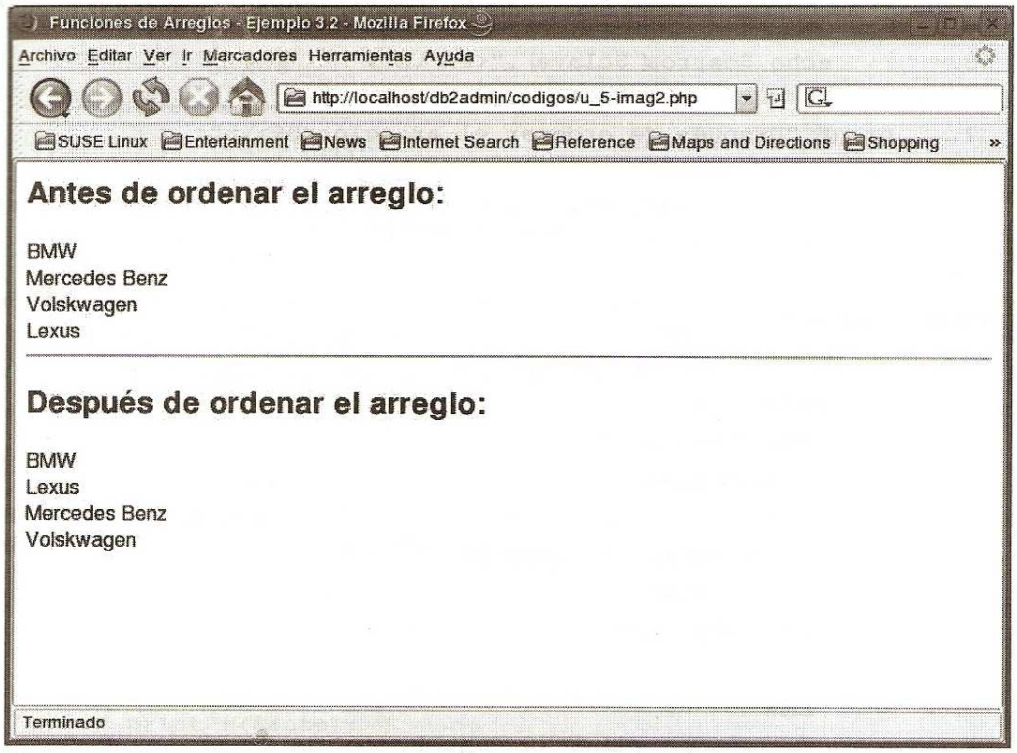


Figura 3.2: Resultado de la Ejecución de `ejemplo-3.2.php`

En el código del Ejemplo 3.2, se puede apreciar el uso de las funciones **`reset()`**, **`next()`** y **`key()`**.

- La función **`reset(arreglo)`** inicializa el índice de un arreglo (lo coloca en 0).

- La función **next(arreglo)** avanza al siguiente elemento del arreglo.
- La función **key()** retorna el elemento índice o clave de la posición actual en el arreglo.
- La función **count()** retorna el número de elementos de un arreglo.

Fin del Ejemplo 3.2

- La función **arsort()** ordena un arreglo en orden inverso, de modo que los índices mantengan su correlación con los elementos a los que están asociados. Ordena a partir de los valores. Se utiliza principalmente para arreglos asociativos, pero también se puede utilizar con arreglos indexados.

```
1.  <?
2.  $carros = array ("a"=>"BMW", "c"=>"Mercedes Benz",
    "b"=>"Volskwagen", "d"=>"Lexus");
3.  print "Antes de ordenar el arreglo:<br>\n";
4.  for (reset ($carros); $clave = key($carros); next($carros))
    {
5.      echo $carros[ $clave] ."<BR>\n";
6.  }
7.  print "Después de ordenar el arreglo:<BR>\n";
8.  arsort ($carros);
9.  for (reset ($carros); $clave = key($carros); next($carros))
    {    echo $carros[ $clave] ."<BR>\n";  }
10. ?>
```

La salida es la siguiente:

```
Antes de ordenar el arreglo:
        BMW
        Mercedes Benz
        Volskwagen
        Lexus
Después de ordenar el arreglo:
        Volskwagen
        Mercedes Benz
        Lexus
        BMW
```

- La función **ksort()** ordena un arreglo por la clave, manteniendo las correlaciones entre la clave y los datos.

```
1.  <?
2.  $carros = array ("a"=>"BMW", "c"=>"Mercedes Benz",
    "b"=>"Volskwagen", "d"=>"Lexus");
3.  print "Antes de ordenar el arreglo:<BR>\n";
```

```
4.  for (reset ($carros); $clave = key($carros); next($carros))
    {
5.      echo $carros[ $clave] ."<BR>\n";
6.  }
7.  print "Después de ordenar el arreglo:<BR>\n";
8.  ksort ($carros);
9.  for (reset ($carros); $clave = key($carros); next($carros))
    {
10.     echo $carros[ $clave] ."<BR>\n";
11. } ?>
```

La salida es la siguiente:

```
Antes de ordenar el arreglo:
        BMW
        Mercedes Benz
        Volskwagen
        Lexus
Después de ordenar el arreglo:
        Volskwagen
        Mercedes Benz
        Lexus
        BMW
```

- La función **`sort()`** ordena los elementos de un arreglo de menor a mayor. Puede ser que no se mantengan la correspondencia entre el índice y la clave.
- La función **`rsort()`** ordena los elementos de un arreglo de mayor a menor. Puede ser que no se mantengan la correspondencia entre el índice y la clave.

3.4.2 Función each()

Devuelve el par *clave/valor* actual para un arreglo y avanza el cursor al siguiente elemento. Generalmente se usa dentro de estructuras iterativas.

```
1.  <?
2.  $autos = array ("Volskwagen", "BMW", "Mercedes Benz",
    "Lexus");
3.  $auto = each($autos);
4.  print_r($auto);
5.  ?>
```

La salida es la siguiente:

```
Array ([ 1] => Volskwagen [ value] => Volskwagen [ 0] => 0 [ key] =>
0)
```

- La función **`print_r(variable)`** imprime los valores de las variables en forma legible. Si recibe como parámetro un arreglo, los valores serán presentados en un formato que muestra las claves y los elementos.

3.4.3 Función `list()`

Se utiliza para asignar una lista de variables en una sola operación.

```
1.  <?
2.  $carros = array ("a"=>"BMW", "c"=>"Mercedes Benz",
    "b"=>"Volskwagen", "d"=>"Lexus");
3.  while (list ($clave, $val) = each($carros)) {
4.      echo "$clave => $val <BR>";
5.  }
6.  ?>
```

La salida es la siguiente:

```
a => BMW
c => Mercedes Benz
b => Volskwagen
d => Lexus
```

Existe gran cantidad de funciones para trabajar con arreglos muy útiles, pero no es el propósito de este curso hacer una revisión exhaustiva de todas ellas. Para mayor información consulte el Manual de PHP, disponible en: *http://www.php.net/docs.php*.

Ahora se estudiarán algunas funciones de PHP para trabajar con archivos.

3.5 Funciones de Manejo de Archivos

PHP ofrece gran cantidad de funciones para trabajar con el sistema de archivos.

3.5.1 Función `fopen(nombrearchivo,modo)`

Permite abrir un archivo ubicado en el servidor o desde un URL (a través de los protocolos HTTP o FTP). En el argumento `nombrearchivo` se debe especificar el nombre del archivo incluyendo la ruta, el `modo` se refiere a la forma en la que se desea abrir el archivo. La Tabla 3.3 describe y presenta un ejemplo de cada modo.

Modo	Definición	Ejemplo
"a"	Abre el archivo para agregar datos al final. No sobrescribe los datos que existen. Si el archivo no existe trata de crearlo.	`fopen("arch.txt","a");`
"a+"	Abre el archivo para lectura y agregación. No sobrescribe los datos que existen. Si el archivo no existe trata de crearlo.	`fopen("arch.txt","a+");`
"r"	Abre el archivo para sólo lectura. (Esta opción debe ser utilizada si se usa el protocolo HTTP para acceder el archivo)	`fopen("arch.txt","r");`
"r+"	Abre el archivo para lectura y escritura. Los datos son agregados desde el inicio del archivo	`fopen("arch.txt","r+");`
"w"	Abre el archivo para sólo escritura. Si el archivo existe, todos los datos serán eliminados y sustituidos por los nuevos datos que se agreguen. Si el archivo no existe trata de crearlo.	`fopen("arch.txt","w");`
"w+"	Abre el archivo para lectura y escritura. Si el archivo existe, todos los datos serán eliminados y sustituidos por los nuevos datos que se agreguen. Si el archivo no existe trata de crearlo.	`fopen("arch.txt","w+");`

Tabla 3.3. Modos de Apertura de Archivos

Nota: Es importante destacar, que para trabajar con archivos, se deben incluir en la misma carpeta donde están alojados los archivos .php en el servidor Apache. De no ser así, debe especificarse la ruta absoluta de la ubicación del archivo.

3.5.2 Función `fclose(punteroArch)`

Se utiliza para cerrar el archivo que fue previamente abierto con la función fopen(). El argumento `punteroArch` se refiere al apuntador al archivo abierto que se desea cerrar.

```
1.  <?
2.  $archivo = "archivo.txt";
3.  $fp=fopen($archivo, "w");
4.  fclose($fp);
5.  ?>
```

3.5.3 Función `filesize(punteroArch)`

Devuelve el tamaño del archivo apuntado por `punteroArch` o `FALSE` en caso de error.

3.5.4 Funciones `fputs(punteroArch)` y `fwrite(punteroArch)`

Mediante estas dos funciones se pueden escribir datos en el archivo especificado incluyendo caracteres de nueva línea (\n).

```
1.  <?
2.  $archivo = fopen("archivo.txt", "a");
3.  fputs($archivo,"Linea 1\n");
4.  fwrite($archivo,"Linea 2\n");
5.  ?>
```

3.5.5 Función `fpassthru(punteroArch)`

Lee todo el contenido del archivo e imprime la salida en el navegador. Cuando se usa esta función, no es necesario usar la función `fclose()` ya que el archivo se cierra automáticamente. Además arroja el total de caracteres que posee el archivo.

```
1.  <?
2.  $archivo = fopen("archivo.txt","r");
3.  $var = fpassthru($archivo);
4.  ?>
```

3.5.6 Funciones `fgets(punteroArch,n)` y `fgetss(punteroArch,n)`

- La función **`fgets()`** lee una cadena de caracteres de tamaño `n - 1` del archivo especificado por `punteroArch`. Si la cantidad de bytes en el archivo es menor a n leerá todo el archivo. Si no se especifica `n`, `fgets()` lee una cadena hasta que encuentra el carácter de nueva línea.
- La función **`fgetss()`** es similar a la función `fgets()`, pero trata de eliminar cualquier etiqueta HTML o PHP que consiga en el archivo.

3.5.7 Función `fread(punteroArch,n)`

Permite leer el contenido de un archivo como un texto del tamaño indicado por n. La lectura acaba cuando n bytes se han leído o se alcanza EOF, lo que ocurra primero.

3.5.8 Función `feof(punteroArch)`

Esta función retorna `TRUE` si el apuntador al archivo (`punteroArch`) está en `EOF` ('Fin de Archivo') o si ocurre un error. Si no ha ocurrido error y el apuntador al archivo no está al final, devuelve `FALSE`.

A continuación se mostrará un ejemplo donde se ponen en práctica las funciones de manipulación de archivos, vistas hasta el momento.

Nota: Para trabajar con archivos en Linux, es necesario cambiar la permisología de los mismos, ya que por defecto este Sistema Operativo los crea con restricción para Grupo y Otros. Se debe hacer lo siguiente: Botón derecho sobre el archivo, se selecciona Propiedades – Permisos y se asigna:

Propietario – Lectura y escritura posibles

Grupo – Lectura y escritura posibles

Otros – Lectura y escritura posibles

Ejemplo 3.3

El siguiente script lee el contenido de un archivo y lo muestra en el navegador. Luego agrega en otro archivo nuevos datos y muestra el resultado en el navegador.

El código PHP comienza aquí...

```
1.  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
2.  <HTML>
3.  <HEAD>
4.  <TITLE>Manejo de Archivos - Ejemplo 3.3 </TITLE></HEAD>
5.  <BODY>
6.  <H1>Manejo de Archivos</H1>
7.  <HR>
8.  <?php
9.  $archivo1="archivo1.txt";
10. /* Se abre el archivo archivo1.txt para sólo lectura */
11. $fp1=fopen($archivo1,"r");
12. echo "<H2>Leyendo un archivo con la función fgets()</H2>\n";
13. /* Se lee del archivo línea por línea */
14. while (!feof($fp1)){
15. echo fgets($fp1)."<BR>\n";
16. }
17. /* Se cierra el archivo */
18. fclose($fp1);
19. echo "<BR>";
20. echo "<H2>Leyendo un archivo con la función
    fpassthru()</H2>\n";
21. echo "<B>Se agrega contenido y luego se imprimen los
    datos:</B>\n<BR><BR>\n";
22. $archivo2="archivo2.txt";
23. /* Se abre el archivo archivo2.txt para agregación */
24. $fp2=fopen($archivo2,"a");
25. fwrite($fp2,"<BR>Y agrego una mas... <BR>");
26. fclose($fp2); /* Se cierra archivo2.txt */
27. /* Ahora se abre para lectura */
28. $fp2=fopen($archivo2,"r");
29. $num=fpassthru($fp2);
30. /* No es necesario cerrar el archivo, porque fpassthru()  lo
    cierra*/
31. ?>
```

```
32.  </BODY>
33.  </HTML>
```

El código PHP termina aquí

La salida del código de `ejemplo-3.3.php`, se muestra en la Figura 3.3.

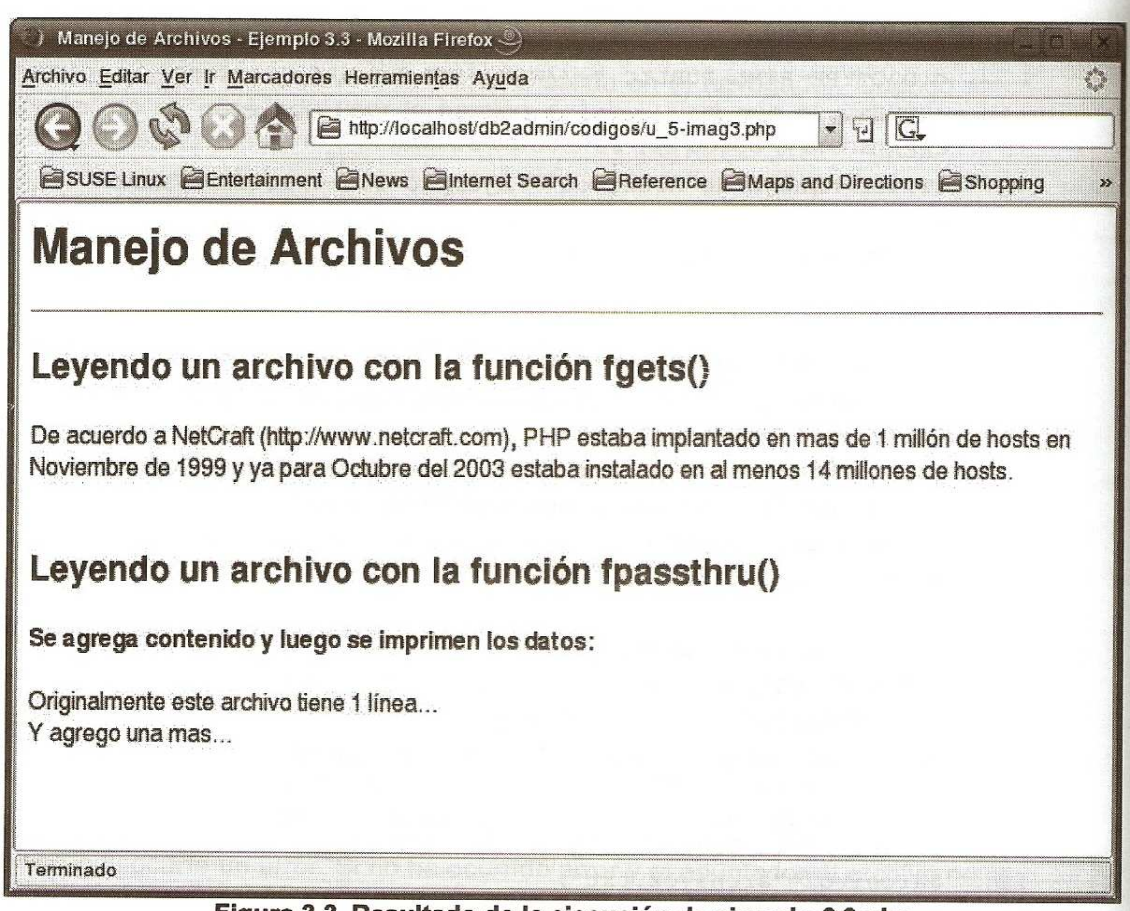


Figura 3.3. Resultado de la ejecución de ejemplo-3.3.php

Fin del Ejemplo 3.3

Se ha finalizado la discusión de los elementos fundamentales del lenguaje PHP y de algunas de las funciones incorporadas que posee. Ahora se culminará la unidad discutiendo cómo se incorporan módulos de extensión a PHP.

4. Extensiones en PHP

Los módulos externos o extensiones en PHP permiten incorporar mayor funcionalidad a los programas desarrollados. Una extensión puede ofrecer la funcionalidad de interactuar con una base de datos específica, convertir los elementos de XML en objetos y usarlos dentro del código, utilizar los objetos de Java, etc.

Una extensión específica se puede cargar de forma automática (o por defecto) o de forma dinámica.

Para incorporar una extensión de forma automática se debe modificar el archivo de configuración `php.ini`. La extensión habilitada estará disponible por defecto para todos los scripts que se ejecuten. Para habilitar una extensión por defecto, se debe agregar la directiva `'extension'`, junto con el nombre de la extensión, dentro del archivo `php.ini`. Por ejemplo:

```
extension=php_java.dll
```

Para incorporar una extensión dinámicamente se debe hacer directamente desde el script PHP, utilizando la función **`dl()`**. La extensión se carga en tiempo de ejecución y estará disponible sólo para el script actual.

```
1.  <?php
2.      dl('php_mysql.dll');
3.      print "Se cargó el módulo de extensión de MySQL";
4.  ?>
```

Además de esto se deben configurar algunos parámetros propios de la extensión. En el `php.ini` están cargadas por defecto muchas extensiones útiles.

Una vez incorporada la extensión se puede hacer uso de sus funciones, como cualquier otra función de PHP. Por ejemplo:

```
1.  <?php
2.  // * Conectarse al servidor MySQL
3.  $idConn =
    mysql_connect("localhost:3306","mysqladmin","mysqladmin");
4.  if ($idConn == 0) {
5.   echo "Falló la Conexión al Servidor MySQL!";
6.        // * Obtener el error que ha ocurrido
7.  $sqlerror = mysql_error($idConn);
8.   echo"$sqlerror";
9.  }
10.  else {
11.   echo "La Conexión a Base de Datos fue satisfactoria";
12. // * Seleccionar la BD
```

```
13.    $dbSelect = mysql_select_db('SAMPLE', $idConn);
14.   // * Cerrar la conexión
15.    mysql_close($idConn);
16.    }
17.  ?>
```

En el segmento de código anterior se agregó la extensión para MySQL de forma dinámica. Luego se utilizaron algunas de las funciones disponibles, como `mysql_connect()`, `mysql_select_db()` y `mysql_close()`.

En la Unidad 7 - *Acceso a Base de Datos usando PHP* se utilizará la extensión de PHP para trabajar con bases de datos MySQL. En la Unidad 9 – *Extensiones de PHP*, se trabajará con algunas extensiones útiles.

En las unidades siguientes se discutirán otros elementos que permitirán desarrollar aplicaciones PHP más avanzadas.

Resumen

Ahora que ha completado esta unidad, usted debe ser capaz de:

- Explicar y desarrollar las funciones definidas por el usuario.
- Describir y utilizar las funciones del lenguaje incorporadas en PHP.
- Indicar cómo se incorporan las extensiones en PHP.

Unidad 3: Examen de Autoevaluación

1) Una función condicional en PHP, se caracteriza por:
   a) Cuando se define una función condicional, su definición debe ser procesada antes que sea llamada
   b) Una función condicional sólo se pueden utilizar si la condición del bloque `if` donde está definida se cumple, porque sólo en ese caso la función existe
   c) Una función condicional se puede utilizar aun cuando la condición del bloque `if` donde está definida no se cumple
   d) En PHP no se pueden definir funciones condicionales

2) A partir de PHP4 las funciones se deben definir antes de referenciarse.
   a) Verdadero
   b) Falso

3) En PHP las funciones son sensibles a mayúsculas y minúsculas
   a) Verdadero
   b) Falso

4) ¿Cuáles de las siguientes sentencias son ciertas?
   a) En PHP no se pueden definir funciones anidadas.
   b) Una función anidada es aquella que se declara dentro del cuerpo de otra función.
   c) Una función anidada no puede ser llamada sino hasta que se invoque primero la función padre, es decir, la función dentro de la cual ha sido declarada
   d) Una función anidada se puede invocar en cualquier momento en cualquier punto del script.

5) ¿Cuáles de las siguientes son formas de pasar parámetros a funciones en PHP?
   a) Paso de parámetros por valor
   b) Paso de parámetros por referencia
   c) Paso de parámetros por defecto
   d) Sólo a y c

6) A partir de PHP4 las funciones pueden recibir un número variable de parámetros en las funciones definidas por los usuarios
   a) Verdadero
   b) Falso

7) La función **trim()** elimina todos los caracteres de espacio en blanco de una cadena de caracteres
   a) Verdadero
   b) Falso

8) Al ejecutar la siguiente función:

```
<? $subcadena = substr("PHP es fácil de aprender", -     11,11);
      print $subcadena;
?>
```

Se obtiene el siguiente resultado
   a) Se imprime: 'PHP es fácil'
   b) Se imprime: 'PHP es fácil'
   c) Se imprime: 'de aprender'
   d) Se imprime: 'P es fácil'

9) ¿Cuál de las siguientes sentencias son ciertas?
   a) La función `explode()` devuelve un arreglo de cadenas, cada una de las cuales es una subcadena de la cadena original
   b) La función `time()` retorna la hora actual del sistema
   c) La función `reset(arreglo)` inicializa el índice de un arreglo (lo coloca en 0)
   d) La función `count(arreglo)` retorna el número de elementos de un arreglo

10) ¿Qué se obtiene al ejecutar la siguiente función `fopen(infophp.txt,w+)`?
   a) Abre el archivo `infophp.txt`, para agregar datos al final. No sobrescribe los datos que estén en el archivo
   b) Abre el archivo `infophp.txt`, para sólo lectura
   c) Abre el archivo `infophp.txt`, para sólo escritura
   d) Abre el archivo `infophp.txt` para lectura y escritura. Si el archivo existe, todos los datos serán eliminados

Unidad 3: Respuestas a Examen de Autoevaluación

1) a y b
2) b
3) b
4) b y c
5) a, b y c
6) a
7) b
8) c
9) a, c y d
10) d

Unidad 4: Lab. de Elementos del Lenguaje y Funciones

Objetivos de Aprendizaje

Al final de esta unidad, usted será capaz de:

- Escribir scripts básicos usando los elementos del lenguaje PHP.
- Crear funciones definidas por el usuario.
- Utilizar las funciones incorporadas del lenguaje.
- Utilizar algunas extensiones de PHP.

Ejercicios de Laboratorio

1) Escribir tres funciones en PHP y mostrar los resultados de la ejecución de cada una.
   - La primera función debe determinar si un número dado es par o impar
   - La segunda función debe determinar si un número A es divisible por un número B. (Está función debe recibir como parámetros los números A y B).
   - La tercera función debe recibir como entrada dos cadenas de caracteres, las debe invertir y luego concatenar.
2) Escriba un programa en PHP que genere los números primos dentro de un rango de números dado. Los números generados deben ser almacenados en un arreglo asociativo. La clave para posicionar el elemento se denota como "primoi", donde i, es el i-esimo número primo encontrado.

   Ejemplo:

```
$primos[ "primo1 "] =1;  $primos[ "primo2 "] =2;
$primos[ "primo3"] =3;  $primos[ "primo4"] =5;
$primos[ "primo5"] =7;  $primos[ "primo8"] =11;  ...
```

Luego debe imprimir los números primos que se encuentren en las posiciones impares, asumiendo que `$primos[ "primo1"]` es el primer elemento y ocupa la posición cero "0". (Para imprimir este arreglo use las funciones `list()` y `each()`).

Desarrolle funciones definidas por el usuario para la solución de este ejercicio.

3) Realizar un programa en PHP que tome el contenido de un archivo `ARCH1` y de otro archivo `ARCH2` y muestre el contenido de cada archivo por separado en el navegador.

   Luego el contenido del archivo `ARCH1` debe copiarse dentro del archivo `ARCH2` (sin sobrescribirlo) y por último debe mostrar el nuevo contenido de `ARCH2`.

Unidad 5: Desarrollo de Sitios Web

Objetivos de Aprendizaje

Al final de esta unidad, usted será capaz de:

- Definir el término Aplicación Web.
- Discutir la utilidad de las variables predefinidas de PHP.
- Explicar cómo se procesan los formularios en PHP.
- Explicar la administración de sesiones en PHP.

1. Introducción

En la Unidad 2 – *Elementos del Lenguaje PHP* y en la Unidad 3 – *Funciones y Extensiones PHP*, se establecieron los fundamentos para desarrollar scripts PHP. En esta unidad se estudiarán características más avanzadas que permitirán desarrollar aplicaciones web más avanzadas que faciliten el procesamiento de valores entrada dados por el usuario. Para ello, se aprenderá a trabajar con formularios y a recuperar la información enviada dentro de un script PHP. También se establecerán las bases para trabajar con sesiones de usuario.

Antes de comenzar a discutir cómo desarrollar aplicaciones web con PHP, se definirá lo que una aplicación web representa.

2. Aplicaciones Web

Una aplicación web es una aplicación desarrollada usando tecnologías basadas en el entorno web como HTML, XML, JavaScript, PHP, ASP, CGI – Perl, Java Servlets, etc. Las aplicaciones web utilizan otras aplicaciones como navegadores, servidores web y los protocolos de Internet para poder funcionar.

Una aplicación web típicamente se conecta a otros servidores, como sistemas de bases de datos o sistemas de procesamiento de transacciones. Generalmente, tiene una arquitectura multicapas, lo cual significa que las aplicaciones son divididas en componentes.

Las funcionalidades de los componentes de una aplicación web son las siguientes:

- **Navegador Web:** Este es el componente en el lado del cliente. El navegador es una aplicación usada para solicitar páginas web, recuperarlas y mostrar a los usuarios la información solicitada ya procesada.
- **Servidor Web:** Este componente acepta las solicitudes por páginas web hechas por el cliente y las entrega. El servidor web puede aceptar también solicitudes por scripts en el lado del servidor que generen páginas web dinámicas, por ejemplo scripts PHP.
- **Recursos Externos:** Las aplicaciones web necesitan acceder a la información o solicitar servicios de más de una fuente. Un Sistema de Administración de Base de Datos Relacional (RDBMS), un sistema de procesamiento de transacciones o cualquier otro recurso, que sea externo al servidor web, cae en esta categoría.

La Figura 5.1 muestra el modelo de componentes de aplicación web.

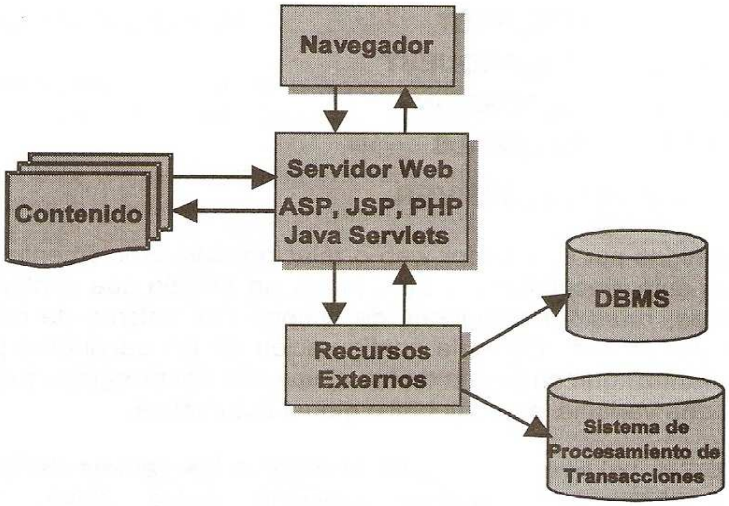


Figura 5.1: Modelo de Componentes de una Aplicación Web

Un formulario es el objeto de HTML fundamental para enviar datos por parte del cliente al servidor web. Un sitio o aplicación web que requiera un mínimo de interacción con el usuario debería utilizar formularios, pues este tiene distintos componentes que permiten ingresar datos, para luego ser procesados.

