Sé diferente, intégrate...

Mca022

Manual
Programando Aplicaciones Web con PHP

Autor: Orlando Gutiérrez
Fecha: 01/03/2013

INDICE

LECCION 1 INTRODUCCIÓN A PHP: CONCEPTOS, ALCANCE, CARACTERISTICAS

PHP es el acrónimo o "stand" de "**Hypertext Preprocessor**". PHP es un lenguaje "**open source**"; permitiendo la posibilidad de obtener el código fuente del motor de ejecución de PHP escrito en Lenguaje C para ser revisado o modificado. PHP es un lenguaje **interpretado**, todas las instrucciones de PHP son ejecutadas una por una en lugar de ser compiladas en un programa ejecutable. Las líneas de código escrito en PHP se encuentran embebidas **"embebed"** en páginas HTML; por lo tanto el código PHP se especifica entre "tags" HTML. PHP se ejecuta del **lado del servidor**, en lugar de ejecutarse del lado del cliente como es el caso de los javascripts.

Un ejemplo introductorio

```
<html>
  <head>
     <title>Ejemplo</title>
  </head>
  <body>

     <?php
     echo " Bienvenidos: Primer script PHP";
     ?>

  </body>
</html>
```

El estilo de programación de PHP consiste en colocar instrucciones de PHP identificadas por tags dentro de una página HTML. Cuando un cliente requiere de un servidor de páginas WEB una página PHP, el motor de ejecución interpreta el código de PHP y convierte el resultado de la evaluación de las expresiones en texto HTML. El cliente finalmente recibe en su navegador un archivo HMTL con el resultado de la operación. Este esquema de operación definido permite lograr la seguridad del código, ocultando a los clientes las porciones de código escritas en PHP.
La tecnología de PHP es totalmente orientada al servidor distinguiéndose de otras tecnologías como la de Javascript, la cual se ejecuta del lado del cliente. Un servidor WEB puede ser configurado para procesar todos los archivos HTML con PHP.
El rango de aplicabilidad de PHP es bastante amplio. Programadores principiantes pueden realizar páginas PHP, pero por otro lado ofrece una librería de funciones y unos constructores de programación para ser utilizados por programadores profesionales.
Aunque la mayoría de los desarrollos en PHP se basan en la programación de "scripts" del lado del servidor; PHP se puede utilizar de varias otras formas como por ejemplo para la ejecución de programas desde la línea de comandos o aplicaciones "desktop".

ALCANCE DE PHP

PHP implementa la "interfaz" de comunicación **CGI "Common Gateway Interface"**; por lo tanto cualquier funcionalidad implementada en un "script" CGI puede ser desarrollada por PHP. Entre las funciones más comunes de CGI se encuentra el procesamiento de la información de formularios, la generación de contenido HTML dinámico y el envío y la recepción de "cookies".

Los programas en PHP pueden ser agrupados en tres grandes grupos:

- **Aplicaciones WEB**, programadas como Scripts PHP ejecutándose del lado del servidor. Este es el tipo de aplicaciones más popular dentro de la comunidad de PHP y para lo cual el lenguaje fue diseñado. La premisa de diseño de PHP se basa en la generación de páginas dinámicas. Para desarrollar aplicaciones WEB con PHP se requiere un servidor de páginas WEB, un intérprete de PHP instalado y configurado en la máquina donde se ejecuta el servidor de las páginas web y un cliente (el cliente puede ser cualquier navegador de html) . El resultado de una aplicación WEB se obtiene del lado del cliente, siendo el servidor el encargado de interpretar el código PHP generando una página HTML dinámica como resultado.

- Aplicaciones en **línea de comandos (estilo consola)** como Scripts ejecutándose en una máquina "StandAlone". Para este tipo de aplicaciones no se requiere un servidor de páginas web, únicamente se requiere el interpretador de PHP instalado en la máquina donde se va a ejecutar el Script. Puede crear un script PHP y correrlo sin ningún servidor web o navegador. Generalmente este tipo de scripts se emplean para

ejecutar tareas del sistema de Operación (como por ejemplo las tareas programadas a través del comando cron en Linux, o el administrador de programas de Windows). En general pueden ser utilizados para la ejecución de programas tipo consola.

- Aplicaciones **gráficas (estilo Windows o Motif)** como Scripts ejecutándose en una máquina "StandAlone". Debe considerarse que PHP no es el lenguaje más adecuado para desarrollar este tipo de aplicaciones. Para trabajar en las aplicaciones gráficas debe utilizarse **PHP-GTK**, el cual es un ambiente integrado de desarrollo IDE. PHP-GTK no forma parte de las instalaciones PHP por defecto (distribución principal). La página web donde puede conseguirse información de PHP-GTK es http://gtk.php.net

PHP se encuentra disponible en la mayoría de los sistemas de operación comerciales (**plataformas de hardware**), incluyendo Linux, muchas versiones de Unix (HP-UX, Solaris y OpenBSD), Microsoft Windows, Mac OS X, RISC OS.

PHP puede ser interpretado por la mayoría de **servidores web** actuales, como Apache, Microsoft Internet Information Server (IIS), Personal Web Server, Netscape e iPlanet, Oreilly Website Pro server, Caudium, Xitami, OmniHTTPd, Mozilla.
La distribución de PHP contiene un grupo de **módulos** disponibles, los cuales pueden ser empleados por la mayoría de los servidores web. PHP también puede ser utilizado como el procesador del estándar **CGI**, en cualquier navegador soportando este estándar.