Antes de comenzar la explicación acerca del manejo de formularios en PHP, se tratarán algunas variables predefinidas en PHP, que serán de utilidad para ese propósito y para otros aspectos que se discutirán posteriormente.

Como se mencionó en la Unidad 2 – *Elementos del lenguaje PHP*, PHP ofrece un conjunto de variables predefinidas. Se considerarán las que sean de mayor relevancia para el tratamiento de los tópicos que se presentarán en esta unidad.

3. Variables Predefinidas en PHP

Las variables predefinidas en PHP, son variables que están disponibles para cualquier script que se ejecute y cada una de ellas tiene distinto propósito.

La mayoría de estas variables son *superglobales* o globales automáticas. Esto significa que están disponibles en todos los contextos a lo largo de un script.

La siguiente es una lista de algunas de las variables predefinidas en PHP.

- Variables de Servidor: **$_SERVER**
- Variables de entorno: **$_ENV**
- Cookies HTTP: **$_COOKIE**
- Variable HTTP_GET: **$_GET**

- Variable HTTP_POST: **$_POST**
- Variables de Solicitud: **$_REQUEST**
- Variables de Sesión: **$_SESSION**
- Variables Globales: **$GLOBALS**

3.1 Variables de Servidor $_SERVER

Son variables definidas por el servidor web o relacionadas directamente con el entorno donde el script se está ejecutando. `$_SERVER` es un arreglo que contiene información tal como cabeceras, rutas y ubicaciones de scripts. Los valores de este arreglo son creados por el servidor web. Como la configuración de los servidores puede cambiar entre sí, no se garantiza que cada uno de los elementos del arreglo `$_SERVER` adquiera un valor. Esta es una variable 'superglobal' o global automática.

Se utiliza `$_SERVER['elemento']`, para acceder a los valores de las variables de servidor. El nombre del elemento funciona como clave. Por ejemplo, `$_SERVER['SERVER_NAME']`, almacena el nombre del servidor. Algunos de los elementos contenidos en `$_SERVER` se listan a continuación:

- `'PHP_SELF'`: Nombre del script que se está ejecutando actualmente, relativo al directorio raíz.
- `'SERVER_NAME'`: Nombre del servidor donde se está ejecutando el script actual.
- `'REQUEST_METHOD'`: Indica cuál método de solicitud fue usado para acceder a la página, que puede ser 'GET' o 'POST'.
- `'QUERY_STRING'`: Cadena de consulta mediante la cual se accedió a la página, si existe.
- `'DOCUMENT_ROOT'`: Directorio raíz de documentos bajo el que está siendo ejecutado el script actual, tal y como está definido en el archivo de configuración del servidor.
- `'REMOTE_ADDR'`: Dirección IP del cliente que accede la página actual.
- `'REMOTE_HOST'`: Nombre del Host del usuario que accede la página actual.
- `'REMOTE_PORT'`: Puerto que es usado en la máquina en el host del cliente para comunicarse con el servidor web.
- `'SCRIPT_FILENAME'`: Ruta absoluta del nombre del script que se está ejecutando actualmente.
- `'SCRIPT_NAME'`: Ruta del script actual.

3.2 Variables de Entorno: $_ENV

Son variables que son proporcionadas al script dependiendo del entorno del computador donde se esté ejecutando el script. Esta es una variable 'superglobal' o global automática.

3.3 Cookies HTTP: $_COOKIE

Es un arreglo asociativo que contiene las variables cookies pasadas al script actual a través del encabezado HTTP. Es una variable 'superglobal' o global automática.

Más adelante en esta unidad se aprenderá cómo usar esta variable.

3.4 Variable HTTP_GET: $_GET

Es un arreglo asociativo de las variables pasadas al script actual a través del método *GET*. Esta es una variable 'superglobal' o global automática.

Más adelante en esta unidad se aprenderá cómo usar esta variable y sus funciones asociadas.

3.5 Variable HTTP_POST: $_POST

Es un arreglo asociativo de las variables pasadas al script actual a través del método POST. Es una variable 'superglobal' o global automática.

Más adelante en esta unidad se aprenderá cómo usar esta variable.

3.6 Variables de Solicitud: $_REQUEST

Son proporcionadas al script por medio de cualquier mecanismo de entrada del cliente. Es un arreglo asociativo que consiste en los contenidos de `$_GET`, `$_POST` y `$_COOKIE`. Es una variable 'superglobal' o global automática.

Más adelante en esta unidad se aprenderá cómo usar esta variable.

3.7 Variables de Sesión: $_SESSION

Son variables que han sido registradas en la sesión del script que se está ejecutando y que han sido especificadas como tal. Es un arreglo asociativo que contiene las variables de sesión disponibles en el script actual. Es una variable 'superglobal' o global automática.

Más adelante en esta unidad se aprenderá cómo usar esta variable y funciones asociadas.

3.8 Variables Globales: $GLOBALS

Es una variable 'superglobal' o global automática que contiene referencias a todas las variables globales disponibles para el script que se está ejecutando. Es un arreglo asociativo por medio del cual se puede acceder a todas las variables que están definidas en el contexto global del script. Los nombres de las variables son las claves para acceder el arreglo. Observe el ejemplo:

```
1.  <?php
```

```
2.    $glob1 = 10;
3.    $glob2 = 20;
4.    function miFuncion(){
5.    /* Accediendo a las variables globales definidas*/
6.    $GLOBALS[ "glob2"] = $GLOBALS[ "glob1"] + $GLOBALS[ "glob2"] ;
7.    }
8.    ?>
```

Se ha realizado un breve recorrido por las variables predefinidas que posee PHP. Cada una de ellas se empleará cuando sea conveniente. Pero antes se mostrará un ejemplo donde se utilice una variable de entorno.

En el siguiente ejemplo se accederá a una de las variables predefinidas de PHP, $_SERVER.

Ejemplo 5.1

A través del siguiente script, se accede a la variable predefinida $_SERVER y se muestra su contenido en el navegador.

El código PHP comienza aquí...

```
1.    <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
2.    <HTML>
3.    <HEAD><TITLE>Variables predefinidas - Ejemplo 5.1
      </TITLE></HEAD>
4.    <BODY>
5.    <?php
6.      print "<H2>Variable \$_SERVER</H2>";
7.      print "<HR>";
8.      print "<B>El contenido de \$_SERVER es: </B><BR>";
9.      foreach ($_SERVER as $clave=>$valor)
10.     {
11.      echo "$clave: $valor <BR>";
12.     }
13.   ?>
14.   </BODY>
15.   </HTML>
```

El código PHP termina aquí

El resultado de la ejecución de `ejemplo-5.1.php` se muestra en la Figura 5.2. En este ejemplo se observa parte del contenido de la variable predefinida $_SERVER.

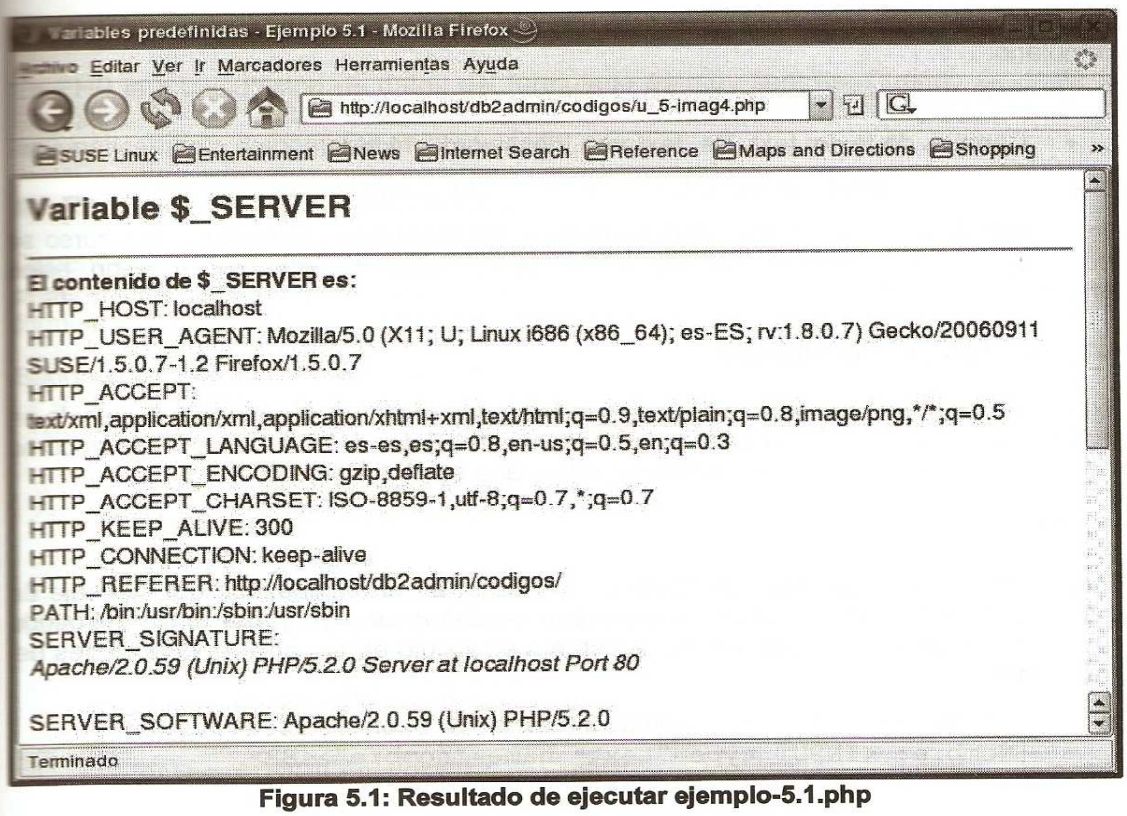


Figura 5.1: Resultado de ejecutar ejemplo-5.1.php

Fin del Ejemplo 5.1

Ahora se aprenderá a trabajar con formularios en PHP.

4. Uso de Formularios en PHP

Un formulario es un objeto perteneciente a la jerarquía DOM de HTML. Actúa como receptor de datos del cliente para posteriormente enviarlos al servidor web. Estos datos de entrada son enviados al script PHP especificado como un "action" de ese formulario.

La etiqueta `FORM` permite crear un formulario dentro del documento HTML. Una sentencia de formulario puede ser la siguiente:

```
<FORM name="Form1" method="post" action="miScript.php">

        <<Elementos del Formulario>>

</FORM>
```

A continuación se presenta una lista de los elementos que puede contener un formulario:

- Botones: button, reset y submit.
- Campos de texto: text, file, textarea, password y hidden.
- Componentes de selección: `checkbox`, `radiobutton`, `select`.

No se detallarán cada uno de estos elementos del formulario, pues para este curso se requiere conocimientos básicos de HTML y es necesario que se manejen estos términos.

4.1 Métodos del Formulario

Existen dos formas por medio de las cuales se puede enviar la información contenida en el formulario a un script que hará su procesamiento. Estas formas son conocidas como *métodos del formulario*. Los métodos del formulario son los siguientes:

- **Get**

Este método envía los datos del formulario al script en el encabezado del mensaje de solicitud HTTP (HTTP Request). En este caso los datos son agregados al URL. Cuando se usa este método una cadena de caracteres es visible en la barra de direcciones del navegador.

```
<FORM name="form1" method="GET" action="miScript.php">
Nombre: <INPUT type="text" name="nombre"><BR>
Edad: <INPUT type="text" name="edad">
<INPUT type="submit" value="enviar" name="enviar">
</FORM>
```

Cuando se envía el formulario, después del URL se agrega un signo de interrogación '?' seguido de una cadena que contiene los datos del formulario enviado. Esos serán los valores que procesará el script. Vea el siguiente ejemplo:

`http://localhost/miScript.php?nombre=Josefina+Pérez&edad=25.`

- **Post**

Este método envía los datos del formulario al script a través del cuerpo del mensaje de solicitud HTTP (HTTP Request). Los datos no son agregados al URL.

```
<FORM name="form1" method="POST" action="miScript.php">
Nombre: <INPUT type="text" name="nombre"><br>
Edad: <INPUT type="text" name="edad">
<INPUT type="submit" value="enviar" name="enviar">
</FORM>
```

Cuando se envía el formulario, no se agrega información al URL:

http://localhost/miScript.php

En PHP la manipulación de datos del formulario es muy sencilla. Cuando se envía un formulario a un script PHP, las variables de dicho formulario pasan a estar automáticamente disponibles para ese script.

4.2 Recuperar Información de un Formulario con PHP

Dependiendo de la configuración que se haya hecho en el `php.ini` y de acuerdo a preferencias personales, existen muchas maneras de acceder a los datos de los formularios HTML. Observe algunas de ellas.

Suponga que se tiene el siguiente formulario:

```
<FORM name="form1" method="POST" action="miScript.php">
Nombre: <INPUT type="text" name="nombre"><br>
Edad: <INPUT type="text" name="edad">
<INPUT type="submit" value="enviar" name="enviar">
</FORM>
```

4.2.1 Utilizando las Variables de Entorno $_POST, $_GET y $_REQUEST

- Si se usa el método 'POST' para enviar el formulario se emplea la variable `$_POST` y la variable `$_REQUEST`.

```
<?php
  print $_POST['nombre'];
  print $_POST['edad'];
  // * También puede utilizar
  $nombre = $_REQUEST['nombre'];
?>
```

- Si se usa el método 'GET' para enviar el formulario se emplea la variable `$_GET` y la variable `$_REQUEST`.

```
<?php
  $nombre = $_GET['nombre'];
  $edad = $_GET['edad'];
 // * También puede utilizar
  $nombre = $_REQUEST['nombre'];
?>
```

Opcionalmente, se puede utilizar la función `import_request_variables()` para recuperar los valores del formulario.

4.2.2 Función `import_request_variables(tipo, prefijo)`

Esta función importa variables GET, POST o COOKIE. Es útil si se ha deshabilitado la directiva `register_globals` en el `php.ini`, pero desea ver algunas variables en el contexto global.

Cuando la directiva **`register_globals`** se encuentra activa, introducirá en los scripts PHP todo tipo de variables, como por ejemplo los valores provenientes de formularios y tratará estas variables como si fuesen globales. Es altamente recomendable desactivarla, porque es muy probable que el script presente vulnerabilidades o ambigüedades.

En la función `import_request_variables(tipo,prefijo)` el parámetro `tipo` indica el tipo de variable que se desea recuperar. Puede usar los caracteres 'G', 'P' y 'C' para indicar GET, POST y COOKIE, respectivamente. Estos caracteres no son sensibles a mayúsculas o minúsculas. Cualquier otra letra diferente a 'G', 'P' o 'C' es ignorada. El parámetro `prefijo` indica el prefijo fijo que tendrá la variable. Para visualizar esta función, utilice el ejemplo anterior.

```
<FORM name="form1" method="POST" action="miScript.php">
Nombre: <INPUT type="text" name="nombre"><BR>
Edad: <INPUT type="text" name="edad">
<INPUT type="submit" value="enviar" name="enviar">
</FORM>
```

La información del formulario se recuperaría de la siguiente forma:

```
1.    <?php
2.      import_request_variables('P', 'p_');
3.      $nom = $p_nombre;
4.      /* el prefijo se completa con el nombre del campo */
5.      $ed = $p_edad;
6.     /* Si se usa el método GET
7.     import_request_variables('G', 'g_');
8.     $nom = $g_nombre;
9.   */ ?>
```

Ahora observe un ejemplo donde se recuperen valores del formulario.

Ejemplo 5.2

En el siguiente ejemplo se solicitan los datos del usuario en un formulario. Luego se muestran los datos ingresados y se le genera un id de usuario y una clave.

El código HTML comienza aquí...

```
1.   <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
2.   <HTML>
```

```
3.   <HEAD><TITLE>Uso de Formularios - Ejemplo-5.2</TITLE></HEAD>
4.   <BODY>
5.     <H2 align="center">Registro de Usuarios</H2>
6.     <HR>
7.     <H3>Regístrese y le enviamos su userid y su
     password</H3>
8.     <FORM method="post" action="ejemplo-5.2.php">
9.     Nombre: <INPUT type=text name="nombres"><BR><BR>
10.    Apellido: <INPUT type=text name="apellidos">
     <BR><BR>
11.    Email:       
12.   <INPUT type=text name="email"><BR><BR>
13.   Dirección: <TEXTAREA name="direccion"></TEXTAREA>
14.    <BR><BR>
15.   <INPUT type="submit" name="enviar" value="ENVIAR">
16.  </FORM>
17.  </BODY>
18.  </HTML>
```

El código HTML termina aquí

El código PHP comienza aquí...

```
1.   <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
2.   <HTML>
3.   <HEAD><TITLE>Uso de Formularios - Ejemplo 5.2</TITLE></HEAD>
4.   <BODY>
5.   <H2 align="center">Datos de Registro</H2>
6.   <HR>
7.   <?php
8.    function  CrearUserId($nom, $apel){
9.     $nombre=substr($nom, 0, 3);
10.    $apellido=substr($apel, 0, 3);
11.    $usuario=$nombre.$apellido;
12.    srand((double) microtime() * 1000000);
13.   // * Se selecciona un valor arbitrario
14.    $final= rand(0,1000);
15.    $usuario= $usuario.$final;
16.    print("<H3>Su USERID para ingresar a nuestro sitio
     Web es:  $usuario</H3>");
17.  }
18.
```

```
19.  function CrearPassword(){
20.    $cadena= "Cadena para crear un password aleatorio";
21.    $longitud= 8; // * Longitud de la clave
22.   /* Se realiza una función hash (dispersión) sobre la
     cadena de caracteres */
23.    $cadena=md5($cadena);
24.    $longcadena=strlen($cadena); // * longitud de la  cadena
25.  // * se genera un valor random
26.    srand((double) microtime() * 1000000);
27.  // * Se selecciona un punto inicial arbitrario
28.    $inicio= rand(0,($longcadena - $longitud - 1));
29.    $Password= substr($cadena, $inicio, $longitud);
30.    print("<H3>Su PASSWORD para ingresar a nuestro sitio   Web
     es: $Password</H3>");
31.  }
32.  ?>
33.  <?php
34.   $nombre=$_POST[ 'nombres'];
35.   $apellido=$_POST[ 'apellidos'];
36.   $email=$_POST[ 'email'];
37.   $dir=$_POST[ 'direccion'];
38.   echo "<B>Sus Nombres: </B>\n",$nombre,"<BR>\n";
39.   echo "<B>Sus Apellidos: </B>\n",$apellido,"<BR>\n";
40.   echo "<B>Su Email: </B>\n",$email,"<BR>\n";
41.   echo "<B>Su Dirección: </B>\n",$dir,"<BR>\n";
42.   echo "<HR>";
43.   CrearUserId($nombre,$apellido);
44.   CrearPassword();
45.  ?>
46.  </BODY>
47.  </HTML>
```

El código PHP termina aquí

En la Figura 5.3 se muestra el formulario HTML y en la Figura 5.4 se muestra el resultado de llamar a `ejemplo-5.2.php`, cuando se hace clic en el botón submit.

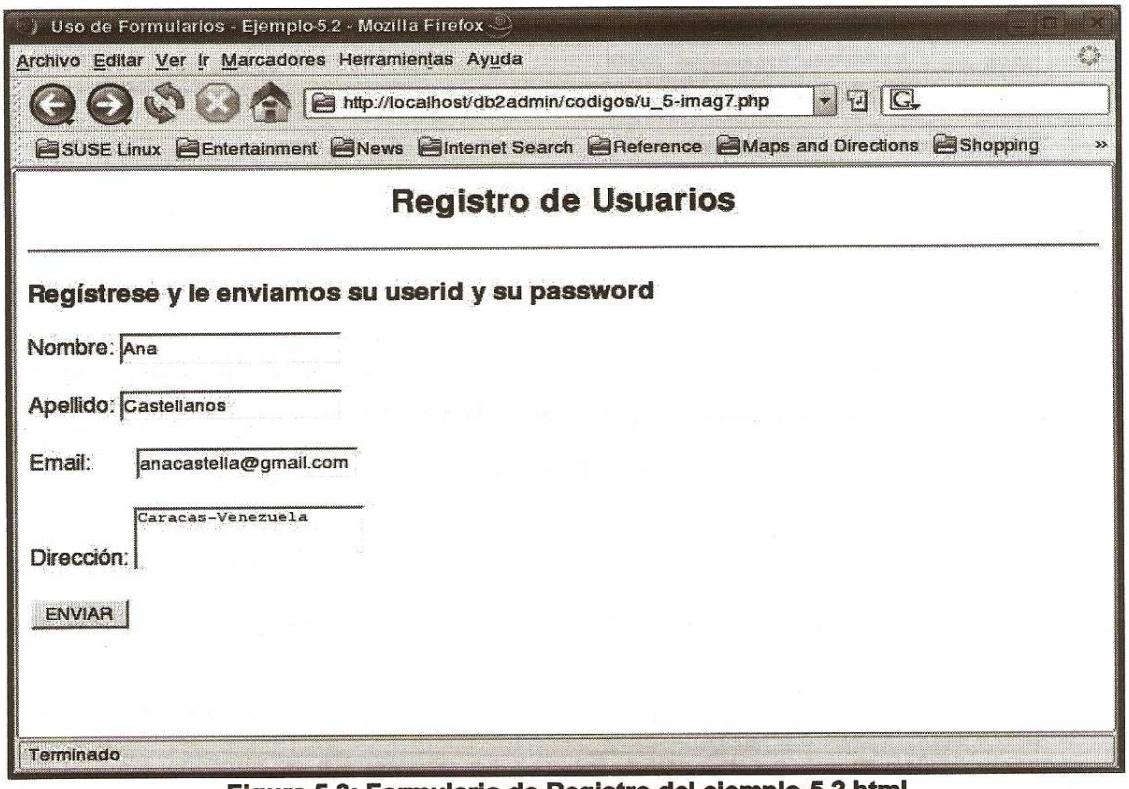


Figura 5.3: Formulario de Registro del ejemplo-5.2.html

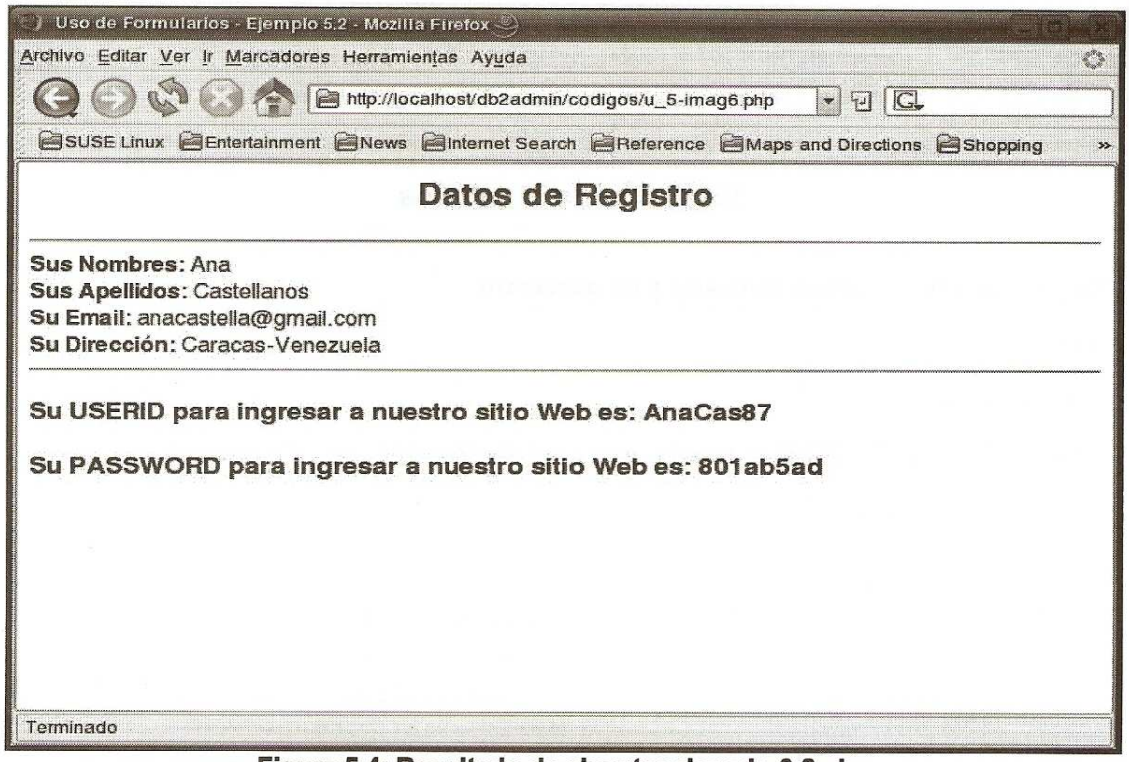


Figura 5.4: Resultado de ejecutar ejemplo-3.2.php

En el ejemplo se utilizó la función **`md5()`**, es una función de dispersión que utiliza el algoritmo MD5 para generar un valor hash.

Fin del Ejemplo 5.2

5. Administración de Sesiones

En una aplicación web pueden haber muchos clientes que acceden a la aplicación al mismo tiempo, haciendo solicitudes particulares y enviando datos propios.

Dado que HTTP no guarda estados, el servidor no puede determinar en un momento dado quién hizo una solicitud y quién envío ciertos datos. Necesita de ayuda adicional para determinar quién, exactamente, está haciendo la solicitud.

Para identificar a los usuarios, a cada cliente se le solicita que se presente a sí mismo antes de hacer la solicitud al servidor y luego se debe llevar el control de los usuarios a partir de esa identificación. Esto se conoce como *Administración de Sesión*.

Una sesión (session) es el período de tiempo que un usuario permanece en un sitio web, desde el instante en que ingresa al sitio web hasta que sale de ese sitio. La sesión continúa mientras el usuario navega de página en página en el sitio web.

En PHP existen varios métodos para administrar las sesiones de usuarios. A continuación se presentan dos de ellos:

- Cookies HTTP: **$_COOKIE**
- Variables de Sesión: **$_SESSION**

5.1 Cookies HTTP: $_COOKIE

Una cookie es un conjunto de datos que se pasa en el encabezado HTTP (HTTP Header). Las cookies se almacenan en el computador del cliente en forma de archivo de texto. Sus valores están en formato ASCII. Las cookies no aceptan datos binarios. El archivo cookie contiene:

- Nombre de dominio del servidor que creó la cookie.
- Ruta donde se almacena la cookie.
- Nombre de la cookie.
- Tiempo de vida de la cookie.
- Valor de la cookie.

Las cookies creadas se guardan dentro del computador del cliente y pueden usarse después. Por ejemplo, una cookie puede utilizarse para almacenar el nombre del usuario y enviar un mensaje de bienvenida más adelante. El propietario de la cookie establece su tiempo de vida.

Los valores cookies se almacenan en pares "`nombre = valor`", delimitados por un punto y coma (`;`). Se pueden crear nombres de variable y asignarles valores.

Hay cuatro pares de valores que se almacenan en una cookie:

- **El atributo domain:** Se usa para identificar el servidor que ha creado la cookie, si se necesita usarla en una referencia futura. Determina el dominio de Internet al cual la cookie debe ser enviada. Por ejemplo, domain = www.ibm.com.
- **El atributo path:** Se usa para especificar el lugar dentro del sitio web donde la cookie debe enviarse de regreso al servidor.
- **El atributo secure:** Especifica que la cookie puede ser transmitida solamente si el canal de comunicación con el host es seguro.
- **El atributo expires:** Especifica la fecha de expiración y debe escribirse en un formato particular para ser reconocido universalmente, tal como: `day,dd-mm-yy, hh:mm:ss, GMT.`

Un ejemplo de un encabezado de mensaje HTTP que lleva una cookie se muestra a continuación:

```
HTTP/1.1 200 OK
Date: Mon, 25 Aug 2003 13:40:22 GMT
Server: Apache/2.0.47 (Unix) PHP/5.0.0b1
```

```
X-Powered-By: PHP/5.0.0b1
Set-Cookie: miCookie=JLS; expires=Thu, 25-Aug-2005 14:40:27
GMT; path=/; domain=ve.ibm.com
Connection: close
Content-Type: text/html; charset=ISO-8859-1
```

Las cookies en PHP, se almacenan en la variable predefinida `$_COOKIE`. Si se necesita recuperar el valor de una cookie se escribe lo siguiente:

```
<? $miCookie = $_COOKIE[ 'nombreCookie' ] ; ?>
```

Para crear una cookie en PHP se utiliza la función `setcookie()`.

5.1.1 Función `setcookie(nombre,valor,expires,path,domain,secure)`

Define una cookie para ser enviada con el resto de la información del encabezado HTTP. Las cookies deben enviarse antes de mandar cualquier otra información de encabezado (esta es una restricción de las cookies, no de PHP). Esto requiere que sitúe las llamadas a esta función antes de cualquier etiqueta `<html>` o `<head>`.

Todos los parámetros excepto `'nombre'` son opcionales. Si sólo se especifica el parámetro `'nombre'`, la cookie con ese nombre no se almacenará en el cliente remoto. También puede sustituir cualquier parámetro por una cadena de texto vacía ("") y saltar así ese parámetro. Los parámetros `'expire'` y `'secure'` son números enteros si se quieren obviar utilice un cero (0). El parámetro `'expire'` es un entero de tiempo en formato GMT. El parámetro `'secure'` indica que la cookie se debe transmitir única y exclusivamente sobre una conexión segura HTTPS.

Así se crea una cookie:

```
<?
setcookie("idUs","Josefina",time()+82000,"/","",0);
?>
```

Y así se recupera su valor:

```
<?
$miCookie = $_COOKIE[ "idUs"] ;
echo $miCookie; // Imprime: 'Josefina'
?>
```

Si no se especifica el atributo `'expires'`, la cookie no se almacenará en el computador del cliente, pero existirá mientras el navegador no se cierre o mientras no sea destruida explícitamente.

Ahora observe un ejemplo donde se trabaje con cookies.

Ejemplo 5.3

En el siguiente ejemplo el usuario debe ingresar ciertos datos en un formulario. Una vez enviados, se realiza el registro del usuario guardando su información en cookies. Posteriormente, el usuario puede consultar la información que ingresó, mediante la página `misDatos.php`.

Para realizar este ejemplo se necesita una página html (`ejemplo-5.3.html`), que contiene el formulario para ingresar la información; también se requiere un script PHP (`registro.php`), que se encarga de almacenar los datos en las cookies. Finalmente, el script `misDatos.php`, obtiene la información almacenada en las cookies y la muestra en el navegador.

El código HTML comienza aquí...

```
1.  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
2.  <HTML>
3.  <HEAD><TITLE>Uso de las Cookies - Ejemplo 5.3</TITLE></HEAD>
4.  <BODY>
5.     <H2>Registro de Usuarios</H2>
6.     <HR>
7.  <FORM name="formDatos" action="registro.php" method="get">
8.  Nombre: <INPUT type="text" name="nombre"><BR>
9.  Apellido: <INPUT type="text" name="apellido"><BR>
10. Sexo: <INPUT type="radio" name="sexo" value="femenino">
    femenino
11.    
12. <INPUT type="radio" name="sexo" value="masculino">
    masculino<BR>
13. E-mail: <INPUT type="text" name="email"><BR><BR>
14. Pasatiempos:<BR>
15. <INPUT type="checkbox" name="pasatiempos[]"
    value="Leer">Leer<BR>
16. <INPUT type="checkbox" name="pasatiempos[]" value="Video
    juegos">
17. Video juegos<BR>
18. <INPUT type="checkbox" name="pasatiempos[]" value="Playa">
19. Playa<BR>
20. <INPUT type="checkbox" name="pasatiempos[]" value="Cine">
21. Cine<BR><BR>
22. Ciudades:<BR>
23. <SELECT name = "ciudades[]" multiple>
24. <OPTION>Londres</OPTION>
25. <OPTION>Praga</OPTION>
```

```
26.  <OPTION>New York</OPTION>
27.  <OPTION>Paris</OPTION>
28.  <OPTION>Barcelona</OPTION>
29.  </SELECT><BR><BR>
30.  <INPUT type="submit" name="Submit" value="Enviar">
31.  </FORM>
32.  </BODY>
33.  </HTML>
```

El código HTML termina aquí

El código anterior corresponde a la página ejemplo-5.3.html. Note que los controles del formulario tales como el `checkbox` y `select`, pueden contener múltiples valores.

Para que el script PHP pueda acceder a todos los valores que almacenan esos controles, se debe especificar el nombre de estos como si fuesen arreglos (`pasatiempos[ ]` y `ciudades[ ]`, respectivamente). De esta forma las variables de entorno utilizadas para obtener los datos enviados, ya sea `$_POST`, `$_GET` o `$_REQUEST`, almacenarán un arreglo con todos los valores que corresponden, y no un único valor.

A continuación se muestra el código para el script `registro.php`.

El código PHP inicia aquí...

```
1.   <?php
2.   // * Inicializando las variables a utilizar
3.   $nombre=NULL;
4.   $sexo=NULL;
5.   $email=NULL;
6.   $pasatiempos=NULL;
7.   $ciudades=NULL;
8.   // * Se verifica si existen los campos dentro de la variable
     $_GET
9.   if (array_key_exists('nombre', $_GET))
10.  {
11.    $nombre=$_GET[ "nombre"] ;
12.  }
13.
14.  if (array_key_exists('apellido', $_GET))
15.  {
16.    $nombre.=" ".$_GET[ "apellido"] ;
17.  }
18.
```

```
19.  if (array_key_exists('sexo', $_GET))
20.  {
21.    $sexo=$_GET[ "sexo"] ;
22.  }
23.
24.  if (array_key_exists('email', $_GET))
25.  {
26.    $email=$_GET[ "email"] ;
27.  }
28.  // * Se recorre el arreglo de pasatiempos para tomar cada
     uno
29.  if (array_key_exists('pasatiempos', $_GET))
30.  {
31.     $pasatiempos=$_GET[ "pasatiempos"][ 0] ;
32.     for ($i=1; $i<count($_GET[ "pasatiempos"] ); $i++)
33.    {
34.       $pasatiempos.=",".$_GET[ "pasatiempos"][ $i] ;
35.     }
36.  }
37.  // * Se recorre el arreglo de ciudades para tomar cada una
38.  if (array_key_exists('ciudades', $_GET))
39.  {
40.   $ciudades=$_GET[ "ciudades"][ 0] ;
41.   for ($i=1; $i<count($_GET[ "ciudades"] ); $i++) {
42.      $ciudades.=",".$_GET[ "ciudades"][ $i] ;
43.   }
44.  }
45.  // * Se crean las cookies con la información del usuario
46.  setcookie("nombre", "$nombre",time()+82000,"/","",0);
47.  setcookie("sexo", "$sexo",time()+82000,"/","",0);
48.  setcookie("email", "$email",time()+82000,"/","",0);
49.  setcookie("pasatiempos","$pasatiempos",time()+82000,"/","",0
     );
50.  setcookie("ciudades","$ciudades",time()+82000,"/","",0);
51.  ?>
52.  <HTML>
53.  <HEAD><TITLE>Uso de las Cookies - Ejemplo 5.3</TITLE></HEAD>
54.  <BODY>
55.  <H2 align="center">Información Registrada!</H2>
56.  <HR>
```

```
57.  <?php
58.  print ("<TABLE bordercolor=blue border=1 align=center>\n");
59.  print("<TR><TD>Nombre:</TD><TD>$nombre</TD></TR>");
60.  print("<TR><TD>Sexo:</TD><TD>$sexo</TD></TR>");
61.  print("<TR><TD>Email:</TD><TD>$email</TD></TR>");
62.  print("<TR><TD>Pasatiempos:</TD><TD>$pasatiempos</TD></TR>")
     ;
63.  print("<TR><TD>Ciudades:</TD><TD>$ciudades</TD></TR>");
64.  print ("</TABLE>");
65.  print ("<HR>");
66.  print("Puede consultar la información enviada en:
     http://localhost/misDatos.php");
67.  ?>
68.  </BODY>
69.  </HTML>
```

El código PHP termina aquí

Observe en el código anterior, como se accede a las variables pasatiempos y ciudades.

A continuación se muestra el código para el script `misDatos.php`. Este script se ejecuta luego de que se ha registrado la información para el usuario, cuando se presiona el botón "enviar".

Se recomienda cerrar la ventana del navegador, abrir una nueva ventana del navegador y ejecutar este script directamente, para que compruebe que la información del usuario aún persiste y se recupera de las cookies.

El código PHP comienza aquí...

```
1.   <?php
2.   $nombre=$_COOKIE['nombre'];
3.   $sexo=$_COOKIE['sexo'];
4.   $email=$_COOKIE['email'];
5.   $pasatiempos=$_COOKIE['pasatiempos'];
6.   $ciudades=$_COOKIE['ciudades'];
7.   ?>
8.   <HTML>
9.   <HEAD><TITLE>Uso de las Cookies - Ejemplo 5.3</TITLE></HEAD>
10.  <BODY>
11.  <H2 align="center">Bienvenido(a)! Sr(a) <?=$nombre?></H2>
12.  <HR>
13.  <H3 align="center">Esta es la información
     registrada:</H3>
```

```
14.  <?php
15.  print ("<TABLE bordercolor=blue border=1 align=center>\n");
16.  print("<TR><TD>Nombre:</TD><TD>$nombre</TD></TR>");
17.  print("<TR><TD>Sexo:</TD><TD>$sexo</TD></TR>");
18.  print("<TR><TD>Email:</TD><TD>$email</TD></TR>");
19.  print("<TR><TD>Pasatiempo:</TD><TD>$pasatiempos</TD></TR>");
20.  print("<TR><TD>Ciudades:</TD><TD>$ciudades</TD></TR>");
21.  print ("</TABLE>");
22.  print ("<HR>");
23.  print("Gracias por consultar su información!");
24.  ?>
25.  </BODY>
26.  </HTML>
```

El código PHP termina aquí

El resultado de la ejecución de `ejemplo-5.3.html`, se muestra en la Figura 5.5. Luego se muestra el resultado de `registro.php` en la Figura 5.6. Y por último, se muestra el resultado de `misDatos.php`, en la Figura 5.7.

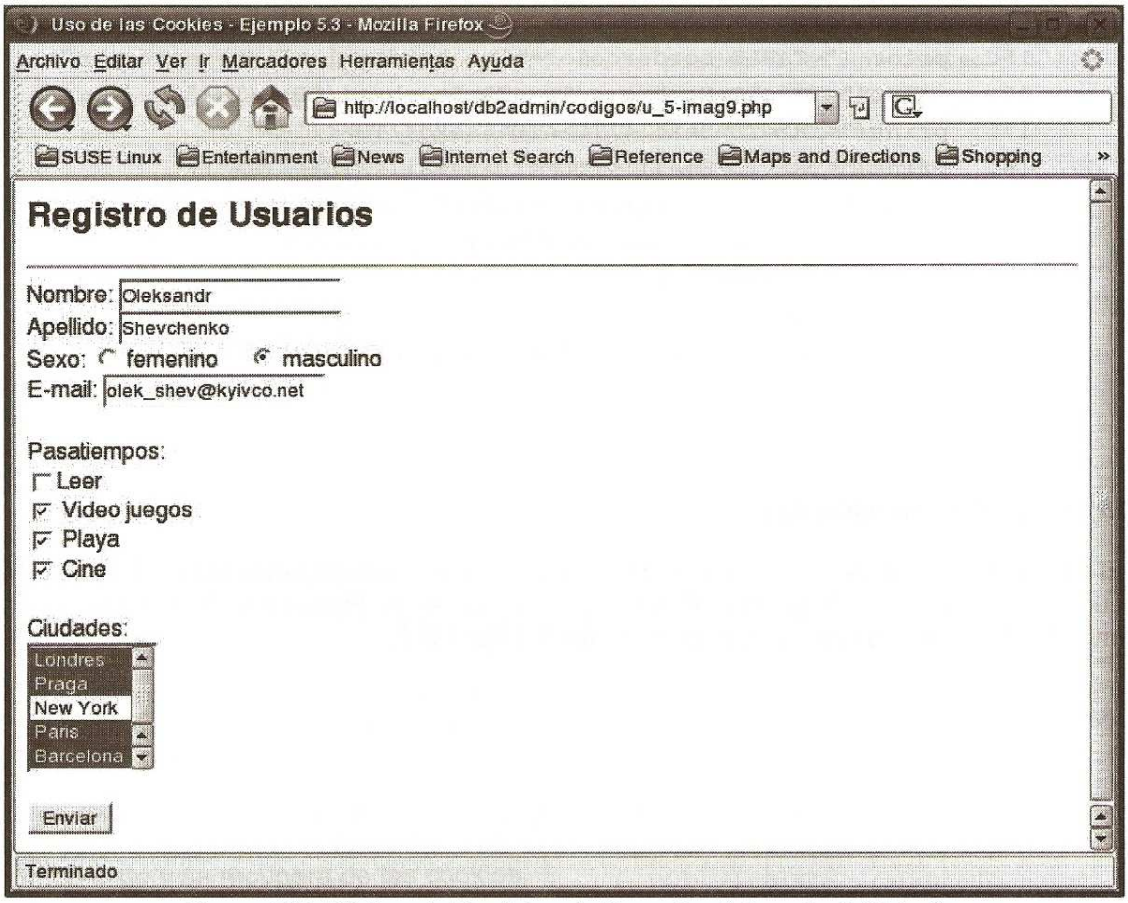


Figura 5.5: Página de Registro de Usuarios (ejemplo-5.3.html)

Cuando el usuario completa la información y presiona el botón "enviar", se ejecuta `registro.php`. el resultado se observa en la siguiente imagen.

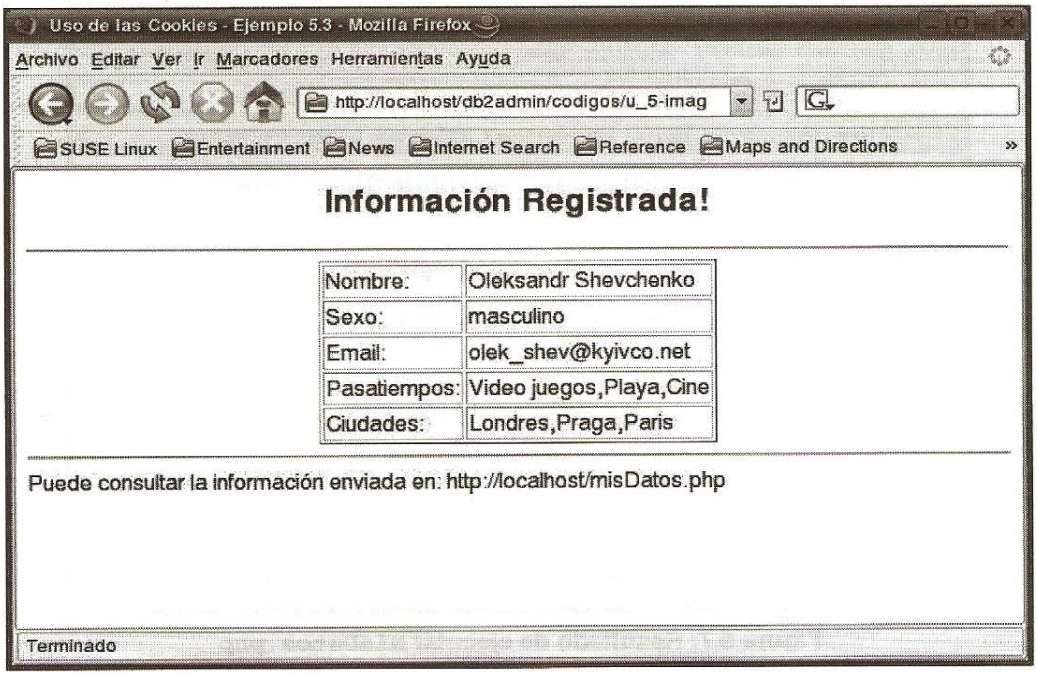


Figura 5.6: Resultado de Ejecutar `registro.php`

Al ejecutar el script `misDatos.php`, se obtiene el siguiente resultado:

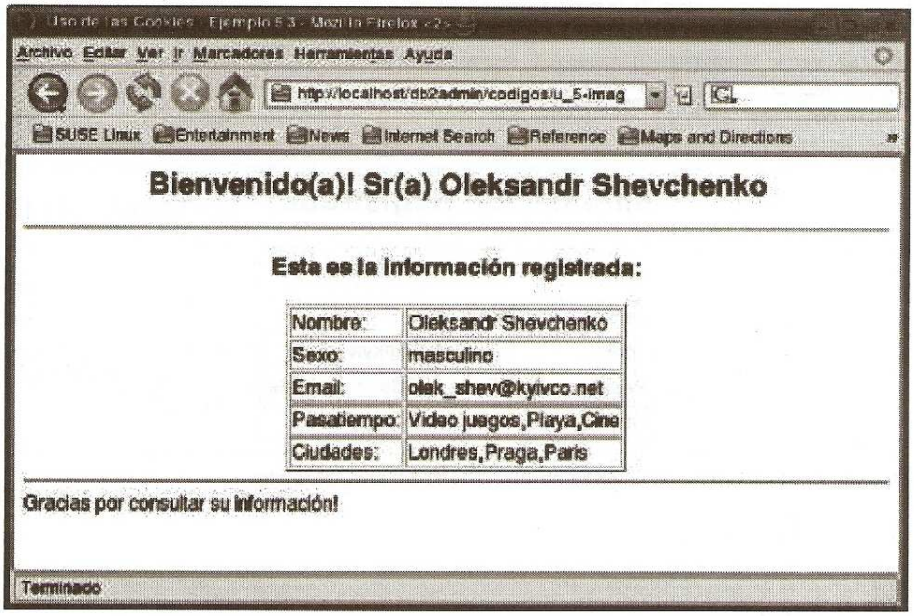


Figura 5.7: Resultado de ejecutar `misDatos.php`

Fin del Ejemplo 5.3

5.2 Variables de Sesión: `$_SESSION`

Las variables de sesión en PHP se guardan en el arreglo asociativo, `$_SESSION`. Contiene las variables de sesión disponibles en el script actual.

Las variables de sesión se crean como una cookie temporal y se envían en el encabezado del mensaje HTTP, como se muestra a continuación:

```
HTTP/1.1 200 OK
Date: Tue, 26 Aug 2003 16:54:44 GMT
Server: Apache/2.0.47 (Unix) PHP/5.0.0b1
X-Powered-By: PHP/5.0.0b1
Set-Cookie: PHPSESSID=b3228ce5e66834bc2ced42a899328796;
path=/
Expires: Thu, 19 Nov 1980 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-
check=0, pre-check=0
Pragma: no-cache
Content-Type: text/html; charset=ISO-8859-1
```

El soporte a las sesiones permite registrar un número arbitrario de variables de sesión que se conservarán en las subsiguientes peticiones.

Cuando un visitante accede a un sitio web, PHP comprobará automáticamente si `session.auto_start` está puesto a 1 (`session.auto_start` es una directiva que se encuentra en el archivo `php.ini`) o cuando se especifique explícitamente utilizando la función `session_start()` o implícitamente a través de la función `session_register()`.

Si existe un "`session id`" (`PHPSESSID = b3228ce5e66834bc2ced42a89938796.`) específico con la solicitud del cliente, se pueden recuperar las variables que se habían guardado anteriormente.

Si la directiva `sesion.auto_start` se coloca en 1 en el `php.ini`, cada vez que un cliente acceda a un script en el servidor por primera vez, automáticamente se creará una sesión que genera un id de sesión. Por eso es conveniente iniciarlas dentro de los scripts PHP, sólo cuando se necesite. Para ello se utiliza la función `session_start()`.

5.2.1 Función `session_start()`

Crea una sesión (o la continúa basándose en un '`session id`' ya generado). Esta función debe llamarse antes de mandar cualquier otra información de encabezado (esta es una restricción de las cookies, no de PHP). Esto requiere que sitúe las llamadas a esta función antes de cualquier etiqueta <HTML> o <HEAD>.

```
1.  <?php
2.    session_start();
3.  ?>
4.  <HTML>
5.  <HEAD>
6.  <TITLE>Creando una Sesión</TITLE>
7.  </HEAD>
8.  <BODY>
9.  <?php
10.  $ID = session_id(); //retorna el id de sesión creado
11.  print "ID de la sesión: ".$ID;
12. ?>
13. </BODY>
14. </HTML>
```

La función `session_id()` devuelve el '`session id`' de la sesión actual. Si se le pasa como parámetro un `id`, reemplazará el '`session id`' actual.

5.2.2 Crear una Variable de Sesión

Las variables de sesión se pueden crear de dos formas:

- Directamente en el arreglo asociativo (variable predefinida) `$_SESSION`

Sencillamente se especifica el nombre de la variable como índice.

```
<?php
$_SESSION[ "userid"] ="Usuario1";
$_SESSION[ "nombre"] ="Joe";
?>
```

- Utilizando la función session_register(nombre)

Esta función registra una o más variables globales en la sesión actual.

```
<?
$userid = "Usuario1";
session_register("userid"); //crea la variable userid
?>
```

5.2.3 Acceder a una Variable de Sesión

Una variable de sesión se puede recuperar de la siguiente forma:

```
<?
$uid = $_SESSION[ "userid"] ;
$nom = $_SESSION[ "nombre"] ;
print $uid; // imprime el valor de userid
print $nom; // imprime el valor de nombre
?>
```

5.2.4 Destruir una Variable de Sesión

Las variables se pueden eliminar haciendo uso de las siguientes funciones: `session_destroy`, `session_unregister` y `session_unset`.

La función **`session_destroy()`**, destruye todos los datos asociados con la sesión actual. No destruye ninguna de las variables globales asociadas a la sesión ni la cookie.

La función **`session_unset()`**, elimina y libera el espacio ocupado por todas las variables de la sesión actual registradas.

La función **`session_unregister(nombreVar)`**, elimina la variable global llamada `nombreVar` de la sesión actual.

Observe el siguiente código:

```
<?
session_start();
session_destroy(); // Destruye la sesión
?>
```

Ahora observe un ejemplo donde se trabaje con variables de sesión.

Ejemplo 5.4

En el siguiente ejemplo se tiene un formulario en una página HTML (`ejemplo-5.4.html`), donde el usuario coloca un nombre de usuario y una contraseña. Al hacer clic en el botón aceptar, se ejecuta el script `inicioSesion.php`, que se encarga de validar los datos introducidos. Si los datos son correctos, se inicia la sesión y se muestran la información del usuario que está almacenada en una cookie. Al visitar otra página (`Licencia.php`) la sesión del usuario aún estará disponible.

Adicionalmente se tiene el script `cookieUsuario.php`, que se debe ejecutar antes de hacer el inicio de sesión, pues éste es el encargado de crear la cookie con los datos del usuario, incluyendo el login y el password para hacer la verificación.

El siguiente es el código para `cookieUsuario.php`.

El código PHP inicia aquí...

```
1.  <?php
2.   $nombre="Josefina Pérez";
3.   $email="josefp@hotmail.com";
4.   $direccion="Caracas - Venezuela";
5.   $login="josefp";
6.   $password="password1";
7.   setcookie("nombreUs", "$nombre",time()+82000,"/","",0);
8.   setcookie("emailUs", "$email",time()+82000,"/","",0);
9.   setcookie("direccionUs",
     "$direccion",time()+82000,"/","",0);
10.  setcookie("loginUs","$login",time()+82000,"/","",0);
11.  setcookie("passwordUs","$password",time()+82000,"/","",
     0);
12.  ?>
```

El código PHP termina aquí

El siguiente es el código para `ejemplo-5.4.html`.

El código HTML comienza aquí...

```
1.  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
```

```
2.  <HTML>
3.  <HEAD>
4.  <TITLE>Uso de las Variables de Sesión - Ejemplo 5.4</TITLE>
5.  </HEAD>
6.  <BODY>
7.  <H2 align="center">Inicio de sesión</H2>
8.  <HR>
9.  <H3 align="center">Ingrese su login y password:</H3>
10. <FORM name="formLogin" action="inicioSesion.php"
    method="POST">
11.    <TABLE align="center">
12.     <TR>
13.       <TD>Login: </TD>
14.       <TD><input type="text" name="login"></TD>
15.     </TR>
16.     <TR>
17.      <TD>Password: </TD>
18.      <TD><INPUT type="password" name="password"></TD>
19.     </TR>
20.     <TR>
21.      <TD><INPUT type="submit"
    name="aceptar"value="Aceptar"></TD>
22.    <TD><INPUT type="reset" name="borrar"
    value="Borrar"></TD>
23.   </TR>
24.  </TABLE>
25. </FORM>
26. </BODY>
27. </HTML>
```

El código HTML termina aquí

La página `ejemplo-5.4.htm`, invoca al script `inicioSesion.php`, cuando se envía el formulario.

El siguiente es el código para `inicioSesion.php`.

El código PHP comienza aquí...

```
1.  <?php
2.  /* Para ejecutar este script es necesario que haya ejecutado
    cookieUsuario.php, éste crea la cookie donde se almacenan
    los datos del usuario */
3.  session_start();
```

```
4.   $loginUs=$_COOKIE[ 'loginUs'] ;
5.   $passwordUs=$_COOKIE[ 'passwordUs'] ;
6.   ?>
7.   <HTML>
8.   <HEAD>
9.   <TITLE>Uso de las Variables de Sesión - Ejemplo 5.4</TITLE>
10.  </HEAD>
11.  <BODY>
12.  <?php
13.  if (($_POST[ 'login'] !=NULL) and ( $_POST[ 'password'] !=NULL))
14.  {
15.   $login=$_POST[ "login"] ;
16.   $password=$_POST[ "password"] ;
17.   if ($login==$loginUs and $password==$passwordUs)
18.         {
19.         $nombreUs=$_COOKIE[ 'nombreUs'] ;
20.         $emailUs=$_COOKIE[ 'emailUs'] ;
21.         $direccionUs=$_COOKIE[ 'direccionUs'] ;
22.         $ID = session_id();
23.         $_SESSION[ "idUsuario"]  = $loginUs;
24.         $_SESSION[ "nombreUsuario"]  = $nombreUs;
25.
26.  echo "<H3 align=center>
27.        Sr(.a) $nombreUs , ha iniciado sesión!</h3>";
28.   echo "<HR>";
29.   echo "<P align=center>
30.  Esta es la información almacenada:</P>";
31.  print ("<TABLE bordercolor=blue border=1
     align=center>\n");
32.  print("<TR><TD>Nombre:</TD><TD>$nombreUs</TD></TR>");
33.  print("<TR><TD>Email:</TD><TD>$emailUs</TD></TR>");
34.  print("<TR><TD>Dirección</TD><TD>$direccionUs</TD></TR>");
35.  print ("</TABLE>");
36.   echo "<P align=center>
37.  Su ID de sesión: <B>".$ID."</B><BR>\n";
38.   echo "Su ID de Usuario es:
39.  <B>".$_SESSION[ "idUsuario"]  ."</B></P>";
40.   print ("<HR>");
41.   echo "<A href=\"Licencia.php\">Ver condiciones</A>";
42.         }
```

```
43.         else
44.         {
45.   echo "<H3 align=center>
46.   Login y/o Password incorrectos!</H3>";
47.         echo "<A href=\"ejemplo-5.4.html\">
48.   Intente de nuevo!</A>";
49.         echo "<HR>";
50.         }
51.   }
52.   else
53.   {       echo "<H3 align=center>
54.   Debe completar todos los campos!</H3>";
55.   echo "<A href=\"ejemplo-5.4.html\">Intente de nuevo!</A>";
56.    echo "<HR>";
57.   }
58.   ?>
59.   </BODY>
60.   </HTML>
```

El código PHP termina aquí

El siguiente es el código para `Licencia.php`.

El código PHP comienza aquí...

```
1.      <?
2.        session_start();
3.      ?>
4.    <HTML>
5.    <HEAD>
6.    <TITLE>Uso de las Variables de Sesión - Ejemplo 5.4</TITLE>
7.    </HEAD>
8.    <BODY>
9.    <H2 align="center"> Condiciones de Uso </H2>
10.   <HR>
11.   <P>Sr(a).  
12.   <B>
13.   <?php print $_SESSION[ "nombreUsuario"];?>
14.   </B><BR>
15.   Id de Usuario:  <B>
16.   <?php print $_SESSION[ "idUsuario"];?>
17.   </P>
```

```
18.  <BR>
19.  <B> * Estas son las condiciones de uso:</B>
20.  <BR>
21.  <FONT face="verdana"size="1" color="#CCCCCC"> Ud. está de
     acuerdo con todo!!!</FONT>
22.  </BODY>
23.  </HTML>
```

El código PHP termina aquí

La página `ejemplo-5.4.html`, que contiene el formulario de inicio de sesión, se muestra en la Figura 5.8.

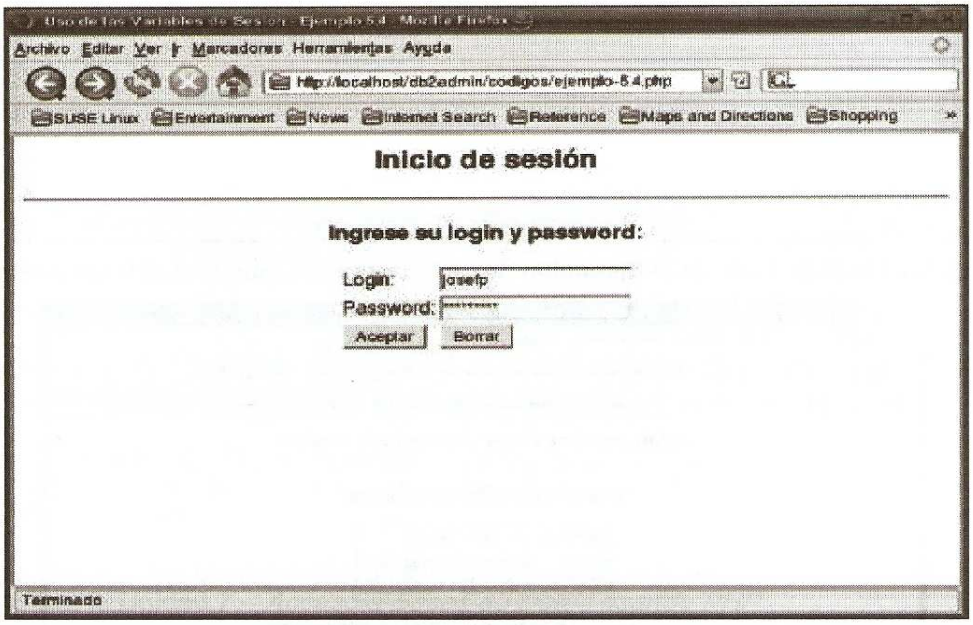


Figura 5.8: Página de Inicio de Sesión `ejemplo-5.4.html`

Cuando se presiona el botón 'Aceptar' se invoca al script inicioSesion.php. El resultado se puede observar en las Figuras 5.9 y 5.10. En el caso de que los datos introducidos sean incorrectos, se muestra un mensaje de error. Si los datos son correctos se muestra la información del usuario.

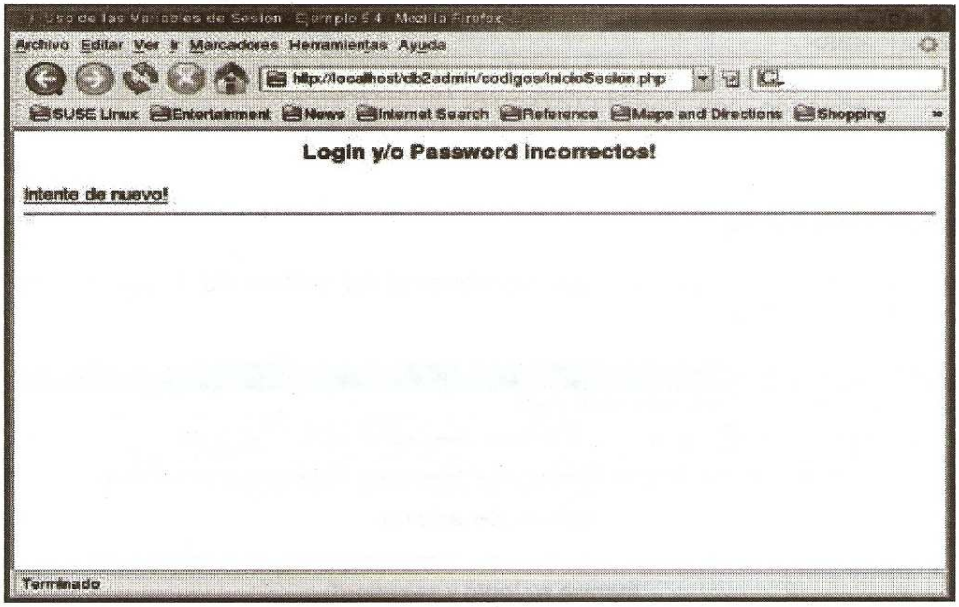


Figura 5.9: Resultado de la Ejecución de `inicioSesion.php` (datos incorrectos)

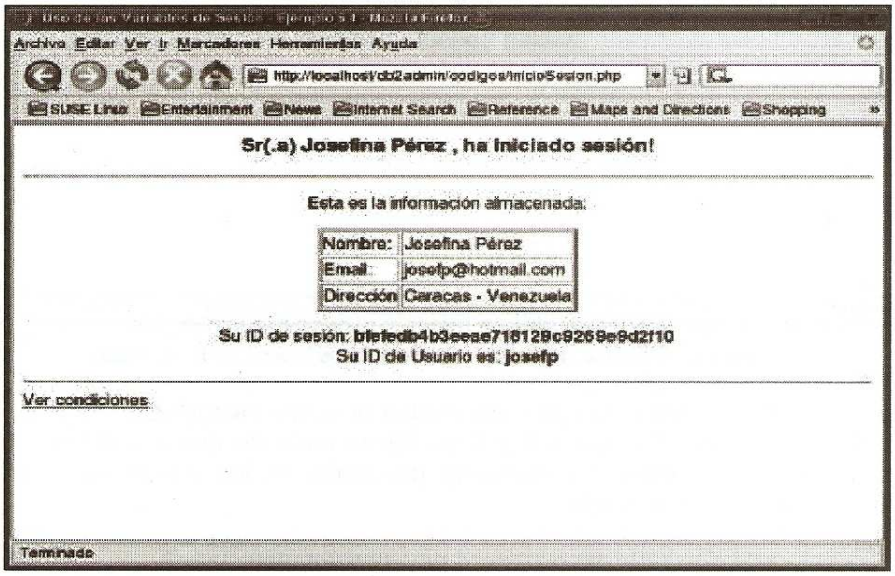


Figura 5.10: Resultado de la Ejecución de `inicioSesion.php` (datos correctos)

Cuando se presiona el enlace 'Ver acuerdo de licencia', se invoca a Licencia.php. El resultado se puede observar en la Figura 5.11.

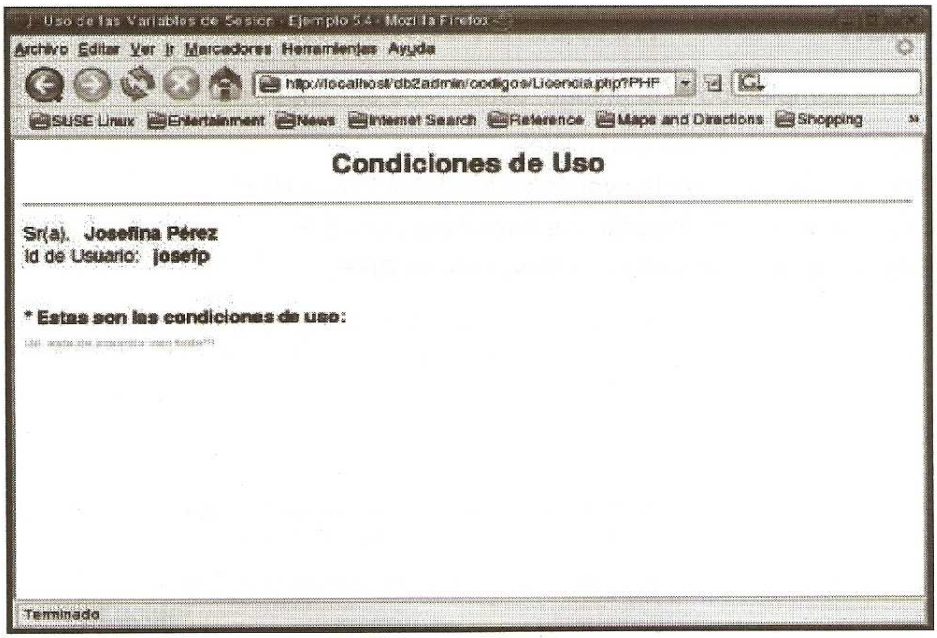


Figura 5.11: Resultado de ejecutar `Licencia.php`

Fin del Ejemplo 5.4

Aquí culmina la discusión de los elementos necesarios para desarrollar sitios web que logren interacción con el usuario.

Se han presentado los fundamentos necesarios para manipular los datos del formulario y administrar las sesiones de los usuarios. Ambos aspectos son primordiales para desarrollar sitios web.

Adicionalmente, otro elemento importante es el acceso a bases de datos. Este tópico se estudiará en la Unidad 7 – *Acceso a Bases de Datos usando PHP*.

Resumen

Ahora que ha completado esta unidad, usted debe ser capaz de:

- Definir el término Aplicación Web.
- Discutir la utilidad de las variables predefinidas de PHP.
- Explicar cómo se procesan los formularios en PHP.
- Explicar la administración de Sesiones en PHP.

Unidad 5: Examen de Autoevaluación

1) ¿Cuáles de las siguientes son variables predefinidas de PHP?
   a) $_REQUEST
   b) $_ARRAYS
   c) $_POST
   d) $GLOBALS

2) Una variable predefinida *superglobal* o global automática, está disponible en todos los contextos a lo largo de un script.
   a) Verdadero
   b) Falso

3) ¿Cuáles de las siguientes sentencias son ciertas con respecto a algunos de los elementos contenidos en la variable `$_SERVER` ?
   a) `SERVER_NAME` representa el nombre del servidor donde se está ejecutando el script actual
   b) `PHP_SELF` contiene el nombre del script que se está ejecutando actualmente
   c) `METHOD_GET` contiene la cadena de consulta mediante la cual se accedió a la página
   d) `REQUEST_METHOD`: indica cual método de solicitud fue usado para acceder a la página, que puede ser 'GET' o 'POST'.

4) ¿Cuáles de las siguientes son formas de recuperar los datos de un formulario?
   a) Si se usa el método 'POST' para enviar el formulario se utiliza la variable $_POST y la variable $_REQUEST, para recuperar la información
   b) Con la función `import_request_variables(tipo,prefijo)`, se pueden importar las variables GET o POST
   c) Si se usa el método 'GET' para enviar el formulario se utiliza la variable $_GET y la variable $_REQUEST, para recuperar la información
   d) Si se usa el método 'GET' para enviar el formulario se utiliza la variable $_GET y la variable $GLOBALS, para recuperar la información

5) ¿Cuáles de las siguientes variables de entorno permiten trabajar con sesiones de usuario?
   a) $_GET y $_POST
   b) $_COOKIE y $_SESSION
   c) $_SERVER y $_SESSION
   d) $_ENV y $_SESSION

6) ¿Cuáles de las siguientes afirmaciones son ciertas con respecto a las cookies en PHP?
   a) La función `setcookie()`, permite crear una cookie
   b) Las cookies deben enviarse antes de mandar cualquier otra información de encabezado
   c) Todos los parámetros que recibe la función `setcookie()` son obligatorios
   d) El parámetro `'secure'` indica que la cookie se debe transmitir única y exclusivamente sobre una conexión segura HTTPS

7) ¿Cuáles de las siguientes afirmaciones son ciertas con respecto al método GET?
   a) Este método envía los datos del formulario al script en el cuerpo del mensaje de solicitud HTTP
   b) Los datos enviados son agregados al URL
   c) Cuando se usa este método una cadena de caracteres es visible en la barra de direcciones del navegador
   d) Este método envía los datos del formulario al script en el encabezado del mensaje de solicitud HTTP

8) Las variables de sesión en PHP se guardan en el arreglo asociativo `$_SESSION`
   a) Verdadero
   b) Falso

9) ¿Cuáles de las siguientes afirmaciones son ciertas con respecto a las sesiones en PHP?
   a) La función `session_start()` se utiliza para iniciar una sesión en PHP
   b) La llamada a la función `session_start()` se puede ubicar en cualquier punto del script PHP, incluso luego de las etiquetas <html> o <head>
   c) La función `session_id()` devuelve el '`session id`' de la sesión actual
   d) Una variable de sesión se puede recuperar utilizando la variable predefinida $_ENV

10) ¿Qué se está realizando en el siguiente segmento de código?

```
<?php
import_request_variables('P', 'p_');
      $nom = $p_nombre;
      $ed = $p_edad;
 ?>
```

   a) Se está recuperando la información de un formulario que se envió con el método GET

b) Se está recuperando la información de un formulario que se envió con el método POST
c) Se está recuperando la información de una cookie
d) Se está recuperando la información de una variable de sesión

Unidad 5: Respuestas a Examen de Autoevaluación

1) a, c y d
2) a
3) a, b y d
4) a, b y c
5) b
6) a, b y d
7) b, c y d
8) a
9) a y c
10) b

Unidad 6: Lab. Desarrollo de Sitios Web

Objetivos de Aprendizaje

Al final de esta unidad, usted será capaz de:

- Crear una aplicación web sencilla en PHP.
- Obtener y manipular los campos de un formulario como entrada de datos.
- Crear y utilizar cookies.
- Administrar sesiones en PHP.

Ejercicios de Laboratorio

1) Realice una pequeña aplicación web en PHP, que permita crear un carrito de compras sencillo. Para ello utilice cookies.
   - Debe tener un formulario inicial para capturar los datos del usuario (nombre y dirección de correo). (`formulario.html`)
   - Cuando el usuario presione al botón 'ingresar' debe aparecer otro formulario donde puede seleccionar los productos que desea comprar (utilice campos checkbox). (`carrito.php`).
   - Cuando el usuario haya seleccionado los productos, debe presionar un botón 'submit' para registrar su compra. En la siguiente página se deben imprimir los productos que el compró. (`resultado.php`).
   - Si el usuario regresa a la página del carrito de compras deben aparecer los valores que él ya había seleccionado.
2) Crear una aplicación web en PHP de conversión de moneda. Las variables de factores de conversión deben ser inicializadas como se detalla continuación:

	Monedas Destino				
Moneda Origen	Bolívar (VEB)	Dólar (USD)	Euro (EUR)	Yen (JPY)	Peso (MXP)
Bolívar (VEB)	1	0,00063	0,00052	0,0734	0,0658
Dólar (USD)	1920	1	0,9	117,45	10,53
Euro (EUR)	2500	1,38	1	139,307	12,49
Yen (JPY)	13,623	0,0085	0,0071	1	0,089
Peso (MXP)	191,95	0,095	0,08	11,538	1

Tabla 6.1: Tabla de Conversión de Moneda

   - El script PHP debe tomar como entrada de usuario la moneda origen y la cantidad de monedas a convertir. La cantidad ingresada debe ser convertida a las otras monedas. Por ejemplo si ingresa como moneda origen bolívar, debe llevar la cantidad de bolívares ingresados a todas las demás monedas (dólar, euro, yen, peso mexicano).
   - La conversión de moneda debe escribirse como una función. El script PHP que realiza el procesamiento del formulario y retorna el resultado, debe usar la función definida por el usuario.
   - El resultado de la conversión debe ser presentado en una tabla.
   - Las variables de factores de conversión deben almacenarse en un archivo. De allí se deben leer los valores y usarlos dentro de la aplicación.

Unidad 7: Acceso a Base de Datos usando PHP

Objetivos de Aprendizaje

Al final de esta unidad, usted será capaz de:

- Describir cómo se trabaja con bases de datos en PHP.
- Explicar cómo conectarse y desconectarse a una base de datos.
- Explicar cómo recuperar, insertar, eliminar y actualizar registros.
- Utilizar la extensión de PHP para acceder bases de datos en MySQL.

1. Introducción

Se ha visto en este curso la utilidad de PHP para construir aplicaciones web. En esta unidad, se discutirá cómo los scripts PHP pueden interactuar con bases de datos. La creciente complejidad de las aplicaciones de negocio ha creado la necesidad de acceder a datos de diferentes fuentes de datos. Las fuentes de datos pueden variar desde bases de datos a archivos planos. PHP proporciona a los usuarios una interfaz para interactuar con diferentes bases de datos.

2. Acceso a Bases de Datos

Como es bien conocido, una base de datos relacional es un conjunto de tablas que mantienen relaciones entre sí, permitiendo almacenar información en forma íntegra y consistente.

Entre las bases de datos relacionales que maneja PHP se pueden mencionar MySQL, MS SQLServer, Oracle e IBM UDB/DB2. En este curso se empleará MySQL.

Para trabajar las bases de datos es necesario manejar un conjunto de comandos que van a permitir realizar operaciones como:

- Conexión y desconexión a la base de datos.
- Crear y eliminar tablas.
- Sentencias SQL para insertar, borrar y actualizar registros.
- Sentencias SQL para recuperar registros de las tablas.

En PHP, dichas operaciones se realizan mediante funciones específicas.

2.1 Extensiones de PHP para Acceso a Bases de Datos

PHP posee extensiones para manipular diferentes bases de datos como: PostgreSQL, MySQL, Oracle (OCI7 y OCI8), Sybase, MS-SQL, InterBase, Informix, entre otras. Las funciones para trabajar con cada base de datos en particular, están disponibles en su respectiva extensión. Por lo tanto, esas funciones difieren para cada base de datos que se vaya a manejar.

En PHP, las funciones de manejo de bases de datos poseen como prefijo el nombre del manejador de base de datos que se va utilizar. Por ejemplo, si se trata de MySQL, se utiliza `mysql_connect()` para conectarse a la base de datos o `mysql_db_query()` para la ejecución de una sentencia SQL.

Para trabajar con el RDBMS MySQL se va a utilizar una extensión de PHP dedicada a este propósito. Para el entorno Windows, ésta extensión es el archivo "`php_mysql.dll`", que se encuentra disponible en el directorio de instalación de PHP, en el subdirectorio "`ext`".

3. Acceso a Bases de Datos con la Extensión MySQL

Para que un script PHP se conecte a una base de datos utilizando una extensión, ésta debe estar disponible para el script. Para ello se tienen dos formas de hacerlo:

- Incorporar la extensión de forma automática o por defecto:

Para ello se debe modificar el archivo de configuración `php.ini`. Para habilitar la extensión de MySQL se debe agregar la directiva `'extension'`, junto con el nombre de la extensión, dentro del archivo `php.ini`. Si se está trabajando en ambiente Windows, se agrega la siguiente sentencia:

```
extension="ext/php_mysql.dll"
```

La extensión habilitada estará disponible por defecto para todos los scripts que se ejecuten.

- Incorporar la extensión dinámicamente:

Para incorporar una extensión dinámicamente se debe hacer directamente desde el script PHP, utilizando la función **`dl()`**. La extensión se carga en tiempo de ejecución y estará disponible sólo para el script actual. Observe el siguiente segmento de código:

```
<?php
dl('ext/php_mysql.dll');
print "Se cargó el módulo de extensión de MySQL";
?>
```

Una vez incluida la extensión de MySQL para PHP, se puede comenzar a trabajar con las bases de datos.

3.1 Abrir una Conexión al Servidor MySQL

Antes de conectarse a una base de datos MySQL, se debe establecer la conexión con el servidor MySQL. Para ello se utilizan las funciones `mysql_connect()` y `mysql_pconnect()`.

- **`mysql_connect('host:numpuerto', $userid, $password)`**

Establece una conexión a un servidor MySQL. Retorna un identificador de conexión positiva si tiene éxito o falso si ocurre un error. Este identificador se utiliza luego para conectarse a la base de datos.

El parámetro `host` es el nombre del servidor, `numpuerto` es el número de puerto del servicio MySQL (este parámetro es opcional), `$userid` se refiere a un usuario autorizado para ingresar al servidor y `$password` es la clave que autentifica al usuario.

```
1.  <?
2.  $idConn =
    mysql_connect('localhost:3306','mysqladmin','mysqladmin');
3.  if (!$idConn) {
4.      die("No se conectó al servidor: " . mysql_error());
5.  }
6.  echo "Conexión satisfactoria";
7.  mysql_close($idConn);
8.  ?>
```

- **`mysql_pconnect ('host:numpuerto', $userid, $password);`**

Establece una conexión persistente a servidor MySQL. Retorna un identificador de conexión persistente si tiene éxito o falso si ocurre un error. Trabaja de forma similar que `mysql_connect()`, la diferencia radica en que `mysql_pconnect()` intenta encontrar una conexión abierta con el mismo host, usuario y password. Si lo encuentra, devuelve el identificador de esa conexión en vez de abrir otra conexión.

Una conexión establecida con `mysql_pconnect()` no será cerrada cuando acabe la ejecución del script. La conexión permanecerá abierta para ser usado en otros scripts.

Nota: Las conexiones permanentes no funcionan si PHP es usado como programa CGI.

3.2 Cerrar una conexión al Servidor MySQL

Una conexión al servidor MySQL se cierra haciendo uso de la función `mysql_close()`.

- **`mysql_close($idConn)`**

`mysql_close()` cierra la conexión MySQL que está asociada con el identificador de conexión especificado. Si no se especifica identificador de conexión, se asume por defecto la última conexión. Observe el siguiente segmento de código donde se abre y se cierra una conexión a base de datos.

```
1.  <?
2.  $idConn = mysql_connect
3.  ("localhost:3306","mysqladmin","mysqladmin");
4.  if ($idConn == 0) {
5.   echo "Falló la Conexión al Servidor MySQL!";
6.   $sqlerror = mysql_error($idConn);
7.   echo"$sqlerror";
8.   }
9.   else {
10.        echo "La Conexión a Base de Datos fue satisfactoria";
11. mysql_close($idConn); //Cerrando la conexión
```

```
12.  }
13.  ?>
```

Nota: `mysql_close()` no cierra un enlace establecido con `mysql_pconnect()`.

3.3 Seleccionar una Base de Datos en MySQL

Una vez establecida la conexión al servidor MySQL se puede seleccionar la base de datos con la cual se va a trabajar. Esto se hace utilizando la función `mysql_select_db()`.

- **`mysql_select_db($BD,$idConexion)`**

`mysql_select_db()` establece la base de datos activa que estará asociada con el identificador de conexión dado. El parámetro `$BD` contiene el nombre de la base de datos que se quiere seleccionar y `$idConexion` especifica el identificador de conexión del servidor. Si no se especifica un identificador de conexión, se asume la última conexión abierta.

Esta función retorna `true` si se pudo seleccionar correctamente la base de datos, `false` en caso de contrario. Observe el siguiente segmento de código, en el cual se selecciona una base de datos.

```
1.   <?php
2.   $idConn = mysql_connect('localhost:3306', 'mysqladmin',
     'mysqladmin');
3.   if (!$idConn) {
4.   die('No se pudo establecer la conexión : ' . mysql_error());
5.   }
6.   $dbSelect = mysql_select_db('SAMPLE', $idConn);
7.   if (!$dbSelect) {
8.   die ('No se pudo seleccionar la BD : ' . mysql_error());
9.   }
10.  ?>
```

Una vez que se ha seleccionado una base de datos, las consultas que se ejecutan se harán sobre esa base de datos utilizando la función `mysql_query()`. Si se quiere ejecutar consultas sobre otra base de datos se debe hacer una nueva selección.

4. Ejecutar Sentencias SQL Usando Funciones MySQL

La extensión de PHP para MySQL provee una función que permite ejecutar cualquier sentencia SQL. Esta función es `mysql_query()`.

- **mysql_query ("sentenciaSQL",$idConn)**

`mysql_query()` envía una consulta a la base de datos que está activa en el servidor asociado con el identificador de conexión `$idConn`. Recibe como parámetros la cadena de consulta SQL "`sentenciaSQL`" y opcionalmente el identificador de la conexión `$idConn`. Si no se especifica el identificador de conexión la consulta se ejecuta sobre la última base de datos seleccionada asociada a la última conexión abierta.

Si se ejecuta una sentencia SQL del tipo: `UPDATE`, `DELETE`, `DROP`, `INSERT` o `CREATE`, `mysql_query()` retorna `TRUE` en caso de éxito y `FALSE` en caso contrario.

Si se ejecuta una sentencia SQL del tipo `SELECT`, `SHOW` o `DESCRIBE`, `mysql_query()` regresa un `resource` en caso de éxito o `FALSE` en caso de error.

Un `resource` es una variable especial, que contiene una referencia a un recurso externo, como por ejemplo una tabla, una vista, un conjunto de registros o una fila. Los recursos son creados y usados por funciones especiales.

Nota: La sentencia mysql_query(), no sólo ejecuta sentencias SQL estándar, también ejecuta sentencias de creación y eliminación de bases de datos.

4.1 Crear una Tabla

En el siguiente código se crea una tabla llamada `EMPLEADOS` que tiene tres campos: `EMP_ID`, `EMP_NOMBRE` y `EMP_CARGO`.

```
1.  <?
2.  // Crear una tabla de productos
3.  $strsql = "CREATE TABLE EMPLEADOS(
4.   EMP_COD INTEGER NOT NULL,
5.          EMP_NOMBRE VARCHAR(20),
6.          EMP_PRECIO VARCHAR(20));";
7.  $result = mysql_query($strsql,$idConn);
8.  if (!$result) {
9.      die('Sentencia no ejecutada : ' . mysql_error());
10. }
11. ?>
```

La función `die()` imprime un mensaje y termina el script actual, en caso de que ocurra algún error.

4.2 Insertar Registros

A continuación se inserta un registro en la tabla `EMPLEADOS`.

```
1.  <?
```

```
2.  // Insertar un registro en EMPLEADOS
3.  $strsql = "INSERT INTO EMPLEADOS VALUES(10,'PEDRO PEREZ',
    'PROGRAMADOR');";
4.  $result = mysql_query($strsql,$idConn);
5.  ?>
```

4.3 Actualizar Registros

Es posible actualizar registros usando la sentencia `UPDATE`.

```
1.  <?
2.  // Actualizar un registro en EMPLEADOS
3.  $strsql = "UPDATE EMPLEADOS SET EMP_CARGO = 'DISEÑADOR'
    WHERE EMP_COD = 10;";
4.  $result = mysql_query($strsql,$idConn);
5.  ?>
```

En el código anterior, se ha cambiado el cargo del empleado con `EMP_COD = 10` a 'Diseñador'.

4.4 Eliminar Registros

La sentencia `DELETE` se usa para eliminar registros de una tabla. Vea el siguiente código ejemplo:

```
1.  <?
2.  // Eliminar un registro en EMPLEADOS
3.  $strsql = "DELETE FROM EMPLEADOS WHERE EMP_COD = 10;";
4.  $result = mysql_query($strsql,$idConn);
5.  ?>
```

En este código ejemplo, se eliminó un registro, que tenía el `EMP_COD = 10` de la tabla `EMPLEADOS`.

Ahora se aprenderá a eliminar una tabla.

4.5 Eliminar una Tabla

La sentencia `DROP TABLE` se usa para eliminar una tabla.

```
1.  <?
2.  // Eliminar la tabla EMPLEADOS
3.  $strsql = "DROP TABLE EMPLEADOS;";
4.  $result = mysql_query($strsql,$idConn);
5.  ?>
```

Ahora se aprenderá a obtener un conjunto de registros de una tabla, basados en ciertas condiciones.

4.6 Seleccionar Registros

La sentencia `SELECT` se usa para seleccionar registros específicos de una tabla. Vea el siguiente código ejemplo:

```
1.  <?
2.  // Seleccionar algunos campos de la tabla EMPLEADOS
3.  $strsql = "SELECT EMP_NOMBRE, EMP_CARGO FROM EMPLEADOS WHERE
    EMP_CARGO = 'DISEÑADOR';";
4.  $result = mysql_query($strsql,$idConn);
5.  if (!$result) {
6.      die('Sentencia no ejecutada : ' . mysql_error());
7.  }
8.  ?>
```

Cuando se ejecuta la sentencia `SELECT` se obtiene un conjunto de registros, a diferencia de las sentencias `INSERT`, `UPDATE` o `DELETE` que sencillamente actualizan la base de datos. Cuando se ejecuta una sentencia `SELECT`, la variable `$result` contiene el conjunto de registros recuperados.

La extensión de PHP para MySQL provee funciones que permiten recuperar los registros obtenidos. A continuación se explicarán estas funciones.

5. Recuperar Registros Usando Funciones MySQL

La extensión de PHP para MySQL provee varios métodos para recuperar los registros obtenidos al ejecutar una consulta. Estas funciones son: `mysql_fetch_array`, `mysql_fetch_row`, `mysql_fetch_object` y `mysql_fetch_assoc`.

- **`mysql_fetch_row($result)`**

`mysql_fetch_row()` retorna un vector que corresponde a una fila seleccionada en el conjunto de registros almacenados en `$result`. Selecciona una fila de datos del resultado especificado en `$result`.

Retorna la próxima fila del resultado o `FALSE` si ya no quedan más filas.

```
1.  <?php
2.  $result = mysql_query("SELECT EMP_COD,EMP_NOMBRE FROM
    EMPLEADO WHERE EMP_CARGO = 'PROGRAMADOR'");
3.  if (!$result) {
4.      echo 'No se pudo ejecutar la consulta: ' . mysql_error();
5.      exit;
6.  }
7.  $fila = mysql_fetch_row($result);
```

```
8.
9.    echo $fila[ 0] ;
10.   // * Se imprime el EMP_COD del primer registro retornado
11.   echo "<BR>"; //Para que los datos obtenidos no queden en la
      misma línea
12.   echo $fila[ 1] ;
13.   // * Se imprime el EMP_NOMBRE del primer registro retornado
14.   ?>
```

- **mysql_fetch_array($result)**

mysql_fetch_array()es similar a mysql_fetch_row(), pero además de poder acceder los datos a través de los índices numéricos del arreglo resultado, también se puede acceder mediante índices asociativos, utilizando los nombres de los campos como claves.

Retorna un arreglo que corresponde con un registro recuperado o FALSE si no hay más registros.

```
1.    <?php
2.    $result = mysql_query("SELECT EMP_COD,EMP_NOMBRE FROM
      EMPLEADO WHERE EMP_CARGO = 'PROGRAMADOR'");
3.    if (!$result) {
4.       echo 'No se pudo ejecutar la consulta: ' . mysql_error();
5.       exit;
6.    }
7.    $fila = mysql_fetch_array($result);
8.    echo $fila[ 0] ; // Es equivalente a $fila[ "EMP_COD"]
9.    // * Se imprime el EMP_COD del primer registro retornado
10.   echo "<BR>"; //Para que los datos obtenidos no queden en la
      misma línea
11.   echo $fila[ "EMP_NOMBRE"] ; // Es equivalente a $fila[ 1]
12.   // * Se imprime el nombre del empleado del primer registro
      retornado
13.   ?>
```

- **mysql_fetch_assoc($result)**

mysql_fetch_assoc() es similar a mysql_fetch_array(), sólo que retorna un arreglo asociativo que utiliza los nombres de los campos como claves.

Retorna un arreglo asociativo que corresponde con un registro recuperado o FALSE si no hay más registros.

```
1.    <?php
```

```
2.  $result = mysql_query("SELECT EMP_COD,EMP_NOMBRE FROM
    EMPLEADO WHERE EMP_CARGO = 'PROGRAMADOR'");
3.  $fila = mysql_fetch_array($result);
4.  echo $fila["EMP_COD"];
5.  // * Se imprime el EMP_COD del primer registro retornado
6.  echo "<BR>"; //Para que los datos obtenidos no queden en la
    misma línea
7.  echo $fila["EMP_NOMBRE"];
8.  // * Se imprime el EMP_NOMBRE del primer registro retornado
9.  ?>
```

- **mysql_fetch_object($result)**

`mysql_fetch_object()` es similar a `mysql_fetch_array()`, con la diferencia que retorna un objeto en vez de un arreglo. Sólo se pueden acceder a los datos por el nombre del campo y no por su posición.

Retorna un objeto que corresponde con un registro recuperado o `FALSE` si no hay más registros.

Para acceder a las propiedades de un objeto se utiliza el operador `->` . Por ejemplo: `$objeto->propiedad`.

```
1.  <?php
2.  $result = mysql_query("SELECT EMP_COD,EMP_NOMBRE FROM
    EMPLEADO WHERE EMP_CARGO = 'PROGRAMADOR'");
3.  $objetoFila = mysql_fetch_object($result);
4.  echo $objetoFila->EMP_COD;
5.  // * Se imprime el EMP_COD del primer registro retornado
6.  echo "<BR>"; //Para que los datos obtenidos no queden en la
    misma línea
7.  echo $objetoFila->EMP_NOMBRE;
8.  // * Se imprime el EMP_NOMBRE del primer registro retornado
9.  ?>
10.
```

La salida de las funciones anteriormente explicadas es la misma y se puede observar en la siguiente Figura:

Figura 7.1: Recuperar registros usando funciones MySQL

Para poder recuperar cada uno de los registros obtenidos en la consulta se debe un utilizar una estructura iterativa para recorrer todos los registros. Observe el siguiente ejemplo:

```
1.  <?php
2.  mysql_connect("localhost:3306", "mysqladmin", "mysqladmin");
3.  mysql_select_db("SAMPLE");
4.  $result = mysql_query("select * from EMPLEADO");
5.  while ($fila = mysql_fetch_array($result)) {
6.      echo $fila["EMP_COD"];
7.      echo $fila["EMP_NOMBRE"];
8.      echo $fila["EMP_CARGO"];
9.  }
10. mysql_free_result($result);
11. ?>
```

Se han mencionado las funciones básicas de la extensión de PHP para MySQL que permiten trabajar con bases de datos y manipular registros recuperados. A continuación se listan otras funciones de gran utilidad cuando se trabaja con base de datos.

6. Otras Funciones Útiles de la Extensión MySQL

- **`mysql_error($idConexion)`:** Retorna el texto del mensaje de error de la última operación realizada en MySQL.
- **`mysql_errno($idConexion)`:** Retorna el número de error de la última operación realizada en MySQL.
- **`mysql_ping($idConexion)`:** Verifica si la conexión con el servidor está activa o no. Si se ha interrumpido intenta realizar una reconexión.

- **mysql_info($idConexion):** Retorna información detallada sobre la última consulta realizada.
- **mysql_field_name($resultado,$indiceCampo):** Retorna el nombre del campo especificado. Recibe como parámetros el identificador de resultado (`$resultado`) y el índice del campo (`$indiceCampo`).
- **mysql_field_type($resultado,$indiceCampo):** Retorna el tipo de dato del campo especificado. Recibe como parámetros el identificador de resultado (`$resultado`) y el índice del campo (`$indiceCampo`).
- **mysql_num_fields($resultado):** Retorna el número de campos que existen en el conjunto de registros recuperados, los cuales están especificados por `$resultado`.
- **mysql_num_rows($resultado):** Retorna el número de filas que existen en el conjunto de registros recuperados, los cuales están especificados por `$resultado`.
- **mysql_list_tables($BD,$idConexion):** Retorna la lista de tablas que se encuentran en una base de datos.
- **mysql_list_dbs($idConexion):** Retorna la lista de bases de datos disponibles en el servidor MySQL.
- **mysql_free_result($resultado):** Libera la memoria ocupada por el resultado de la ejecución de una consulta. `$resultado` identifica el resultado obtenido.

Ejemplo 6.1

El código PHP comienza aquí...

```
1.  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
2.  <HTML>
3.  <HEAD><TITLE>Manejo de BD con MySQL - Ejemplo
    7.2</TITLE></HEAD>
4.  <BODY>
5.  <?
6.  //* Abrir una conexión con el servidor
7.  $id_conn =
    mysql_connect("localhost","mysqladmin","mysqladmin");
8.  if ($id_conn == 0) {
9.   echo "Falló la conexión a la base de datos!<br>";
10.  echo mysql_errormsg($id_conn);
11.     }
12. else{
13. // * Crear una BD
14. $dbResult= mysql_query('CREATE DATABASE BD_PAISES;');
15. if (!$dbResult) {
```

```
16.       die('No se pudo crear la BD: ' . mysql_error());
17.   }
18.   // * Seleccionar la Base de Datos de Países
19.   $dbSelect = mysql_select_db('BD_PAISES', $id_conn);
20.   if (!$dbSelect) {
21.   die ('No se pudo seleccionar la BD : ' . mysql_error());
22.   }
23.   //* Crear una tabla con el nombre PAIS
24.   $strsql= "CREATE TABLE PAIS
25.   (ID INTEGER NOT NULL, NOMBRE VARCHAR(20),
26.   PRESIDENTE VARCHAR(40),CONTINENTE VARCHAR(20));";
27.
28.   $result=mysql_query($strsql);
29.
30.   //**** Insertar registros en la tabla PAIS
31.   $strsql = "INSERT INTO PAIS VALUES(100,'USA','GEORGE
      BUSH','NORTEAMERICA');";
32.   $result=mysql_query($strsql);
33.
34.   $strsql = "INSERT INTO PAIS VALUES(200,'VENEZUELA','HUGO
      CHAVEZ','SURAMERICA');";
35.   $result=mysql_query($strsql);
36.
37.   $strsql = "INSERT INTO PAIS VALUES(300,'FRANCIA','NICOLAS
      SARKOZY','EUROPA');";
38.   $result=mysql_query($strsql);
39.
40.   $strsql = "INSERT INTO PAIS VALUES(400,'ESPAÑA','JOSE L.
      RODRIGUEZ ZAPATERO','EUROPA');";
41.   $result=mysql_query($strsql);
42.
43.   $strsql = "INSERT INTO PAIS VALUES(500,'CHINA','HU
      JINTAO','ASIA');";
44.   $result=mysql_query($strsql);
45.
46.   //**** Mostrar los detalles de la tabla PAIS
47.   $strsql = "SELECT ID, NOMBRE, PRESIDENTE,CONTINENTE FROM
      PAIS WHERE CONTINENTE='EUROPA';";
48.   $result=mysql_query($strsql);
49.   ?>
50.   <H2 align="center">Países Europeos</H2>
```

```
51.  <TABLE border="1" align="center">
52.  <TR>
53.  <TH>ID</TH><TH>NOMBRE</TH><TH>PRESIDENTE</TH><TH>CONTINENTE<
     /TH>
54.  </TR>
55.  <?
56.  while($registro=mysql_fetch_array($result)) {
57.   echo"<TR><TD>",$registro[ "ID"] ,"</TD><TD>",
58.   $registro[ "NOMBRE"] ,"</TD><TD>",
59.   $registro[ "PRESIDENTE"] ,"</TD><TD>",
60.   $registro[ "CONTINENTE"] ,"</TD></TR>";
61.  }
62.  ?>
63.  </TABLE>
64.  <?
65.  //**** Actualizar un registro en PAIS
66.  $strsql = "UPDATE PAIS SET ID = (ID + 500) WHERE CONTINENTE
     = 'ASIA';";
67.  $result=mysql_query($strsql);
68.
69.  //**** Eliminar un registro de la tabla PAIS
70.  $strsql = "DELETE FROM PAIS  WHERE ID > 500;";
71.  $result=mysql_query($strsql);
72.
73.  //**** Mostrar los detalles de la tabla PAIS
74.  $strsql = "SELECT ID, NOMBRE, PRESIDENTE, CONTINENTE FROM
     PAIS;";
75.  $result=mysql_query($strsql);
76.  ?>
77.  <HR>
78.  <H2 align="center">Tabla País Resultante</H2>
79.  <TABLE border="1" align="center">
80.  <TR><TH>ID</TH><TH>NOMBRE</TH><TH>PRESIDENTE</TH><TH>CONTINE
     NTE</TH></TR>
81.  <?
82.  while($registro=mysql_fetch_array($result)) {
83.  echo"<TR><TD>",$registro[ "ID"] ,"</TD><TD>",
84.  $registro[ "NOMBRE"] ,"</TD><TD>",
85.  $registro[ "PRESIDENTE"] ,"</TD><TD>",
86.  $registro[ "CONTINENTE"] ,"</TD></TR>";
87.  }
```

```
88.  ?>
89.  </TABLE>
90.  <? } ?> //Cierre del else de la Creación de la BD
91.  </BODY>
92.  </HTML>
```

El código PHP termina aquí

El resultado de ejecutar el `ejemplo-6.1.php`, se muestra en la Figura 7.1.

Figura 7.1: Resultado de Ejecutar el script `ejemplo-7.1.php`

Fin del Ejemplo 6.1

Se ha concluido la discusión acerca de las funciones de la extensión de PHP para MySQL que permiten el acceso y manipulación de datos en una base de datos.

Se han abarcado los tópicos fundamentales para acceder a una base de datos en entorno web utilizando PHP. Este aspecto es primordial para la creación de aplicaciones web.

Resumen

Ahora que ha completado esta unidad, usted debe ser capaz de:

- Describir cómo se trabaja con bases de datos en PHP.
- Explicar cómo conectarse y desconectarse a una base de datos.
- Explicar cómo recuperar, insertar, eliminar y actualizar registros.
- Utilizar la extensión de PHP para acceder bases de datos en MySQL.

Unidad 7: Examen de Autoevaluación

1) PHP posee extensiones para trabajar con Bases de Datos
   a) Verdadero
   b) Falso

2) PHP posee una extensión específica para el RDBMS MySQL.
   a) Verdadero
   b) Falso

3) ¿Cuáles son las 2 formas en las que se puede incorporar una extensión MySQL a un script PHP?
   a) En forma automática o por defecto
   b) Dinámicamente
   c) Utilizando la función `mysql_query()`
   d) Ninguna de las Anteriores

4) La función `mysql_fetch_array()` permite acceder a los datos a través de índices numéricos y asociativos.
   a) Verdadero
   b) Falso

5) La función `mysql_query()` permite ejecutar sentencias SQL estándar, sentencias de creación y eliminación de base de datos.
   a) Verdadero
   b) Falso

6) ¿Qué función se utiliza para cargar una extensión dinámicamente?
   a) die()
   b) dl()
   c) extension
   d) Ninguna de las Anteriores

7) La función `mysql_connect()` permite establecer una conexión directamente a una base de datos.
   a) Verdadero
   b) Falso

8) ¿Cuáles de las siguientes afirmaciones son ciertas?
   a) `mysql_select_db()` selecciona una base de datos, que está asociada con el identificador de conexión dado.
   b) `mysql_close()` cierra una conexión que se haya establecido con `mysql_pconnect()`.
   c) `mysql_create_db()` permite crear una base de datos nueva en el servidor asociado a un identificador de conexión.
   d) `mysql_drop()` permite eliminar una base de datos del servidor asociado un identificador de conexión

9) ¿Qué funciones provee la extensión de PHP para MySQL, para recuperar registros de una consulta?
   a) `mysql_fetch_hash()`
   b) `mysql_fetch_row()`
   c) `mysql_fetch_array()`
   d) `mysql_fetch_class()`

10) `mysql_field_type()` retorna el tipo de dato de un campo especificado.
   a) Verdadero
   b) Falso

Unidad 7: Respuestas a Examen de Autoevaluación

1) a
2) a
3) a y b
4) a
5) a
6) b
7) b
8) a y c
9) b y c
10) a

Unidad 8: Lab. Acceso a Bases de Datos

Objetivos de Aprendizaje

Al final de esta unidad, usted será capaz de:

- Conectarse a bases de datos en MySQL a través de PHP.
- Implementar una aplicación web que pueda agregar, modificar y eliminar datos de una base de datos en MySQL.

Ejercicios de Laboratorio

Se tiene una Base de Datos en MySQL que almacena información sobre los empleados de una compañía. Ud debe elaborar un script PHP que permita:

1) Insertar nuevos empleados en la BD. Los datos de los empleados se tomarán de un formulario HTML. Debe mostrar un mensaje de éxito en caso de que se pueda realizar la inserción y un mensaje de fallo en caso contrario.
2) Eliminar un empleado específico. Se tomará como entrada el ID del empleado que se desea eliminar. Debe mostrar un mensaje de éxito en caso de que se pueda realizar la eliminación y un mensaje de fallo en caso contrario.
3) Actualizar los datos de un empleado. El dato que se actualizará es el Telf. Se tomará como entrada el ID del empleado que se quiere actualizar.
4) Mostrar la lista de empleados existentes. Los detalles de los empleados se deben mostrar en una tabla. Por cada empleado debe mostrar: ID, Nombre, Telf., Cargo y Departamento.

El siguiente es el script necesario para crear las tablas de empleados.

```
CREATE TABLE EMPLEADOS (
IDEMP INTEGER NOT NULL,
NOMBRE VARCHAR(30),
FECHA_NAC VARCHAR(20),
SEXO VARCHAR(1),
TLF VARCHAR(20),
CEL VARCHAR(20),
DIRECCION VARCHAR(100),
CARGO VARCHAR(30)
DEPARTAMENTO VARCHAR(30)
);
```

Unidad 9: Introducción a la Programación Orientada a Objetos

Objetivos de Aprendizaje

Al final de esta unidad, usted será capaz de:

- Crear Clases y Objetos en PHP.
- Describir la Herencia en PHP.
- Conocer cómo se implementa el Polimorfismo en PHP.
- Crear Clases Abstractas e Interfaces.
- Discutir cómo se realiza la clonación de objetos.

1. Introducción

La programación orientada a objetos ha ganado una significativa importancia en tiempos recientes. La orientación a objetos provee la habilidad de tratar los aspectos de un sistema complejo. Es una metodología de programación avanzada y bastante extendida, en la que los sistemas se modelan creando clases, que representan entes del mundo real que poseen características (atributos) y comportamiento (métodos).

Las clases son definiciones a partir de las cuales se crean objetos. Los objetos son instancias de una clase determinada que adquieren las características y el comportamiento definidos en la clase. Los objetos interactúan unos con otros a través de mensajes.

La programación orientada a objetos permite crear programas de una forma que se asemeje a la realidad. La tendencia actual es que los lenguajes de programación adopten la programación orientada a objetos para desarrollar sistemas.

En la versión 5 de PHP se puede utilizar la programación orientada a objetos como metodología de desarrollo. En PHP 4 también se puede programar bajo esta metodología, pero en PHP 5 hay un nuevo modelo mejorado. El manejo de objetos ha sido rescrito por completo, permitiendo un mejor desempeño y más características.

La programación orientada a objetos cada día es más utilizada y resulta fundamental manejarla para poder desarrollar en casi cualquier lenguaje moderno. En esta unidad se estudiarán algunos conceptos sobre la programación orientada a objetos en PHP. Aunque es un tema bastante amplio, se presentará la sintaxis básica de PHP para utilizar objetos.

2. Definir Clases y Objetos

La definición de una clase comienza con la palabra "`class`", luego se coloca el nombre de la clase. La definición de los atributos y métodos miembros se hace dentro de llaves. Observe el siguiente código:

```
1.  <?php
2.  class ClaseA
3.  {
        // * atributos
4.    public $var1;
5.    private $var2 = 'Valor por defecto';

        // * método
6.    function mostrarVar() {
7.        echo $this->var2;
8.    }
```

```
9.  }
10. ?>
```

En el código anterior, la variable `$this` es una referencia al objeto actual que se está usando.

Las convenciones para nombrar los atributos son las mismas que para nombrar las variables en PHP (véase la *Unidad 2 – Elementos del Lenguaje PHP*). Las convenciones para definir métodos son las mismas que para definir las funciones en PHP (véase la *Unidad 3 – Funciones y Extensiones en PHP*).

En PHP los atributos siempre deben tener un modificador de acceso (`public`, `protected` o `private`). Los atributos pueden ser de cualquier tipo, incluso pueden ser otros objetos.

2.1 Definición de Objetos

Para crear una instancia de una clase, se crea un nuevo objeto y éste es asignado a una variable.

```
<?php
$obj1 = new ClaseA();
$obj2 = $obj1; // $obj2 hace referencia a la misma instancia
?>
```

Cuando se asigna un objeto previamente creado a otra variable, la nueva variable accederá a la misma instancia del objeto que le fue asignado, esto significa que no se realiza una copia del objeto.

Una nueva instancia de un objeto previamente creado, es decir, una copia, puede hacerse clonando al objeto. PHP tiene mecanismos para clonar objetos, que se detallarán más adelante en esta unidad.

2.2 Acceder a atributos y métodos

Para acceder a los atributos y métodos de un objeto se emplea el operador flecha "->", como se muestra a continuación:

```
1.  <?php
2.  /*
3.  $objeto->nombre_atributo;
4.  $objeto->nombre_metodo;
5.  */
6.  $obj1 = new ClaseA();
7.  echo $obj1->var1; // A la variable se le quita el '$'
8.  $obj1->mostrarVar();
9.  ?>
```

Cuando se accede a un miembro dentro de la misma definición de la clase, se utiliza la variable `$this`, que hace referencia a la instancia actual. Por ejemplo:

```
1.  <?php
2.  class ClaseB
3.  {
4.      private $var1;
5.      private $var2;
6.      public function mostrarVar1() {
7.          $this->var1="Hola a todos!";
8.           echo $this->var1;
9.  }
10.     public function mostrarVar2() {
11.         $this->mostrarVar1();
12.         $this->var2="Adios a todos!";
13.          echo $this->var2;
14.     }
15. }
16. ?>
```

Se debe hacer notar que los atributos y métodos miembros sólo se pueden referenciar de acuerdo a las reglas de visibilidad, es decir, de acuerdo a si son públicos privados o protegidos. Esto se estudiará a continuación.

2.3 Visibilidad de Atributos y Métodos

En PHP, se puede definir la visibilidad para los atributos y métodos. Los tipos de visibilidad disponibles son las siguientes:

- **Pública**: Los elementos que tienen esta visibilidad pueden ser accedidos desde cualquier lugar dentro del programa.
- **Protegida**: Los elementos que tienen esta visibilidad limitan el acceso sólo a la clase que los define y a las clases que heredan de ésta.
- **Privada**: Los elementos que tienen esta visibilidad limitan la visibilidad sólo a la clase que los define.

La visibilidad de un atributo o método puede ser definida al anteponerle a la declaración las palabras reservadas: `public`, `protected` o `private`, como se observa a continuación.

```
1.  <?php
2.  class ClasePadre
3.  {
4.      public $varPublic = 'Atributo Público';
5.      protected $varProtected = 'Atributo Protegido';
```

```
6.      private $varPrivate = 'Atributo Privado';
7.
8.      public function imprimirAtributos()
9.      {
10.         echo $this->varPublic;
11.         echo $this->varProtected;
12.         echo $this->varPrivate;
13.     }
14. }
15.
16. $obj1 = new ClasePadre();
17. echo $obj1->varPublic; //Muestra el contenido de la variable
18. echo $obj1->varProtected; // Error
19. echo $obj1->varPrivate;  // Error
20. $obj1->imprimirAtributos(); // Ejecuta la función y muestra
    todos los valores
21. class ClaseHija extends ClasePadre
22. {
23. /* Todos los atributos y métodos de una clase padre la
    hereda clase hija. Se pueden redeclarar en una clase hija,
    los atributos y métodos públicos y protegidos, pero no los
    privados.
24. */
25.     protected $varProtected = 'Variable redefinida';
26.
27.     function imprimirAtributos()
28.     {
29.         echo $this->varPublic;
30.         echo $this->varProtected;
31.         echo $this->varPrivate;
32.     }
33. }
34. $obj2 = new ClaseHija();
35. echo $obj2->varPublic; //Muestra el contenido de la variable
36. echo $obj2->varPrivate; // no definido (no muestra nada)
37. echo $obj2->varProtected; // Error
38. $obj2->imprimirAtributos(); // Muestra: 'Atributo Público'
39. 'Variable redefinida' y <vacío> (no muestra contenido)
40. ?>
```

Los atributos miembros de una clase siempre deben estar definidos con `public`, `private` o `protected`. Si no se define la visibilidad se genera un error.

Los métodos de clase pueden ser definidos con `public`, `private` o `protected`. Los métodos a los que no se les especifique visibilidad son tomados como `public`.

En PHP 5 se pueden definir métodos constructores y destructores para las clases. Observe cómo se realiza esto.

2.4 Constructor de una Clase

Las clases que tienen un método constructor invocan a este método cada vez que se crea un nuevo objeto. Allí se coloca cualquier inicialización que el objeto requiera antes de ser usado. Un constructor de clase se define con la función `__construct()`.

Los métodos constructores se encargan de realizar la inicialización de los objetos. Cuando se crea un objeto, posiblemente se necesite realizar varias tareas de inicialización, por ejemplo, dar valores a los atributos. Los constructores pueden recibir datos, pasados como parámetros, para inicializar los objetos según sea el caso.

```
1.  <?php
2.  class ClaseA {
3.      public $var1;
4.      private $var2 = 'Valor por defecto';
5.
6.  function __construct($arg1){
7.          $this->var1=$arg1;
8.           $this->var2="Valor 2";
9.     }
10. function mostrarVar() {
11.         echo $this->var1;
12.         echo $this->var2;
13.  }
14.  }
15.  $objA = new ClaseA("Valor1"); // Se ejecuta el constructor
16.  $objA->mostrarVar(); // Imprime: 'Valor 1' 'Valor 2'
17.  ?>
```

Con `$this->var1=$arg1`, se le está asignando al atributo `$var1` del objeto que se está construyendo, el valor que contiene el argumento `$arg1`.

En PHP 4, el constructor de una clase es una función que posee el mismo nombre de la clase.

```
1.  <?php
```

```
2.   class ClaseA {
3.       public $var1;
4.       private $var2 = 'Valor por defecto';
5.   function ClaseA() {
6.         $this->var1="Valor 1";
7.          $this->var2="Valor 2";
8.      }
9.   }
10.  $objA = new ClaseA(); // Se ejecuta el constructor
11.  ?>
```

PHP 5, también acepta este tipo de constructor, para mantener la compatibilidad con la versión anterior. Si no se encuentra una función `__construct()`, pero está definido un método con el nombre de la clase (como se muestra en el segmento de código anterior), éste se toma como el constructor por defecto. Si en una clase están presente ambos tipos de constructores, se toma por defecto el que está definido como `__construct()`.

Nota: `$this` hace referencia al objeto sobre el que se está ejecutando el método. En el caso anterior, como se está utilizando dentro del método constructor, `$this` hace referencia al objeto que se está construyendo.

2.5 Destructor de una clase

El método destructor de una clase, si está definido, se invoca cuando todas las referencias a un objeto en particular sean removidas o cuando el objeto sea explícitamente destruido. Un destructor de clase se define con la función `__destruct()`.

```
1.   <?php
2.   class ClaseA {
3.       public $var1;
4.       private $var2 = 'Valor por defecto';

5.   function __construct($arg1) {
6.         $this->var1=$arg1;
7.          $this->var2="Valor 2";
8.      }
9.   function mostrarVar() {
10.         echo $this->var1;
11.         echo $this->var2;
12.     }

13.  function __destruct() {
```

```
14.          print "Se destruye el objeto!";
15.      }
16.  }
17.  $objA = new ClaseA("Valor 1"); // Se ejecuta el constructor
18.  $objA->mostrarVar(); // Imprime: 'Valor 1' 'Valor 2'

     /* Al terminar de utilizar el objeto o al finalizar el
     script se destruye el objeto */
19.  ?>
```

El destructor de una clase es un método que realiza tareas de finalización cuando un objeto deja de existir. Cuando un objeto ya no está referenciado por ninguna variable, ya no tiene sentido que siga almacenado en la memoria, por tanto, el objeto debería destruirse para liberar espacio en memoria. En el momento de su destrucción se llama al método destructor. El objeto se destruye al terminar de utilizar al objeto (cuando ya no se haga referencia a él) o al finalizar el script.

La definición del método destructor es opcional. Sólo se debe crear si se necesita hacer algún procedimiento particular cuando un objeto se elimine de la memoria.

2.6 Definir Métodos "getter" y "setter" en una Clase

Dos de los principios fundamentales de la programación orientada a objetos son el encapsulamiento y el ocultamiento de la información.

El encapsulamiento significa que los atributos de un objeto y los métodos que trabajan con esos atributos se mantienen ambos en una cápsula. La encapsulación es lograda a través del ocultamiento de la información. Al encapsular los datos y las operaciones, el usuario sólo necesita saber cómo acceder a las operaciones, no directamente a los atributos.

Esto quiere decir que los atributos no deberían ser accedidos directamente desde el exterior. Para ello se deben disponer de métodos que manipulen a los atributos. Dos de estos tipos de métodos que se deben definir son aquellos que permiten asignarle un valor a un atributo (setter) y que permitan tomar el valor de un atributo (getter). A continuación un ejemplo:

```
1.  <?php
2.  class ClaseA {
3.      private $var1;
4.      private $var2;
5.     function __construct($arg1, $arg2) {
6.        $this->var1=$arg1;
7.         $this->var2=$arg2;
8.     }
```

```
9.        function set_var1($valor) {
10.          $this->var1=$valor;
11.       }
12.       function get_var1() {
13.          return $this->var1;
14.       }

15.       function set_var2($valor) {
16.          $this->var2=$valor;
17.       }
18.       function get_var2() {
19.          return $this->var2;
20.       }
21.    }
22.    $objA = new ClaseA("Valor 1","Valor 2");
23.    $objA->set_var1("Nuevo valor de var1");
24.    echo $objA->get_var1(); // Imprime: 'Nuevo valor de var1'
25.    ?>
```

2.7 Definir Constantes en una Clase

Dentro de una clase se pueden definir constantes. Las convenciones para nombrar las constantes son las mismas que para nombrar las constantes en PHP (véase la *Unidad 2 – Elementos del Lenguaje PHP*). Se nombran sin utilizar el signo '$'. Tampoco se definen modificadores de acceso para una constante.

Los valores constantes no pueden ser accedidos desde una instancia. Es decir, las constantes definidas dentro de una clase sólo se pueden acceder a través del nombre de la clase o a través de un método que las retorne o muestre. No se puede utilizar el operador flecha (->) para accederlas. Observe el siguiente segmento de código:

```
1.    <?php
2.    class claseA
3.    {
4.        const miConstante = 'Valor constante!';
5.        function mostrarConstante() {
6.            echo  self::miConstante;
7.          // No está pemitido: $this->miConstante
8.        }
9.        function get_miConstante() {
10.           return self::miConstante;
11.       }
12.   }
```

```
13.  $objetoA = new ClaseA();
14.
15.  $objetoA->mostrarConstante();
16.  $var1=$objetoA->get_miConstante();
17.
18.  echo ClaseA::miConstante;  // Se muestra la constante
19.  echo $objetoA->miConstante;  // Error!
20.  echo $objetoA::miConstante; // Error!
21.  ?>
```

En el ejemplo anterior se utiliza el operador de resolución doble dos puntos (::), para acceder a una constante dentro y fuera de la clase. En la siguiente sección se mostrará el uso de este operador.

2.8 Operador de Resolución ::

El operador de resolución, dobles dos puntos (::), también conocido como "Paamayim Nekudotayim", es un símbolo que permite acceder a los atributos o métodos estáticos y las constantes de una clase.

```
1.  <?php
2.  class ClaseA {
3.      const CONST_A = 'A';
4.  }
5.  class ClaseB extends ClaseA {
6.       const CONST_B= 'B';
7.       public static $estaticaB = 'Variable estática';
8.
9.       public static function mostrarConstantes() {
10.        echo parent::CONST_A;
11.        echo self::CONST_B;
12.
13.      public static function mostrarEstaticas() {
14.         echo self::$estaticaB;
15.      }
16. }
17. echo ClaseA::CONST_A; // Acceder a una constante
18. echo ClaseB::CONST_B;
19. echo ClaseB::estaticaB; // Acceder a una variable estática
20. ClaseB::mostrarConstantes(); // Invocar un método estático
21. ClaseB::mostrarEstaticas();
22. ?>
```

Como se puede observar en el segmento de código anterior, cuando se referencian constantes, atributos o métodos estáticos, desde fuera de la definición de la clase se usa el nombre de la clase.

El operador de resolución también se utiliza cuando se necesita tener acceso desde una clase hija a los atributos o métodos de su clase padre. Observe el siguiente segmento de código.

```
1.  <?php
2.  class ClasePadre {
3.       private $varPriv;
4.       protected $varProt;
5.       function __construct($valor) {
6.          $this->$varPriv=$valor;
7.          $this->$varProt=$valor;
8.       }
9.       protected function miFuncion() {
10.         echo "Función del padre";
11.      }
12. }
13.
14. class ClaseHijo extends ClasePadre
15. {
16.      function __construct() {
17.            // Se puede invocar al constructor del padre
18.         parent::__construct("Inicializada desde el padre");
19.         $this->$varProtected="Inicializada desde el hijo";
20.      }
21.
22.     // Se sobrescribe la función
23.      public function miFuncion()
24.      {
25.      // Se puede invocar a la función del padre
26.          parent::miFuncion();
27.          echo "Función del hijo";
28.      }
29. }
30.
31. $objHijo = new ClaseHijo();
32. $objHijo->miFuncion(); // Invocar la función de la ClaseHijo
33. ?>
```

Como se ha observado en los ejemplos anteriores, las palabras reservadas `self` y `parent` son usadas para acceder los miembros o métodos desde dentro de la definición de la clase.

- **self:** Hace referencia a la clase actual, a la clase que define al método.
- **parent:** Hace referencia a la clase padre.

3. Herencia entre Clases

La herencia se refiere a que alguna entidad X comparte la estructura y comportamiento de alguna otra entidad Y. En este contexto, la entidad Y es llamada la "super clase" y la entidad X es llamada "subclase". Algunas veces la entidad Y es referida como el padre y a la entidad X como su hijo.

La herencia es uno de los mecanismos fundamentales de la programación orientada a objetos. Por medio de la herencia, se pueden definir nuevas clases a partir de la declaración de clases existentes. Las clases que heredan incluyen tanto los métodos como los atributos de la clase de la cual heredan. Además, pueden definir sus propios atributos y métodos.

Una clase puede heredar métodos y atributos de otra clase usando en su declaración la palabra `'extends'` es seguida del nombre de la clase de la cual hereda, como se muestra a continuación:

```
1.  <?php
2.  class ClaseBase {
3.      private $var1;
4.      protected $var2;
5.      function __construct($valor) {
6.          $this->$var1=$valor;
7.          $this->$var2=$valor;
8.      }
9.      protected function miFuncion() {
10.         echo "Funcion de la clase Base";
11.     }
12. }
13. class ClaseDerivada extends ClaseBase {
14.     function __construct() {
15.       // Se puede invocar al constructor del padre
16.         parent::__construct("Inicializada desde el
    padre");
17.         $this->$var2="Inicializada desde el hijo";
18.     }
19. // Se puede sobrescribir la función
```

```
20.       public function miFuncion() {
21.      // Se puede invocar a la función del padre
22.           parent::miFuncion();
23.           echo "Función de la clase Derivada";
24.       }
25.  }
26.  $obj = new ClaseDerivada();
27.  $obj->miFuncion(); // Invocar la función de la ClaseDerivada
28.  ?>
```

En PHP no es posible extender de múltiples clases. Una clase puede heredar sólo de una clase base, es decir, PHP no soporta la herencia múltiple.

Los métodos y atributos heredados pueden ser redefinidos en la subclase, declarándolos con el mismo nombre con el que se definieron en la clase padre. Los métodos sólo pueden sobre escribirse si la clase padre no ha definido ese método como final.

Es posible acceder a los métodos o atributos (sólo los atributos estáticos) de la clase base, aunque hayan sido redeclarados, en la clase derivada haciendo referencia a ellos con `parent::`.

3.1 Sobrescritura de Métodos

La sobrescritura de métodos es muy común en mecanismos de herencia, puesto que los métodos que fueron creados para una clase base no necesariamente se deban implementar del mismo modo para sus subclases. Sobrescribir un método significa redefinirlo, es decir, otorgarle otro comportamiento distinto al definido en la clase padre.

Por ejemplo, si se tiene un método llamado `inscribirse()` declarado en una clase padre llamada `Estudiante`, de la cual heredan las subclases `EstudiantePregrado` y `EstudiantePostgrado`, ese método puede ser diferente para una y otra (las inscripciones pueden ser diferentes para los estudiantes de pregrado y los de postgrado). Entonces cada subclase puede implementar su propio método `inscribirse()`.

En PHP sobrescribir un método es muy sencillo, sólo se tiene que volver a declarar el método en la clase hija con el mismo nombre que se definió en la clase padre. Observe un ejemplo.

```
1.   <?php
2.   class Estudiante {
3.   private $nombre;
4.   private $fechaNac;
5.   private $sexo;
6.   protected $IDEstudiante;
```

```
7.    private $inscrito;
8.    function inscribirse(){
9.    $inscrito = true;
10.   }
11.   }
12.   class EstPregrado extends Estudiante {
13.   function inscribirse(){
14.            parent::inscribirse();
15.      echo "Se inscribió el estudiante de pregrado";
16.   }
17.   }
18.   class EstPostgrado extends Estudiante{
19.         private $agnoGraduacion;
20.         private $carreraPregrado;
21.         private $IDEstudiante;
22.
23.          function inscribirse(){
24.            echo "Año de graduación: ", $agnoGraduacion;
25.            echo "Carrera: ", $carreraPregrado;
26.            parent::inscribirse();
27.      }
28.   }
29.   ?>
```

Los atributos también pueden redeclararse en una subclase. Simplemente se coloca el mismo nombre del atributo que está en la super clase y que se quiere redeclarar, tomando en cuenta que sólo se pueden redeclarar en una subclase aquellos atributos que tengan visibilidad `public` o `protected` en la super clase.

3.2 Clases y Métodos Finales

Las clases finales son aquellas de las cuales no se puede heredar. Los métodos finales son aquellos que no se pueden sobrescribir.

Para declarar a una clase o a un método como final, PHP 5 utiliza la palabra reservada `'final'`. Las subclases no pueden sobrescribir un método si en la super clase se usó el prefijo `'final'` en la definición del método. Si la clase en sí misma es definida como `'final'` entonces no puede ser extendida, es decir, no puede tener subclases.

```
1.    <?php
2.    class Estudiante {
3.    private $nombre;
4.    private $fechaNac;
```

```
5.   private $sexo;
6.   protected $IDEstudiante;
7.   private $inscrito;
8.
9.   /* Este método no puede ser sobrescrito */
10.  final public function inscribirse(){
11.  $inscrito = true;
12.  }
13.  }
14.
15.  /* Ninguna otra clase puede heredar de EstPregrado, pues
     está declarada como Final */
16.  final class EstPregrado extends Estudiante {
17.
18.  // * Esta declaración ocasiona un error fatal! *
19.  function inscribirse(){
20.            parent::inscribirse();
21.     echo "Se inscribió el estudiante de pregrado";
22.  }
23.  }
24.
25.  class EstPostgrado extends Estudiante{
26.        private $agnoGraduacion;
27.        private $carreraPregrado;
28.        private $IDEstudiante;
29.
30.  // * Esta declaración ocasiona un error fatal! *
31.         function inscribirse(){
32.           echo "Año de graduación: ", $agnoGraduacion;
33.           echo "Carrera: ", $carreraPregrado;
34.           parent::inscribirse();
35.     }
36.  }
37.  ?>
```

Si se desea declarar una clase final, la palabra reservada `final` debe colocarse antes de la definición de la clase (antes de la palabra `'class'`). Si se desea declarar un método final, la palabra reservada `final` debe colocarse antes de la definición del método (antes de la palabra `'function'` y de cualquier modificador de acceso).

4. Polimorfismo

Cualquier lenguaje de programación orientado a objetos debe soportar el polimorfismo, esto significa que clases diferentes tendrán un comportamiento distinto para la misma operación. Las clases pueden tener los mismos métodos (con el mismo nombre), pero cada una los puede implementar de forma diferente.

Esto se puede comprender mejor con un ejemplo. Suponga que existen dos clases distintas Automóvil y Triciclo. Ambas tienen sus propios métodos de movimiento, éstos tienen diferentes comportamientos, pero su nombre es el mismo.

```
1.  <?php
2.  class Automovil {
3.        function avanzar() {
4.           echo "Avanza el automóvil";
5.        }
6.        function parar() {
7.        echo "Se detiene el automóvil";
8.        }
9.  }
10. class Triciclo {
11.       function avanzar() {
12.          echo "Avanza el triciclo";
13.       }
14.       function parar() {
15.       echo "Se detiene el triciclo";
16.       }
17. }
18. ?>
```

Como se puede observar, ambas clases tienen los mismos métodos. Suponga ahora que se tiene otra clase que controla el movimiento de los vehículos (es aquí donde entra en juego el polimorfismo) que dependiendo del objeto que se trate, el método al que se invoque actuará de una forma u otra.

```
1.  <?php
2.  class Movimiento {
3.        function mover_adelante($obj) {
4.        $obj->avanzar();
5.        }
6.  }
7.  ?>
```

Ahora si se desea mover cualquier vehículo hacia adelante entonces se hace lo siguiente:

```
1.  <?php
2.  $objAuto = new Automovil();
3.  $objTriciclo = new Triciclo();
4.  $objMovimiento = new Movimiento();
5.
6.  // se ejecuta el método avanzar de $objAuto.
7.  $objMovimiento->mover_adelante($objAuto);
8.
9.  // se ejecuta el método avanzar de $objTriciclo.
10. $obj_Movimiento->mover_adelante($objTriciclo);
11. ?>
```

5. Atributos y Métodos Estáticos

Cuando un método o atributo está definido como estático, significa que es un método o atributo de clase, no de objeto. Un método o atributo de clase puede ser usado independientemente de las instancias que se puedan crear. No se necesita instanciar una clase para invocar sus métodos estáticos. Sin embargo, se puede instanciar una clase que posea miembros estáticos, para acceder a los otros miembros que no sean estáticos.

Los atributos y métodos estáticos son accesibles únicamente desde el contexto o ámbito de la clase. Un atributo o método declarado como estático no puede ser accedido a través de una instancia. Además no puede ser redefinido o sobrescrito en las subclases de la super clase donde esté definido. En PHP, para declarar un miembro como estático se utiliza la palabra reservada `'static'`, que debe estar después de la declaración de visibilidad.

Como los métodos y atributos estáticos son accesibles sin que se cree una instancia, la pseudo variable `$this` no se puede usar dentro de los métodos declarados como estáticos.

Los elementos estáticos tampoco pueden ser accedidos a través de un objeto usando el operador de flecha ->. Se utiliza `self::`, para referirse a miembros estáticos dentro de una clase. Se utiliza `parent::`, para referirse a miembros estáticos de la clase padre o super clase. Observe el siguiente segmento de código:

```
1.  <?php
2.  class Vehiculo {
3.        public static $tipoVehiculo="Automotor";
4.        protected $capacidadMotor;
5.
6.        static function verTipoVehiculo() {
7.              echo "Sólo vehículos automotores";
8.              return self::$tipoVehiculo
```

```
9.          }
10.
11.   }
12.   class Automovil extends Vehiculo{
13.   private $capacidadMotor;
14.   // private $tipoVehiculo; // * Error!
15.
16.         function consultarInformacion() {
17.             echo "Capacidad:", $this->capacidadMotor;
18.             echo "Tipo de Vehículo:
      ",parent::tipoVehiculo;
19.         }
20.   }
21.   /* Para utilizar atributos y métodos estáticos desde fuera
      de la definición de la clase, no se necesita utilizar una
      instancia */
22.   Vehiculo::$tipoVehiculo; // Acceder a un atributo estático
23.   Vehiculo::verTipoVehiculo(); // Invocar un método estático
24.   /* Se puede instanciar una clase con miembros estáticos,
      para acceder a aquellos miembros que no sean estáticos. */
25.   $objV1 = new Vehiculo();
26.   ?>
```

6. Clases Abstractas

En ocasiones, cuando se utiliza la herencia en la programación orientada a objetos, se necesita declarar clases que quizás no tengan una definición completa al principio, simplemente se define para empezar una jerarquía de clases, luego las clases que heredan completan o concretan la definición.

Una clase abstracta es aquella que posee métodos abstractos. Los métodos abstractos son aquellos que sólamente se declaran dentro de la clase, pero no tienen implementación. Es decir, estos métodos no incluyen codificación, sino que simplemente se declaran, dejando para las clases que hereden la tarea de codificarlos.

Si una clase tiene por lo menos un método abstracto, forzosamente debe declararse como abstracta. Las clases abstractas no pueden instanciarse. Es decir, no se puede crear objetos a partir de ellas.

Para entender este concepto, suponga que se tiene una clase `Vehiculo`, de la cual heredan la clase `Automóvil` y la clase `Motocicleta`. Un vehículo puede arrancar, frenar, acelerar, retroceder, etc. Pero cada tipo de vehículo lo hace a su manera, es decir, una moto arranca, frena y acelera de forma distinta a la del automóvil. Es decir, no existe un vehículo donde se puedan definir esas operaciones de forma general para

todos los tipos de vehículos, sino más bien existen distintos tipos de vehículos concretos que son los que implementan las operaciones, según lo requieran.

En conclusión, es posible tener un conjunto de objetos que tienen unas características y funcionalidades comunes, pero que difieren en la manera de llevarlas a cabo. Para esto sirve la abstracción.

En PHP 5 se pueden utilizar métodos y clases abstractos. Aplican los siguientes principios, algunos de los cuales ya se han mencionado:

- No se permite crear una instancia de una clase que ha sido definida como abstracta.
- Cualquier clase que contenga por lo menos un método abstracto debe ser abstracta.
- Los métodos abstractos simplemente tienen declaración, no implementación.
- Cuando se hereda de una clase abstracta, todos los métodos abstractos en la declaración de la clase padre, deben de ser implementados por la clase hijo o en algún nivel de la jerarquía de herencia.
- Una subclase debe definir los métodos abstractos que ha heredado y que va a implementar, con la misma o menor visibilidad. Es decir, si el método abstracto es definido como `protected` en la clase padre, la implementación de la función debe ser definida como `protected` o `public` en la clase hijo.

En PHP, para crear clases y métodos abstractos se utiliza la palabra reservada `'abstract'`. Observe el siguiente ejemplo:

```
1.  <?php
2.  abstract class Vehiculo {
3.      public static $tipoVehiculo="Automotor";
4.      protected $capacidad;
5.  /* Las clases que heredan deben definir estos métodos. Si
    no lo hacen serán también abstractas. */
6.   abstract protected function arrancar();
7.   abstract protected function acelerar($velocidad);
8.   abstract protected function frenar();
9.  / * Las clases abstractas pueden tener métodos concretos. */
10.         function __construct($valor){
11.          $this->capacidad=$valor;
12. }
13.         public function mostrarInfo() {
14.         echo "Tipo de vehiculo: ",self::$tipoVehiculo,"<br>";
15.         echo "Capacidad del motor: ",$this->capacidad,"<br>";
16.     }
17. }
```

```
18.
19.  class Automovil extends Vehiculo {
20.      public function arrancar() {
21.          echo "Arranca el automovil <br>";
22.      }
23.
24.      public function acelerar($velocidad) {
25.          echo "Acelera el automovil a: $velocidad MPH <br>";
26.      }
27.      public function frenar() {
28.          echo "Se detiene el automovil <br>";
29.      }
30.  }
31.
32.  class Motocicleta extends Vehiculo {
33.      public function arrancar() {
34.          echo "Arranca la motocicleta <br>";
35.      }
36.
37.      public function acelerar($velocidad) {
38.      echo "Acelera la motocicleta a: $velocidad MPH <br>";
39.      }
40.   public function frenar() {
41.          echo "Se detiene la motocicleta <br>";
42.      }
43.  }
44.
45.  $carro = new Automovil(2.0);
46.  $carro->mostrarInfo();
47.  $carro->arrancar();
48.
49.  $moto = new Motocicleta(0.9);
50.  $moto->mostrarInfo();
51.  $moto->arrancar();
52.  ?>
```

Nota: Si una subclase no implementa alguno de los métodos abstractos que hereda de la clase padre, se convierte en una clase abstracta y debe declararse como tal. Para que una subclase que hereda de una clase abstracta sea concreta, debe implementar todos los métodos abstractos de la clase padre.

7. Interfaces

Las interfaces son mecanismos que se usan comúnmente en la programación orientada a objetos. Dentro de una interface se declaran las funciones que eventualmente serán implementadas por una o varias clases. Cuando una clase codifica las funciones que están definidas en una interface se dice que la clase implementa a la interface.

Puede suceder que existan objetos que no pertenezcan a la misma jerarquía de herencia, pero que deban realizar acciones comunes. Por ejemplo, se puede encender o apagar un TV, un bombillo, un carro, etc y si se considera que esos elementos (la TV, el bombillo, el carro) son objetos, es posible darse cuenta que no heredan de una clase común. Por eso las interfaces son útiles, para especificar métodos que pueden ser desarrollados por una o más clases, aunque éstas no tengan otras características en común.

Las interfaces permiten especificar los métodos que una clase debe implementar, sin tener que definir cómo son manejados esos métodos.

En PHP, las interfaces son definidas usando la palabra reservada `'interface'`, pero los métodos que contiene no están codificados, sólo declarados. Por definición todos los métodos en una interface deben ser públicos.

Observe cómo se declara una interfaz.

```
<?php
interface iInterruptor
{
    public function encender();
    public function apagar();
}
?>
```

Para indicar que una clase implementa una interface se utiliza la palabra reservada `'implements'` . Todos los métodos en la interface deben ser codificados dentro de la clase que la implementa. Si se deja uno o más métodos sin codificar, se genera un error. Las clases pueden implementar más de una interface. En la declaración se separa cada interface con comas.

Observe cómo se declara una clase que implementa una interface.

```
1.  <?php
2.  interface iInterruptor
3.  {
4.      public function encender();
5.      public function apagar();
6.  }
7.
```

```
8.   class Automovil implements iInterruptor
9.   {
10.
11.      public function encender()
12.      {
13.          echo "Se enciende el automovil";
14.      }
15.
16.      public function apagar()
17.      {
18.          echo "Se apaga el automovil";
19.      }
20.  }
21.
22.  class Televisor implements iInterruptor
23.  {
24.
25.      public function encender()
26.      {
27.          echo "Se enciende el televisor";
28.      }
29.      public function apagar()
30.      {
31.          echo "Se apaga el televisor";
32.      }
33.     public function cambiar($canal)
34.      {
35.          echo "Se cambia el televisor al $canal";
36.      }
37.  }
38.
39.  $carro = new Automovil();
40.  $carro->encender();
41.
42.  $tv = new Televisor();
43.  $tv->encender();
44.  ?>
```

Una clase que implementa una interfaz puede poseer otros métodos que no estén definidos en la interface. Fíjese en el ejemplo anterior donde la clase `Televisor` tiene un método llamado `cambiar()`, éste no está declarado en la interface `iInterruptor`.

Sin embargo, una clase debe implementar todos los métodos de la(s) interface(s) que implementa. Si esto no se cumple, genera un error.

8. Clonación de Objetos

En PHP se pueden clonar los objetos. Clonar un objeto significa crear una copia de ese objeto. Cuando se clona un objeto se crea una réplica independiente, es decir, aunque ambos objetos, el original y el clon, sean exactamente iguales, ocupan diferentes espacios de memoria.

Esto no ocurre cuando se asigna directamente un objeto a otro (`$objeto1=$objeto2`), en este caso ambos objetos son iguales y hacen referencia al mismo espacio de memoria, es decir, si se modifica `$objeto1`, `$objeto2` también se modifica, porque ambas variables referencian al mismo objeto.

En PHP, para crear una réplica de un objeto se usa la sentencia `'clone'`. También se puede definir el método `__clone()`, dentro de una clase, para realizar alguna acción específica en caso de que una instancia de esa clase sea clonada. Si se define el método `__clone()`, ese método será llamado en el nuevo objeto clonado. En ese método se pueden realizar acciones para cambiar las propiedades en el clon. Cuando se utiliza la sentencia `'clone'`, se invoca al método `__clone()` del objeto, si es posible. Un método `__clone()` no puede ser llamado directamente. Este método tampoco acepta parámetros.

A continuación se muestra cómo trabaja la clonación de objetos en PHP.

```
1.  <?php
2.  class Estudiante {
3.  private $idEst;
4.  private $nombre;
5.  private $semestre;
6.      function __construct($id, $nom, $sem){
7.        $this->idEst=$id;
8.        $this->nombre=$nom;
9.        $this->semestre=$sem;
10.     }
11.     function mostrar(){
12.       echo "$this->idEst, $this->nombre, $this-
    >semestre";
13.     }
14.     function asignar($id, $nom, $sem) {
15.       $this->idEst=$id;
16.       $this->nombre=$nom;
17.       $this->semestre=$sem;
```

```
18.     }
19.
20.  /* El método __clone() se ejecuta automáticamente cuando
     utiliza la sentencia clone */
21.     function __clone() {
22.        $this->idEst="00";
23.        $this->semestre=0;
24.     }
25.  }
26.  $estudiante1 = new Estudiante("10","Pedro",1);
27.  $estudiante2 = new Estudiante("20","María",2);
28.
29.  $est1 = $estudiante1; // Se asigna la referencia al objeto
30.  $est2 = clone $estudiante2; // Se crea una copia del objeto
31.
32.  $estudiante1->mostrar(); // Se muestra: '10, Pedro, 1'
33.  $est1->mostrar();  // Se muestra: '10, Pedro, 1'
34.
35.  // Se cambian los atributos usando la variable $est1
36.  $est1->asignar("30","Pablo",3);
37.
38.  $est1-> mostrar();  // Se muestra: ' 30, Pablo, 3'
39.  $estudiante1->mostrar(); // Se muestra: ' 30, Pablo, 3'
40.
41.  $estudiante2->mostrar(); // Se muestra: '20, María, 2'
42.  $est2->mostrar(); // Se muestra: '00, María, 0'
43.
44.  $est2->asignar("40","Josefina",4);
45.
46.  $estudiante2->mostrar(); // Se muestra '20, María, 2'
47.  $est2->mostrar(); // Se muestra '40, Josefina, 4'
48.  ?>
```

En el ejemplo anterior se definió el método `__clone()` para la clase `Estudiante`. Por lo tanto, la nueva copia del objeto será modificada de acuerdo a lo que esté especificado en ese método. Si no se hubiese definido ese método, se crea una copia exacta del objeto, sin ningún cambio.

9. Uso de la Función `include()`

Para lograr reutilizar un conjunto de clases en distintos scripts PHP, se puede crear cada una de éstas (o un grupo de ellas) en un archivo aparte (con la extensión `.php` o `.inc`). Luego, estos archivos se incluyen en los scripts que lo requieran, utilizando la función `include()`. Observe los siguientes segmentos de código:

```
/* Código correspondiente al script "classAutomovil.php".
Allí se implementa la clase Automovil */
1.  <?php
2.  class Automovil {
3.        function avanzar() {
4.           echo "Avanza el automóvil";
5.        }
6.        function parar() {
7.        echo "Se detiene el automóvil";
8.        }
9.  }?>
10.
11. /* Código correspondiente al script "classTriciclo.php".
    Allí se implementa la clase Triciclo */
12. <?php
13. class Triciclo {
14.       function avanzar() {
15.          echo "Avanza el triciclo";
16.       }
17.       function parar() {
18.       echo "Se detiene el triciclo";
19.       }
20. }
21. ?>
22.
23. /* Código correspondiente al script "classMovimiento.php".
    Allí se implementa la clase Movimiento */
24. <?php
25. class Movimiento {
26.       function mover_adelante($obj) {
27.       $obj->avanzar();
28.       }
29. }
```

```
30.  ?>
31.  /* Código correspondiente al script "principal.php" */
32.  <?php
33.  include("classAutomovil.php");
34.  include("classTriciclo.php");
35.  include("classMovimiento.php");
36.
37.  $objAuto = new Automovil();
38.  $objTriciclo = new Triciclo();
39.  $objMovimiento = new Movimiento();
40.  $objMovimiento->mover_adelante($objAuto);
41.  $obj_Movimiento->mover_adelante($objTriciclo);
42.  ?>
```

La función `include("archivo")` incluye y evalúa el archivo especificado dentro del script actual. Los elementos que se encuentren en el archivo incluido (variables, funciones, clases, objetos, etc) estarán disponibles para el script que lo incluya.

En PHP, la función `include()` se puede utilizar tanto en la programación orientado a objetos o cuando se programa de forma estructurada.

Ahora que ya se han discutido algunos fundamentos de la Programación Orientada a Objetos en PHP, se presenta un ejemplo, donde se han utilizado algunos de los conceptos presentados en esta unidad.

Ejemplo 9.1

En el siguiente ejemplo el usuario debe ingresar una fecha (día, mes, año). Una vez enviados estos datos, se genera un calendario para el mes del año especificado, resaltando el día elegido.

Para realizar este ejemplo se crea una página html (`ejemplo-9.1.html`), que contiene un formulario para ingresar el día, mes y año. También se crea un script PHP llamado `calendario.php` que se ejecuta cuando el usuario envía los datos del formulario. Este script se encarga de recibir estos datos e invocar los métodos de las clases necesarias para mostrar el calendario. Para poder generar el calendario se cuenta con la clase Calendario (implementada en el archivo `classCalendario.php`) que es la encargada de crear e imprimir el calendario y con la clase Formato (implementada en el archivo `classFormato.php`), que se encarga de los aspectos de visualización del calendario.

El código HTML comienza aquí...

```
1.  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
2.  <HTML>
3.  <HEAD><TITLE>..:: Calendario ::..</TITLE>
```

```
4.  <SCRIPT language="JavaScript">
5.  <!--
6.  function agregar(slctDia,slctMes,slctAgno)
7.  {
8.  for (i=1; i<=31;i++){
9.  var elemD = document.createElement("OPTION");
10. elemD.text = i;
11. elemD.value = i;
12. slctDia.options.add(elemD);
13. }
14. var mes = new Array(' ','Enero', 'Febrero', 'Marzo','Abril',
    'Mayo','Junio','Julio','Agosto','Septiembre','Octubre','Novi
    embre','Diciembre');
15.
16. for (i=1; i<=12;i++){
17. var elemM = document.createElement("OPTION");
18. elemM.text = mes[ i] ;
19. elemM.value = i;
20. slctMes.options.add(elemM);
21. }
22. for (i=1900; i<=2100;i++){
23. var elemA = document.createElement("OPTION");
24. elemA.text = i;
25. elemA.value = i;
26. slctAgno.options.add(elemA);
27. }
28. } // fin funcion
29. //-->
30. </SCRIPT>
31. </HEAD>
32. <BODY onLoad="agregar(document.formFecha.dia,
    document.formFecha.mes, document.formFecha.agno)" >
33. <H2 align="center">..:: Calendario ::..</H2>
34. <HR>
35. <FORM action="calendario.php" method="get" name="formFecha">
36.   <TABLE border="0" align="center">
37.     <TR>
38.       <TD width="146"> DIA:    
39.     <SELECT name="dia">
40.     <OPTION value="0">-- Dia --</OPTION>
41.        </SELECT>
```

```
42.      </TD>
43.       </TR>
44.       <TR>
45.         <TD>MES:   
46.           <SELECT name="mes">
47.          <OPTION value="0">-- Mes --</OPTION>
48.           </SELECT>
49.      </TD>
50.       </TR>
51.       <TR>
52.         <TD>A&Ntilde;O:   
53.           <SELECT name="agno">
54.          <OPTION value="0">-- Año --</OPTION>
55.           </SELECT>
56.      </TD>
57.       </TR>
58.       <TR>
59.         <TD><INPUT type="submit" name="Submit" value="Mostrar
     Calendario"></TD>
60.       </TR>
61.     </TABLE>
62.    </FORM>
63.  </BODY>
64.  </HTML>
```

El código HTML termina aquí

El código anterior corresponde a la página ejemplo-9.1.html. A continuación se muestra el código correspondiente a `calendario.php`.

El código PHP comienza aquí...

```
1.   <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
2.   <HTML>
3.   <HEAD>
4.   <TITLE>..:: Calendario ::..</TITLE>
5.   </HEAD>
6.   <BODY>
7.   <?php
8.   include("classCalendario.php");
9.
10.      $año=$_REQUEST["agno"];
```

```
11.     $mes=$_REQUEST[ "mes"] ;
12.     $dia=$_REQUEST[ "dia"] ;
13.  $meses=array(' ','Enero', 'Febrero', 'Marzo','Abril',
     'Mayo', 'Junio','Julio','Agosto', 'Septiembre',
     'Octubre','Noviembre','Diciembre');
14.
15.  echo "<H1 align='center'>..:: Calendario del mes de ",
     $meses[ $mes] ," ::..</H1>";
16.  echo "<HR>";
17.
18.  $cc = new Calendario($año,$mes,$dia);
19.  $cc->imprimir_calendario();
20.
21.  echo "<HR>";
22.  // * Imprimir el Calendario con otro formato
23.  $cc->formato_cal->setNumeros("Courier", "4", "#FF0000",
     "FFFF00");
24.
25.  $cc->formato_cal->setTitulo("Courier", "4", "#FFFFFF",
     "#FF0000");
26.
27.  $cc->imprimir_calendario();
28.  ?>
29.  </BODY>
30.  </HTML>
```

El código PHP termina aquí

Note que en el script anterior se ha incluido al script `classCalendario.php`. Allí se encuentra la implementación de la clase Calendario. El siguiente código corresponde a `classCalendario.php`.

El código PHP comienza aquí...

```
1.   <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
2.   <?php
3.   include("classFormato.php");
4.   class Calendario
5.   {
6.   private $dia;
7.   private $mes;
8.   private $año;
9.   private $dias = array();
```

```
10.  private $meses = array();
11.  public $formato_cal;
12.
13.  function __construct($a=0, $m=0, $d=0){
14.      $this->setFecha($a, $m, $d);
15.      $this->dias = array (1=>"Lun", "Mar", "Mie", "Jue",
     "Vie", "Sáb", "Dom");
16.      $this->meses = array (1=>"ENERO", "FEBRERO",    "MARZO",
     "ABRIL", "MAYO", "JUNIO", "JULIO", "AGOSTO","SEPTIEMBRE",
     "OCTUBRE", "NOVIEMBRE", "DICIEMBRE");
17.
18.          $this->formato_cal = new Formato();
19.          $this->formato_cal->setNumeros();
20.          $this->formato_cal->setTitulo();
21.          $this->formato_cal->setTabla();
22.  }
23.
24.  function get_dia() {
25.          return $this->dia;
26.  }
27.  function set_dia($valor) {
28.          $this->dia=$valor;
29.  }
30.
31.  function get_mes() {
32.          return $this->mes;
33.  }
34.  function set_mes($valor) {
35.          $this->mes=$valor;
36.  }
37.
38.  function get_año() {
39.          return $this->año;
40.  }
41.  function set_año($valor) {
42.          $this->año=$valor;
43.  }
44.  function setFecha($año, $mes, $dia) {
45.
46.          if ($año >=1 AND $año <= 2500) {
47.                  $this->set_año($año);
```

```
48.         } else {
49.                 $this->set_año(intval(date("Y")));
50.         }
51.
52.         if ($mes >= 1 AND $mes <= 12) {
53.                 $this->set_mes($mes);
54.         } else {
55.                 $this->set_mes(intval(date("m")));
56.         }
57.
58.       if (checkdate($this->mes, $dia, $this->año)) {
59.                 $this->set_dia($dia);
60.         } else {
61.                 $this->set_dia(intval(date("d")));
62.         }
63.
64.   } // Fin setFecha()
65.
66.   function imprimir_calendario()
67.   {
68.           /*  Calculamos cuantos días tiene el mes  */
69.         $ultimodia = 31;
70.         while(!checkdate($this->mes, $ultimodia, $this-
      >año)){
71.         $ultimodia -= 1; }
72.
73.         /* Se calcula el número de día de la semana del
      primer día del mes seleccionado */
74.   $DiaSemanaPrimerDiaMes =
75.   date('w',mktime(0,0,0,$this->mes,1,$this->año));
76.
77.         if ($DiaSemanaPrimerDiaMes == 0 ) {
78.                 $DiaSemanaPrimerDiaMes = 7 ;
79.         }
80.
81.   echo "<DIV align='", $this->formato_cal->get_alineacion(),
      "'>\n";
82.
83.   /*  Se crea la tabla del Calendario del mes dado */
84.   echo "<TABLE border='",$this->formato_cal->get_borde(),
85.       "' cellpadding='0' cellspacing='0'>\n";
```

```
86.         // Se imprime el Mes y el Año"
87.    $alto_tit = $this->formato_cal->get_col_alto() + 10;
88.    echo "<TR> \n";
89.    echo "<TD colspan='7' align='center' bgcolor='",
90.   $this->formato_cal->get_tit_fondo(),"' height='$alto_tit'>";
91.
92.    echo "<B><FONT face='",$this->formato_cal-
      >get_tit_letra(),
93.   "' size='",$this->formato_cal->get_tit_tamaño(),
94.   "' color='",$this->formato_cal-     >get_tit_color(),"'>";
95.    echo $this->meses[ $this->mes] ." de ".$this->año.
96.    "</FONT></B></TD>\n";
97.    echo "</TR> \n";
98.
99.         // nombre de los días de la semana
100.   echo "<TR> \n";
101.
102.  for ($i = 1; $i <= count($this->dias); $i++) {
103.        echo "<TD align='center' bgcolor='",
104.  $this->formato_cal->get_tit_fondo(),
105.  "' height='",$this->formato_cal->get_col_alto(),
106.   "' width='",$this->formato_cal->get_col_ancho(),
107.   "'>\n";
108.
109.  echo "<B><FONT face='",
110.  $this->formato_cal->get_tit_letra(),
111.  "' size='",$this->formato_cal->get_tit_tamaño(),
112.  "' color='",$this->formato_cal->get_tit_color(),
113.  "'>";
114.        echo $this->dias[ $i] ,"</FONT></B></TD>\n";
115.    }
116.
117.     echo "</TR> \n";
118.
119.        /*  Se imprimen celdas vacías hasta el día de la
      semana del primer día  */
120.   echo "<TR> \n";
121.   $diasemana = 1;
122.   while ($diasemana < $DiaSemanaPrimerDiaMes)  {
123.        echo "<TD align='center' bgcolor='",
```

```
124.       $this->formato_cal->get_num_fondo(),
125.       "' height='",$this->formato_cal->get_col_alto(),
126.       "' width='",$this->formato_cal->get_col_ancho(),
127.       "'>  </TD>\n";
128.         $diasemana  +=  1;
129.    }
130.
131.  for($dia = 1; $dia  <=  $ultimodia; $dia++)  {
132.
133.       echo "<TD align='center' bgcolor='",
134.            $this->formato_cal->get_num_fondo(),
135.            "' height='",$this->formato_cal->get_col_alto(),
136.     "' width='",$this->formato_cal-
      >get_col_ancho(),"'>";
137.
138.       echo "<FONT face='",
139.             $this->formato_cal->get_num_letra(),
140.  "' size='",$this->formato_cal->get_num_tamaño(),
141.   "' color='",$this->formato_cal-
      >get_num_color(),"'>";
142.
143.  // * Si es el día actual se resalta
144.              if ($dia==$this->dia)
145.                echo "<B> <U>$dia </U></B></FONT></TD>\n";
146.                else
147.                echo "$dia </FONT></TD>\n";
148.
149.              if ($diasemana  ==  7 AND $dia != $ultimodia)
150.              {
151.                 echo "</TR>\n<TR>\n";
152.                   $diasemana  =  1;
153.                } elseif ($diasemana  ==  7) {
154.                      echo "</TR>\n<TR>\n";
155.           } else {
156.                      $diasemana  +=  1;
157.                }
158.     }
159.                /*  Completamos la semana con celdas vacías */
160.          if ($diasemana < 7) {
161.      while($diasemana  <= 7 )  {
```

```
162.       echo "<TD align='center' bgcolor='",
163.            $this->formato_cal->get_num_fondo(),"' height='",
164.       $this->formato_cal->get_col_alto(),
165.        "' width='",$this->formato_cal->get_col_ancho(),
166.        "'> </TD>\n";
167.            $diasemana  +=  1;
168.         }
169.            echo "</TR>\n";
170.         }
171.  echo "</TABLE>\n";
172.  echo "</DIV>\n";
173.
174.  }  //Fin imprimir_calendario()
175.  }  //Fin Calendario
176. ?>
```

El código PHP termina aquí

Observe que en el script anterior se ha incluido al script `classFormato.php`. Allí se encuentra la implementación de la clase Formato. El siguiente código corresponde a `classFormato.php`.

El código PHP comienza aquí...

```
1.   <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
2.   <?php
3.   class Formato
4.   {
5.    private $num_letra;
6.    private $num_tamaño;
7.    private $num_color;
8.    private $num_fondo;
9.
10.   private $tit_letra;
11.   private $tit_tamaño;
12.   private $tit_color;
13.   private $tit_fondo;
14.
15.   private $alineacion;
16.   private $borde;
17.   private $col_ancho;
18.   private $col_alto;
```

```
19.
20.    function get_num_letra() {
21.          return $this->num_letra;
22.    }
23.    function set_num_letra($valor) {
24.          $this->num_letra=$valor;
25.    }
26.
27.    function get_num_tamaño() {
28.          return $this->num_tamaño;
29.    }
30.    function set_num_tamaño($valor) {
31.          $this->num_tamaño=$valor;
32.    }
33.
34.    function get_num_color() {
35.          return $this->num_color;
36.    }
37.    function set_num_color($valor) {
38.          $this->num_color=$valor;
39.    }
40.
41.    function get_num_fondo() {
42.          return $this->num_fondo;
43.    }
44.    function set_num_fondo($valor) {
45.          $this->num_fondo=$valor;
46.    }
47.
48.    function get_tit_letra() {
49.          return $this->tit_letra;
50.    }
51.    function set_tit_letra($valor) {
52.          $this->tit_letra=$valor;
53.    }
54.
55.    function get_tit_tamaño() {
56.          return $this->tit_tamaño;
57.    }
58.    function set_tit_tamaño($valor) {
```

```
59.         $this->tit_tamaño=$valor;
60.     }
61.
62.     function get_tit_color() {
63.         return $this->tit_color;
64.     }
65.     function set_tit_color($valor) {
66.         $this->tit_color=$valor;
67.     }
68.
69.     function get_tit_fondo() {
70.         return $this->tit_fondo;
71.     }
72.     function set_tit_fondo($valor) {
73.         $this->tit_fondo=$valor;
74.     }
75.
76.     function get_alineacion() {
77.         return $this->alineacion;
78.     }
79.     function set_alineacion($valor) {
80.         $this->alineacion=$valor;
81.     }
82.
83.     function get_borde() {
84.         return $this->borde;
85.     }
86.     function set_borde($valor) {
87.         $this->borde=$valor;
88.     }
89.
90.     function get_col_ancho() {
91.         return $this->col_ancho;
92.     }
93.     function set_col_ancho($valor) {
94.         $this->col_ancho=$valor;
95.     }
96.
97.     function get_col_alto() {
98.         return $this->col_alto;
```

```
99.   }
100.  function set_col_alto($valor) {
101.        $this->col_alto=$valor;
102.  }
103.
104.  function setNumeros($letra= "Arial", $tamaño = "2", $color
      = "#FFFF00", $fondo = "#AA6F00") {
105.
106.        $this->set_num_letra($letra);
107.        if (is_int($tamaño)) {
108.              $this->set_num_tamaño(strval($tamaño));
109.        } else {
110.              $this->set_num_tamaño($tamaño);
111.        }
112.        $this->set_num_color($color);
113.        $this->set_num_fondo($fondo);
114.
115.  } //Fin function setNumeros
116.
117.  function setTitulo($letra= "Arial", $tamaño = "2", $color =
      "#FFFFFF", $fondo = "0000FF") {
118.
119.        $this->set_tit_letra($letra);
120.        if (is_int($tamaño)) {
121.              $this->set_tit_tamaño(strval($tamaño));
122.        } else {
123.              $this->set_tit_tamaño($tamaño);
124.        }
125.        $this->set_tit_color($color);
126.        $this->set_tit_fondo($fondo);
127.
128.  } //FIN function setTitulo
129.
130.
131.  function setTabla($alin = "center", $borde=1, $ancho=35,
      $alto=30) {
132.
133.        $this->set_alineacion($alin);
134.        $this->set_borde(strval($borde));
135.        $this->set_col_ancho(strval($ancho));
136.        $this->set_col_alto(strval($alto));
```

```
137.
138.  }
139. }
140. ?>
```

El código PHP termina aquí

A continuación se muestra el funcionamiento de este ejemplo. La Figura 9.1 muestra el formulario y la Figura 9.2 muestra el resultado de ejecutar el script `calendario.php`.

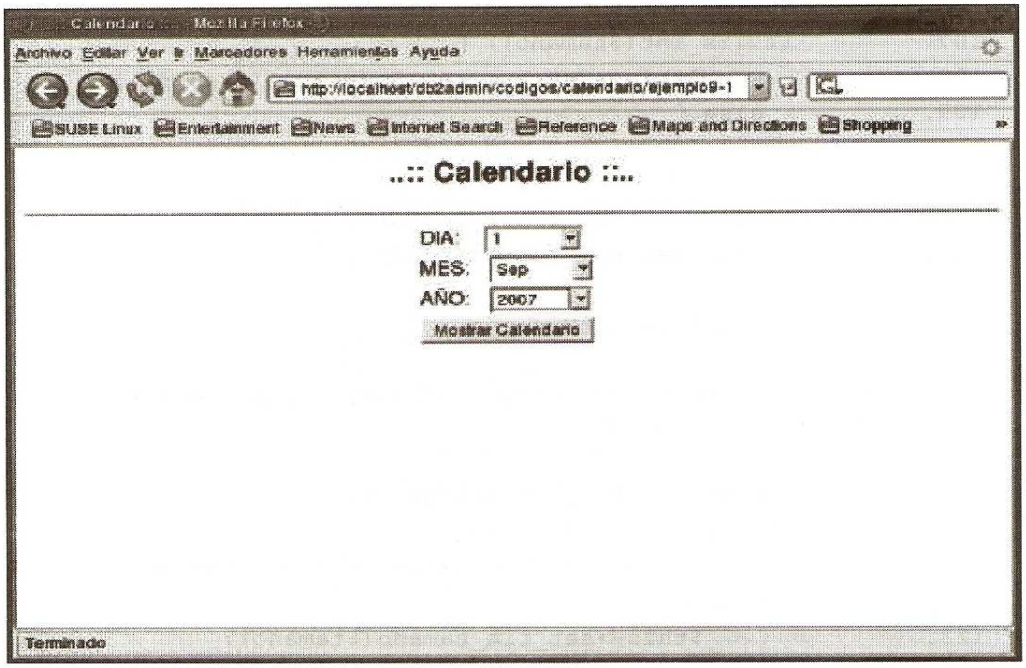


Figura 9.1: Formulario para Ingresar los Datos del ejemplo-9.1.html

Figura 9.2: Resultado de Ejecutar calendario.php

Existen otras características avanzadas relacionadas a la programación orientada a objetos en PHP, que no se han discutido en esta unidad. El objetivo ha sido conocer sólo algunos de los principios de programación orientada a objetos que posee PHP, que permitan crear scripts sencillos con esta forma de programación.

Resumen

Ahora que ha completado esta unidad, usted debe ser capaz de:

- Crear Clases y Objetos en PHP.
- Describir la Herencia en PHP.
- Conocer cómo se implementa el Polimorfismo en PHP.
- Crear Clases Abstractas e Interfaces.
- Discutir cómo se realiza la clonación de objetos.

Unidad 9: Examen de Autoevaluación

1) ¿Cómo se crea un objeto de una clase?
   a) $objeto = new class Clase();
   b) new Clase = $objeto;
   c) $objeto = new Clase();
   d) $objeto = Clase;

2) ¿Cuál operador sirve para acceder los atributos y métodos de un objeto?
   a) Operador dos puntos (::)
   b) Operador ampesand (&)
   c) Operador punto (.)
   d) Operador flecha (->)

3) ¿Cuáles de las siguientes opciones son válidas para definir un método constructor en PHP?

a)
```
<?php
  class Persona {
  private $nombre;
  function __construct($valor) {
  $this->nombre=$valor;
  }
} ?>
```

b)
```
<?php
 class Persona {
 private $nombre;
 function Persona($valor) {
  $this->nombre=$valor;
  }
} ?>
```

c)
```
<?php
  class Persona {
  private $nombre;
```

```
function init($valor) {
  $this->nombre=$valor;
  }
 } ?>
```

d) En PHP no se pueden definir el método constructor

4) La seudo variable $this hace referencia objeto sobre el que se está ejecutando el método, es decir, hace referencia a la instancia actual.
   a) Verdadero
   b) Falso

5) ¿Para qué sirve el operando dos puntos (::)?
   a) Para asignar valores a atributos de un objeto
   b) Permite acceder a los atributos o métodos estáticos y las constantes de una clase
   c) Permite acceder desde una clase hija a los atributos o métodos de su clase padre
   d) Ese operador no existe en PHP

6) ¿Qué palabra reservada se utiliza para indicar que una clase hereda de otra?
   a) implements
   b) interface
   c) clone
   d) extends

7) Un método declarado como "`final`" se puede sobrescribir.
   a) Verdadero
   b) Falso

8) En PHP no se pueden definir clases ni métodos estáticos.
   a) Verdadero
   b) Falso

9) ¿Cuáles de los siguientes principios aplican sobre las clases y métodos abstractos?
   a) Todos los métodos abstractos en la declaración de la clase padre deben de ser implementados por la clase hijo, para que ésta no sea abstracta.

b) Cualquier clase que contenga por lo menos un método abstracto debe ser abstracta.
c) Los métodos abstractos simplemente tienen declaración, no implementación
d) No se permite crear una instancia de una clase que ha sido definida como abstracta.

10) En PHP, para crear una réplica de un objeto se usa la sentencia `'clone'`
a) Verdadero
b) Falso

Unidad 9: Respuestas al Examen de Autoevaluación

1) c
2) d
3) a y b
4) a
5) b y c
6) d
7) b
8) b
9) a, b, c y d
10) a

Unidad 10: Lab. de Programación Orientada a Objetos

Objetivos de Aprendizaje

Al final de esta unidad, usted será capaz de:

- Escribir un script PHP utilizando la Programación Orientada a Objetos.
- Crear una clase que se encargue de realizar las tareas de Bases de Datos.
- Crear clases que sirvan de contenedoras de datos obtenidos de una BD.

Ejercicio de Laboratorio

Se tiene una Base de Datos en MySQL que almacena información sobre los empleados y los departamentos de una compañía. Ud debe elaborar un script PHP que permita:

1) Insertar nuevos empleados en la BD. Los datos de los empleados se tomarán de un formulario HTML. Debe mostrar un mensaje de éxito en caso de que se pueda realizar la inserción y un mensaje de fallo en caso contrario.
2) Eliminar un empleado específico. Se tomará como entrada la ID del empleado que se desea eliminar. Debe mostrar un mensaje de éxito en caso de que se pueda realizar la eliminación y un mensaje de fallo en caso contrario.
3) Actualizar los datos de un empleado. El dato que se actualizará es el Telf. Se tomará como entrada el ID del empleado que se quiere actualizar.
4) Mostrar la lista de empleados existentes. Los detalles de los empleados se deben mostrar en una tabla. Por cada empleado se debe mostrar: ID, Nombre, Telf., Cargo y Departamento.
5) Mostrar la lista de departamentos existentes. Los detalles de los departamentos se deben mostrar en una tabla. Por cada departamento se debe mostrar: ID, Nombre, Teléfono y Área.
6) Debe realizar este ejercicio con la metodología orientada a objetos.
7) Debe definir por lo menos las clases Empleado, Departamento y Controlad. Esta última se encarga de las conexiones a BD y de la ejecución de sentencias SQL.
8) Script para crear las tablas:

```
CREATE TABLE EMPLEADOS (
          IDEMP INTEGER NOT NULL PRIMARY KEY,
          NOMBRE VARCHAR(30),
          FECHA_NAC VARCHAR(20),
          SEXO VARCHAR(1),
          TLF VARCHAR(20),
          CEL VARCHAR(20),
          DIRECCION VARCHAR(100),
          CARGO VARCHAR(30)
          IDDEP INTEGER,
          FOREIGN KEY (IDDEP) REFERENCES
          DEPARTAMENTO(IDDEP))
);
CREATE TABLE DEPARTAMENTO (
          IDDEP INTEGER NOT NULL PRIMARY KEY,
          NOMBREDEP VARCHAR(30),
```

```
            TLF VARCHAR(20),
            AREA VARCHAR(30)
);
```

Unidad 11: Extensiones PHP

Objetivos de Aprendizaje

Al final de esta unidad, usted será capaz de:

- Conocer en qué consiste XML.
- Utilizar la extensión DOM de PHP para trabajar con documentos XML.
- Crear scripts PHP que puedan manipular documentos XML.
- Conocer en qué consiste la librería estándar de PHP.

1. Introducción

En la *Unidad 3 – Funciones y Extensiones PHP*, introdujo brevemente el tema de las extensiones PHP. Las extensiones son módulos externos que permiten incorporar mayor funcionalidad a los programas desarrollados. Una extensión puede ofrecer la funcionalidad de interactuar con una base de datos específica, convertir los elementos de XML en objetos y usarlos dentro del código, utilizar los objetos de Java, etc.

En la *Unidad 7 – Acceso a Bases de Datos usando PHP*, se utilizó la extensión de PHP para trabajar con Base de Datos MySQL. En esta unidad se emplearán algunas funciones de la extensión DOM para trabajar con documentos XML.

Antes de comenzar a trabajar con estas funciones observe una breve introducción acerca del lenguaje XML.

2. ¿Qué es XML?

XML (eXtensible Markup Language) o "Lenguaje de Marcado Extensible" es un lenguaje basado en etiquetas desarrollado por el World Wide Web Consortium (W3C). Es una simplificación y adaptación del SGML (Standard Generalized Markup Language – Lenguaje de Marcado Generalizado Estándar), que permite definir la gramática de lenguajes específicos. Por lo tanto, XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades.

XML se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo, documentos y otros.

XML es una tecnología sencilla que se complementa y combina con otras, lo que aumenta sus características y ventajas. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

2.1 Estructura de un Documento XML

Un documento XML es similar a los documentos HTML, pero un documento XML contiene datos auto definidos, mientras que un documento HTML puede contener datos mezclados con elementos de formato. En XML se separa el contenido de la presentación de forma total.

Al igual que HTML, un documento XML es un archivo de texto plano en los que se utilizan etiquetas para delimitar los elementos del documento. Sin embargo, XML define estas etiquetas en función del tipo de datos que está describiendo y no de la apariencia que tendrán en la página. Además que permite crear nuevas etiquetas y ampliar las existentes.

2.2 Crear un Archivo XML

Un documento XML es un archivo escrito generalmente en texto ASCII plano. Por lo tanto, se puede crear con cualquier editor de texto. EL archivo debe tener la extensión `.xml`.

Todas las páginas HTML necesitan empezar con la etiqueta `<html>`. Así, los documentos XML deben empezar con la siguiente declaración:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

Esta línea especifica la versión de XML a la que corresponde el documento. La codificación (encoding) especifica el conjunto de caracteres que están permitidos en el documento XML.

Los documentos XML pueden tener etiquetas, que están representadas de la misma forma que las etiquetas HTML. Por ejemplo, una etiqueta llamada nombre se puede crear de la siguiente manera:

```
<nombre>Juan</nombre>
```

Los nombres de etiquetas son sensibles a las mayúsculas y minúsculas. Por lo tanto, las etiquetas de apertura y cierre deben ser exactamente las mismas. Escribir un documento XML es bastante fácil. Las reglas de sintaxis son simples, pero deben seguirse estrictamente.

En el siguiente ejemplo se van a representar los detalles de las calificaciones de un sólo estudiante como un documento XML. El documento XML se guarda en el archivo `calificaciones.xml`.

Ejemplo 9.1

El código XML empieza aquí...

```
1.  <?xml version="1.0" encoding="ISO-8859-1"?>
2.  <ResumenNotas>
3.    <Materia>Matemáticas</Materia>
4.    <Calificacion>A</Calificacion>
5.    <Materia>Física</Materia>
6.    <Calificacion>B</Calificacion>
7.    <Materia>Química</Materia>
8.    <Calificacion>B</Calificacion>
9.  </ResumenNotas>
```

El código XML termina aquí

Los datos de las materias y las calificaciones que se han utilizado en el Ejemplo 9.1, se representan con etiquetas que permiten que los datos sean estructurados y significativos. Note también que las etiquetas son todas definidas por el usuario y no etiquetas personalizadas de HTML.

Fin del Ejemplo 9.1

Se han presentado las bases para trabajar con un documento XML. Una discusión más detallada acerca de este tópico está fuera del alcance de este curso.

3. Extensión de PHP para Trabajar con XML

La extensión para soporte a XML ha sido re-estructurada a partir de PHP 4.3.0. Antes se necesitaba incorporar la extensión `'domxml'` para poder manipular los documentos XML. Pero ahora se pueden utilizar las funciones DOM (Document Object Model), que ya vienen incorporadas en PHP. No se necesita ninguna instalación para usar estas funciones, son parte del núcleo de PHP. Se emplearán las funciones de la extensión DOM para manipular documentos XML.

La clase `DOMDocument` implementa las funciones de la extensión DOM para trabajar con archivos HTML o XML.

Nota: Cuando se menciona el archivo `NotasMaterias.xml`, se está haciendo referencia al siguiente documento XML:

```
1.  <?xml version="1.0" encoding="ISO-8859-1"?>
2.  <NotasMaterias>
3.   <NotasEstudiante>
4.   <Estudiante id="100">Juan Medina</Estudiante>
5.   <DetalleMateria materia="Matematicas">A</DetalleMateria>
6.   <DetalleMateria materia="Fisica">A</DetalleMateria>
7.   <DetalleMateria materia="Quimica">C</DetalleMateria>
8.   <DetalleMateria materia="Computacion">B</DetalleMateria>
9.   </NotasEstudiante>
10.  <NotasEstudiante>
11.  <Estudiante id="200">Mariana Mendoza</Estudiante>
12.  <DetalleMateria materia="Matematicas">B</DetalleMateria>
13.  <DetalleMateria materia="Fisica">A</DetalleMateria>
14.  <DetalleMateria materia="Quimica">A</DetalleMateria>
15.  <DetalleMateria materia="Literatura">C</DetalleMateria>
16.  </NotasEstudiante>
17. </NotasMaterias>
```

3.1 Cargar un documento XML

En PHP, se puede cargar un archivo XML en un objeto para luego ser manipulado en el programa. Para cargar un archivo XML se utiliza la función `load()` o `loadxml()` de la clase `DOMDocument`.

- load("nombre_archivo.xml")

Esta función carga un documento XML desde un archivo y lo deja en un objeto del tipo `DOMDocument`. Retorna `true` si se pudo cargar el documento y `false` en caso contrario. Observe el siguiente segmento de código:

```
1.  <?php
2.  /* Se crea un objeto del tipo DOMDocument, que es el
    contenedor del documento xml */
3.  $docXml = new DOMDocument();
4.
5.  /* Se carga el documento XML en el objeto $docXml */
6.  $docXml->load('NotasMaterias.xml ');
7.
8.  /* La funcion saveXML() captura el documento XML cargado en
    el objeto $docXml y retorna el contenido en un string */
9.  echo $docXml->saveXML(); // Se imprime el contenido
10. ?>
11. <?php
12. /* El método load()también se puede invocar como un método
    estático*/
13. $docXml = DOMDocument::load('NotasMaterias.xml');
14. echo $docXml->saveXML();// Se imprime el contenido
15. ?>
```

- **loadXML("codigoXml")**

Esta función carga un documento XML a partir de un string y lo deja en un objeto del tipo `DOMDocument`. Retorna `true` si se pudo cargar el documento y `false` en caso contrario. Observe el siguiente segmento de código.

```
1.  <?php
2.  /* Se crea un objeto del tipo DOMDocument, que es el
    contenedor del documento xml */
3.
4.  $docXml = new DOMDocument();
5.  $strXml="<Materias><Materia>Quimica<Materia></Materias>";
6.  /* Se carga el documento XML en el objeto $docXml */
7.  $docXml->loadXML($strXml);
8.
9.  echo $docXml->saveXML(); // Se imprime el contenido
10. ?>
11. <?php
12. $strXml="<Materias><Materia>Quimica</Materia></Materias>";
13. /* El método loadXML()también se puede invocar como un
    método estático*/
14. $docXml = DOMDocument::loadXML($strXml);
```

```
15.  echo $docXml->saveXML();// Se imprime el contenido
16.  ?>
```

3.2 Capturar un Documento XML

Cuando se ha cargado un documento XML en un objeto del tipo `DOMDocument`, se requiere capturar su contenido ya sea para mostrarlo en pantalla o para almacenarlo en un archivo. Esto se puede realizar con las funciones `saveXML()` y `save()`.

- saveXML()

Captura el documento XML que esté contenido en un objeto `DOMDocument` y lo retorna en un string. Esta función generalmente se llama después de cargar un documento XML. Observe un ejemplo donde se muestre el uso de `saveXML()`.

Ejemplo 9.2

El código PHP comienza aquí...

```
1.  <?php
2.  /* Se crea un objeto del tipo DOMDocument, que es el
    contenedor del documento xml */
3.  $docXml = new DOMDocument();
4.
5.  /* Se carga el documento XML en el objeto $docXml */
6.  $docXml->load('NotasMaterias.xml');
7.
8.  $codXML = $docXml->saveXML(); // Captura el documento XML
9.  echo  $codXML; // Se imprime el contenido
10. ?>
```

El código PHP termina aquí

En la Figura 9.1 se muestra el resultado de ejecutar el código anterior.

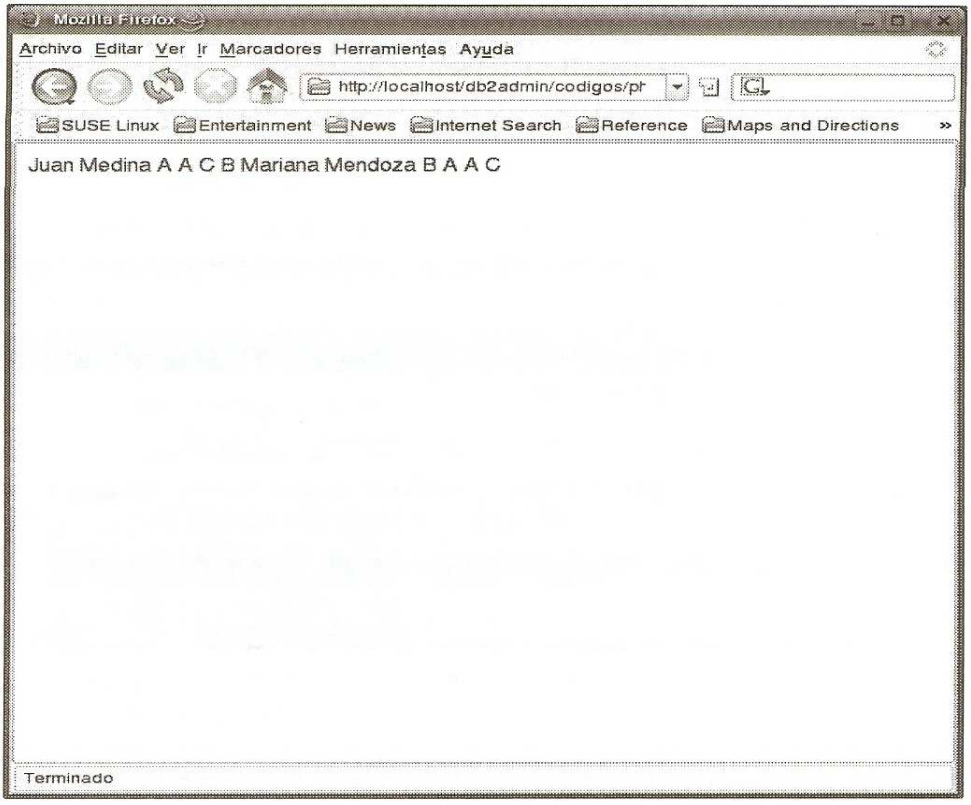


Figura 9.1: Resultado de la ejecución de ejemplo-9.2.php

La función `load()` carga un documento XML desde un archivo y lo deja en el objeto `$docXML`. Luego con el método `saveXML()` se obtiene lo almacenado en `$docXML` y se genera un cadena de texto que se imprime en la página.

Nota: En la Figura 9.1 se observa que el archivo xml cargado desde un .php, no muestra la estructura de árbol convencional, debido a que Firefox no es el navegador Web por defecto, sin embargo, los datos mostrados corresponden a NotasMaterias.xml.

Fin del Ejemplo 9.2

- save($archivo)

Captura el documento XML que esté contenido en un objeto `DOMDocument` y lo almacena en un archivo. Esta función generalmente se llama después de crear un documento XML. Observe un ejemplo donde se muestre el uso de `save()`.

Ejemplo 9.3

El código PHP comienza aquí...

```
1.    <?php
```

```
2.    $strXml="<Materias><Materia>Quimica</Materia></Materias>";
3.    $docXml = DOMDocument::loadXML($strXml);
4.    // Se guarda el contenido en un archivo
5.    $docXml->save("materias.xml");
6.    ?>
```

El código PHP termina aquí

En la Figura 9.2 se muestra el archivo materias.xml, creado al almacenar el documento XML con la funcion `save()`.

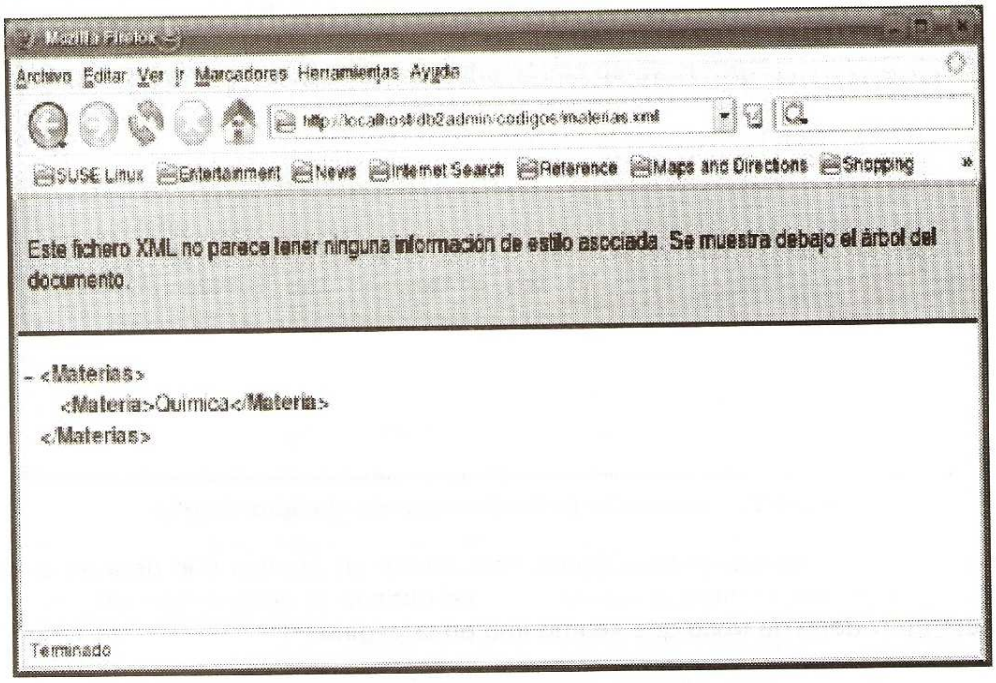


Figura 9.2: Archivo materias.xml, creado con la función `save()`

Fin del Ejemplo 9.3

3.3 Crear Elementos XML

Un documento XML se compone de elementos. Esos elementos, a su vez, pueden tener atributos y contenido. Las siguientes funciones permiten crear y agregar diferentes elementos en un archivo XML. Los elementos en la jerarquía DOM XML se denominan nodos.

- createElement($nombreElemento,$valor)

Crea una nueva instancia de la clase `DOMElement`. Este elemento no formará parte del documento a menos que sea agregado con la función `DOMNode->appendChild()`. El

parámetro `$nombreElemento` es el nombre del elemento que se va a crear y `$valor` el contenido de ese elemento. Si no se especifica $valor, se crea un elemento vacío.

Esta función retorna una instancia de la clase `DOMElement` o `false` en caso de error.

- createAttribute($nombreAtributo)

Crea una nueva instancia de la clase `DOMAttr`. Este atributo no será parte del documento a menos que sea agregado con la función `DOMNode->appendChild()`. Recibe como parámetro el nombre del atributo.

Esta función retorna una instancia de la clase `DOMAttr` o `false` en caso de error.

- createTextNode($contenido)

Crea una nueva instancia de la clase `DOMText`. Este texto no será parte del documento a menos que sea agregado con la función `DOMNode->appendChild()`. Recibe como parámetros el contenido de texto para el elemento.

Esta función retorna una instancia de la clase `DOMText` o `false` si ocurre un error.

- setAttribute($nombreAtr,$valor)

Asigna un valor a un atributo. Si el atributo no existe se crea. Recibe como parámetros el nombre del atributo y su valor.

Esta función retorna `true` si la operación se realizó correctamente o `false` en caso contrario.

- appendChild($nodo)

Agrega un nodo a una lista de nodos. Estos nodos pueden ser atributos, elementos o nodos de texto. El nodo puede ser creado utilizando las funciones `DOMDocument->createElement()`, `DOMDocument->createTextNode()` o `DOMDocument->createAttribute()`.

Estas funciones permiten agregar información nueva a un documento XML. Para comprender estos conceptos a continuación se presenta un ejemplo.

Ejemplo 9.4

En este ejemplo se agrega nuevo contenido al documento XML almacenado en `NotasMaterias.xml`.

El código PHP comienza aquí...

```
1.  <?php
2.  $docXml = new DOMDocument();
3.  $docXml->load('NotasMaterias.xml');
4.  /* Se carga el documento XML en el objeto $docXml*/
```

```
5.
6.    $notasEst = $docXml->createElement("NotasEstudiante");
7.    /* Se crea el elemento <NotasEstudiante> */
8.
9.    $estudiante = $docXml->createElement("Estudiante");
10.   /* Se crea el elemento <Estudiante> */
11.
12.   $nombreEst = $docXml->createTextNode("Eduardo Carrasco");
13.   /* Se crea el texto: "Eduardo Carrasco" */
14.
15.   $estudiante->appendChild($nombreEst);
16.   /* Se agrega el texto al elemento <Estudiante>
17.   Resultado: <Estudiante>Eduardo Carrasco</Estudiante> */
18.
19.   $estudiante->setAttribute("id","300");
20.   /* Se agrega el atributo id="300" al elemento <Estudiante>
21.   Resultado:<Estudiante id="300">Eduardo Carrasco</Estudiante>
      */
22.
23.   $notasEst->appendChild($estudiante);
24.   /* Se agrega el elemento <Estudiante> a <NotasEstudiante>
25.   Resultado: <NotasEstudiante>
26.   <Estudiante id="300">Eduardo Carrasco</Estudiante>
27.   </NotasEstudiante> */
28.
29.   // Ahora se crearán los detalles para una materia
30.
31.   $detalleMat = $docXml->createElement("DetalleMateria");
32.   /* Se crea el elemento <DetalleMateria> */
33.
34.   $nota = $docXml->createTextNode("A");
35.   /* Se crea el texto: "A" */
36.
37.   $detalleMat->appendChild($nota);
38.   /* Se agrega el texto al elemento <DetalleMateria>
39.   Resultado: <DetalleMateria>A</DetalleMateria> */
40.
41.   $detalleMat->setAttribute("materia","Matematicas");
42.   /* Se agrega el atributo materia="Matematicas" al elemento
      <DetalleMateria>
```

```
43.  <DetalleMateria materia="Matematicas">A</DetalleMateria> */
44.
45.  $notasEst->appendChild($detalleMat);
46.  /* Se agrega el elemento <DetalleMateria> a
     <NotasEstudiante>
47.  Resultado: <NotasEstudiante>
48.         <Estudiante id="300">Eduardo Carrasco</Estudiante>
49.         <DetalleMateria
     materia="Matematicas">A</DetalleMateria>
50.                 </NotasEstudiante> */
51.
52.  $docXml->documentElement->appendChild($notasEst);
53.  /* Se agrega <NotasEstudiante> a <NotasMaterias>*/
54.
55.  $codXML = $docXml->saveXML();
56.  echo $codXML;
57.  $docXml->save("NotasMaterias.xml");
58.  ?>
```

El código PHP termina aquí

En la Figura 9.3 se muestra la ejecución del script ejemplo-9.4.php el cual genera como resultado el documento XML modificado (luego de agregar los nuevos elementos).

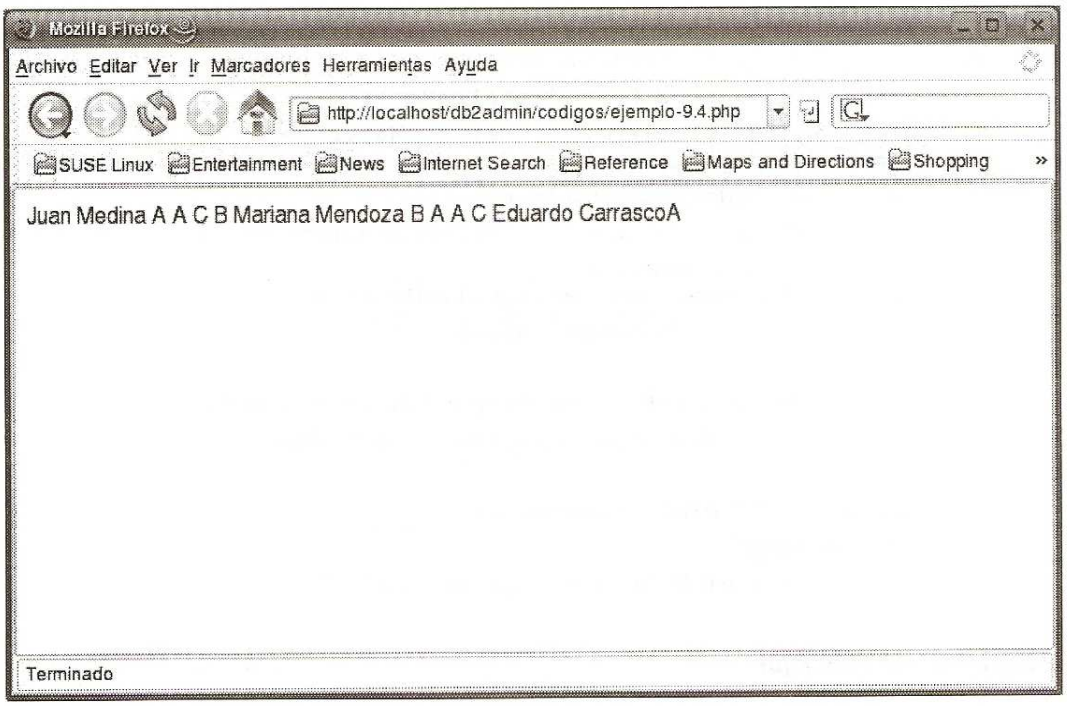


Figura 9.3: Ejecución del script .php que modifica el xml

Fin del Ejemplo 9.4

El archivo `NotasMaterias.xml` es modificado desde ejemplo-9.4.php. A continuación se visualiza, el archivo xml con la información agregada.

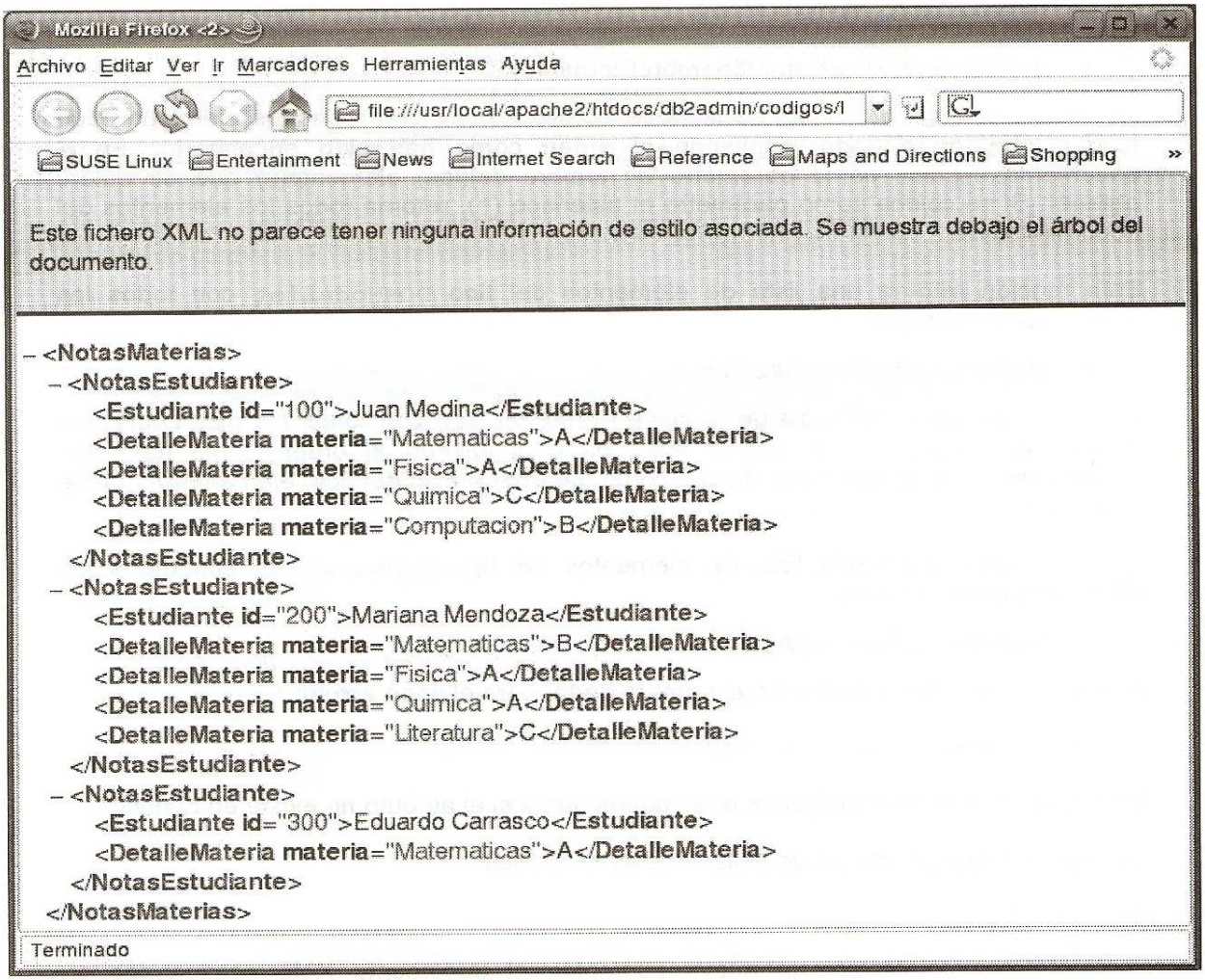


Figura 9.4: Documento xml con la información agregada

A continuación se estudiará cómo seleccionar los elementos XML.

3.4 Obtener Elementos XML

Como se ha podido apreciar en los ejemplos anteriores, un documento XML posee una estructura jerárquica, donde se comienza en un único nodo o elemento raíz (en el ejemplo anterior este elemento es `<NotasMaterias>`), el cual puede poseer varios elementos o nodos hijos, y estos a su vez pueden tener otros nodos hijos y así sucesivamente. Puede ser de interés seleccionar sólo algunos de los nodos que tiene el documento. Esta tarea se puede realizar con las siguientes funciones:

- getElementByTagName($nombreElemento)

Retorna una nueva instancia de la clase `DOMNodeList` con todos los elementos que tengan el nombre dado, siguiendo el orden como han sido encontrados en el documento. Recibe como parámetro el nombre del(los) elemento(s) que se quiere obtener. Si se coloca como parámetro el asterisco (*), retorna todos los elementos del documento.

Esta función retorna una lista de elementos del tipo `DOMNodeList` con todos los elementos seleccionados.

- getElementById($IdElemento)

Retorna una nueva instancia de la clase `DOMNodeList` con todos los elementos que posean un atributo "ID" con el valor dado, siguiendo el orden como han sido encontrados en el documento. Recibe como parámetro el id del (los) elemento(s) que se quiere obtener.

Esta función retorna una lista de elementos del tipo `DOMNodeList` con todos los elementos seleccionados.

- getAttribute($nombreAtributo)

Retorna el valor del atributo con el nombre dado, para el nodo actual:

```
$valor = $nodo->getAttribute("id")
```

Esta función retorna el atributo o una cadena vacía si el atributo no existe en el nodo.

Observe un ejemplo donde se utilizan estas funciones.

Ejemplo 9.5

En este ejemplo se agrega nuevo contenido al documento XML almacenado en `NotasMaterias.xml`.

El código PHP comienza aquí...

```
1.  <?php
2.  $docXml = new DOMDocument();
3.  $docXml->load('NotasMaterias.xml');
4.
5.  $estudiantes = $docXml-> getElementsByTagName("Estudiante");
6.  foreach($estudiantes as $estudiante) {
7.      echo "Nombre del estudiante: ",
8.      $estudiante->textContent,"<br>";
9.      echo "ID del estudiante: ",
```

```
10.      $estudiante->getAttribute("id"),"<br>";
11.      echo "***<br>";
12.  }
13.   ?>
```

El código PHP termina aquí

En el ejemplo se utiliza la propiedad `"textContent"`, del nodo `$estudiante`. Esta propiedad contiene el texto del elemento.

El resultado de la ejecución del script `ejemplo-9.5.php`, se muestra en la Figura 9.5.

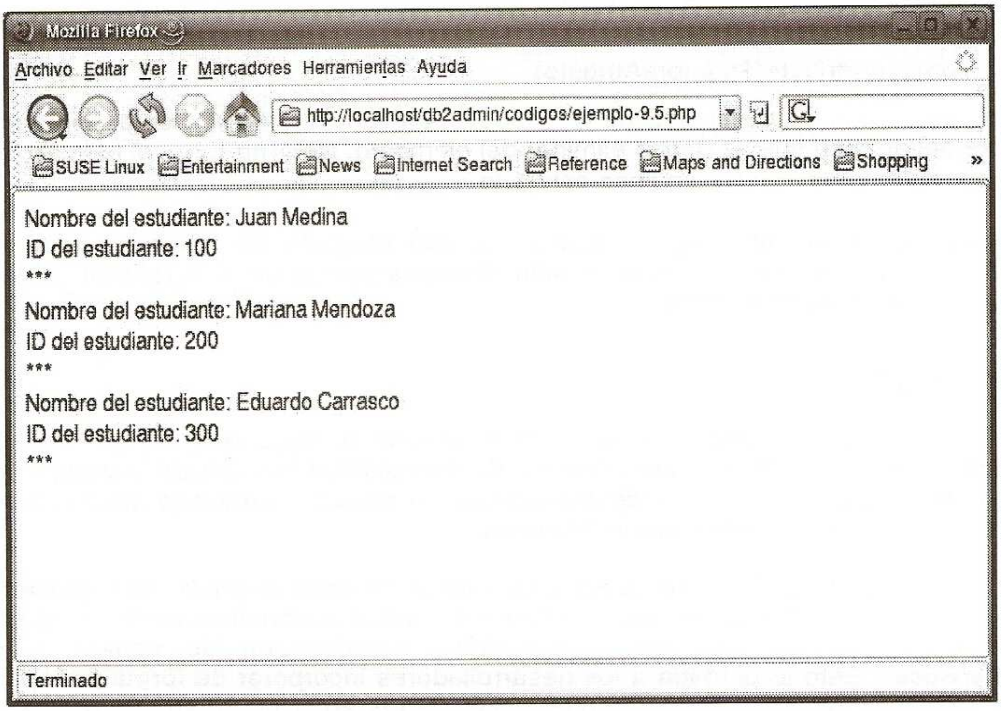


Figura 9.5: Resultado de la Ejecución de ejemplo-9.5.php

3.5 Eliminar Elementos XML

- removeChild($nodo)

Elimina un nodo de la lista de nodos hijos de un elemento especificado. Recibe como parámetro el nodo que se quiere eliminar. Si se puede eliminar, esta función retorna el nodo eliminado. Observe este segmento de código:

```
1.   <?php
2.   $docXml = new DOMDocument;
```

```
3.    $docXml->load('NotasMaterias.xml');
4.
5.    // documentElement se refiere al nodo raíz del documento
6.    $notasMaterias = $docXml->documentElement;
7.
8.    // Se obtiene el primer elemento "NotasEstudiante"
9.    $notasEst = $notasMaterias->
      getElementsByTagName('NotasEstudiante')->item(0);
10.   $notasElim = $notasEst->removeChild($notasEst);
11.
12.   echo $doc->saveXML();
13.   ?>
```

- removeAttribute($nombreAtributo)

Elimina el atributo especificado de un nodo. Esta función recibe como parámetro el nombre del atributo que se desea eliminar del elemento. Retorna `true` si todo se llevó a cabo correctamente, `false` en caso contrario.

La extensión DOM para PHP, posee un amplio conjunto de funciones que no se mencionan en esta unidad. Una discusión detallada acerca de la extensión DOM está fuera del alcance de este curso.

4. PHPLib

PHP comenzó a ser popular a partir de la versión 3. Pero esa versión no estaba diseñada para el mantenimiento eficiente de aplicaciones complejas. Además tareas como el manejo de sesiones, procesamiento de variables de formulario, acceso a bases de datos, presentaban ciertas complicaciones.

Para esta versión de PHP se liberó una librería llamada PHPLib, que contiene un conjunto de clases y funciones que permiten ciertas tareas comunes como el registro de sesiones por usuario, variables persistentes, autenticación de usuario y otras características. Esto le permitía a los desarrolladores incorporar de forma transparente muchas características, sólo tenían que incluir ciertos archivos en sus programas. Por ejemplo, si querían utilizar la librería para crear un carrito de compras sólo tenían que incluir en sus scripts esta línea: "`include('cart.inc')`", luego podían utilizar las funciones provistas.

La última versión liberada de esta librería es la versión 7.2. Muchas de las funciones que ofrece PHPLib fueron mejoradas e incorporadas de forma nativa en las versiones posteriores de PHP.

Actualmente, PHP 5 incorpora de forma nativa una librería estándar, conocida como SPL (Standard PHP Library). La SPL es una colección de interfaces y clases que están hechas para resolver problemas estándar e implementar interfaces y clases eficientes

para acceso de datos. Esta librería ya se encuentra incorporada de forma nativa en PHP 5, es decir, no son necesarios módulos externos para poder utilizar sus características.

La SPL incorpora características que permiten trabajar con arreglos, archivos, programación orientada a objetos, manejo de excepciones y mucho más, de una forma sencilla. Las funciones de arreglos y archivos que se explicaron en la *Unidad 3 - Funciones y Extensiones PHP*, pertenecen a la SPL. Adicionalmente, existen algunas funciones relacionadas con la programación orientada a objetos y manejo de excepciones que se encuentran en esta librería. El desarrollador las utiliza de forma transparente.

Resumen

Ahora que ha completado esta unidad, usted debe ser capaz de:

- Conocer en qué consiste XML.
- Utilizar la extensión DOM de PHP para trabajar con documentos XML.
- Crear scripts PHP que puedan manipular documentos XML.
- Conocer en qué consiste la librería estándar de PHP.

Unidad 11: Examen de Autoevaluación

1) ¿Cuáles de las siguientes son características de XML?
   a) Es un lenguaje de marcado basado en etiquetas
   b) XML mezcla la estructura de los datos con el formato de presentación
   c) Permite definir etiquetas personalizadas
   d) XML se propone como un estándar para el intercambio de información estructurada

2) La extensión DOM de PHP debe habilitarse explícitamente para poder utilizar sus funciones.
   a) Verdadero
   b) Falso

3) ¿Cuáles de las siguientes funciones se utiliza para cargar un documento XML?
   a) save()
   b) saveXML()
   c) load()
   d) loadXML()

4) ¿Cuáles de las siguientes funciones se utiliza para capturar un documento XML y almacenarlo en una variable o en un archivo según sea el caso?
   a) save()
   b) saveXML()
   c) load()
   d) loadXML()

5) La función `createElement()` crea un elemento, es decir, una nueva instancia de la clase `DOMElement`.
   a) Verdadero
   b) Falso

6) Cuando se crea un elemento utilizando la función `createElement()`, automáticamente el elemento creado se agrega al documento XML actual.
   a) Verdadero
   b) Falso

7) ¿Qué hace la función appendChild()?
   a) Crea un nuevo atributo
   b) Carga un documento XML
   c) Agrega un nodo a una lista de nodos del documento XML
   d) Elimina un elemento del documento XML

8) ¿Con cuáles de las siguientes funciones se puede seleccionar un elemento o un conjunto de elementos de un documento XML?
   a) getElementByTagName()
   b) getElementBySize()
   c) getElementbyCount()
   d) getElementById()

9) PHP 5 incorpora de forma nativa una librería estándar llamada SPL.
   a) Verdadero
   b) Falso

10) La SPL (Standard PHP Library) incorpora características que permiten trabajar con arreglos, archivos, programación orientada a objetos y manejo de excepciones.
   a) Verdadero
   b) Falso

Unidad 11: Respuestas a Examen de Autoevaluación

1) a, c y d
2) b
3) c y d
4) a y b
5) a
6) b
7) c
8) a y d
9) a
10) a