Una de las ventajas de PHP sobre otros lenguajes de programación de scripts en WEB es que ofrece la posibilidad de trabajar en múltiples plataformas (sistemas de operación) y la independencia sobre el servidor de páginas web.

Desde el punto de vista de programación, PHP soporta **diferentes técnicas de programación** como: programación estructurada (descomposición de una tarea en funciones), programación modular (descomposición de un programa en módulos) y programación orientada por objetos (un programa como una colección de objetos colaborando entre sí a través del pase de mensajes).

PHP ofrece una **librería de funciones** muy extensa y muy poderosa facilitando su reutilización y permitiendo a los programadores concentrarse en el dominio del problema a resolver. Dentro de las funciones básicas incluye funciones matemáticas, funciones para el manejo de archivos, funciones para el manejo de cadenas, funciones para el manejo de bases de datos, funciones para el manejo de comunicaciones en redes, funciones para el manejo de comercio electrónico, etc.

Para el desarrollo de **proyectos de gran escala** se recomienda emplear la tecnología de Programación Orientada por Objetos. Las librerías de funciones de PHP están desarrolladas basándose en clases de objetos.

PHP no está limitado a la producción de resultados como páginas HMTL, también es posible empleando PHP la **creación de imágenes**, la producción de archivos .PDF, la generación dinámica de presentaciones en Flash (empleando libswf y Ming).

En cuanto a los estándares actuales de integración y desarrollo de aplicaciones orientadas hacia los servicios WEB , PHP soporta el procesamiento de archivos **XML** (eXtended Markup Language) a través de los estándares SAX (Simple Api Xml) y DOM (Document Object Module). Se puede utilizar la extensión XSLT para transformar documentos XML.

Desde el punto de vista de integración de aplicaciones y el desarrollo de aplicaciones comerciales la funcionalidad más apreciada de PHP es el soporte ofrecido para realizar la **interconexión con una amplia gama de servidores de Base de Datos**. De esta manera el desarrollo de una interfaz a través de la WEB cono una Base de Datos es una tare muy sencilla en PHP. A continuación se muestra una lista con las bases de datos soportadas por PHP.

Adabas D
DBase
Empress
FilePro (read-only)
Hyperwave
IBM DB2
Informix
Ingres
Internase
FrontBase
MSQL

Direct MS-SQL
MySQL
ODBC
Oracle (OCI7 and OCI8)
Ovrimos
PostgreSQL
Solid
Sybase
Veloces
Unix dbm

PHP, adicionalmente soporta las bases de datos con extensión **DBX** y el estándar abierto de conexión **ODBC** (Open Data Base Connectivity); de esta manera prácticamente PHP puede conectarse con cualquier manejador de base de datos comercial.

En cuanto al soporte de protocolos para comunicarse con **servicios** ofrecidos por los sistemas de operación o los servidores de páginas WEB, PHP soporta los siguientes **protocolos**: LDAP, IMAP, SNMP, NNTP, POP3, HTTP, COM (en Windows). En PHP, se pueden programar **"sockets"** de interconexión, funciones de intercambio de datos entre lenguajes de programación en web con WDDX. Desde el punto de vista de servicios de **"middleware",** PHP soporta los tres estándares más utilizados: **objetos Java** de forma transparente como objetos PHP , la extensión de **CORBA** puede ser utilizada para acceder a objetos remotos y las interfaces **COM** de Microsoft.

PHP ofrece un conjunto de funciones muy útiles para el procesamiento de texto, manejando expresiones regulares POSIX extendidas o tipo Perl.

PHP en el ámbito del comercio electrónico, ofrece funciones como Cybercash, CyberMUT, VeriSign Payflow Pro y CCVS las cuales facilitan el desarrollo de una tienda virtual.

Durante el resto de este curso se cubrirán en detalle algunas de estas funciones.

CARACTERISTICAS DE PHP

a) Lenguaje de Programación:

PHP comenzó como un conjunto de macros para convertirse en un lenguaje de programación bastante maduro. Su mayor utilidad es como un lenguaje de "scripting", embebido en código con páginas HTML, aunque puede se utilizado como un lenguaje de programación en cualquier ambiente.
PHP soporta Programación Estructurada , Programación Modular y Programación Orientada por Objetos. La versión 5.0, ofrece todos los constructores necesarios para la programación orientada por objetos.

b) Alta conectividad

PHP soporta múltiples funciones de integración con otras aplicaciones. En el caso de Base de datos, ofrece soporte natural con mysql . Soporta librerías de funciones para integrarse con SQL Server, Oracle, acceso ODBC (permitiendo prácticamente conectarse con cualquier manejador de base de Datos soportando esta Standard).
PHP soporta manejo de XML, permitiendo el desarrollo de los servicios WEB, el desarrollo de soluciones e-commerce, e-business.
PHP soporta el desarrollo de aplicaciones WEB, debido a su integración directa con HTML.

c) Interpretación de scripts y generación dinámica de páginas HMTL

El motor de PHP interpreta el código de lo "scripts" y lo traduce en texto en HTML. De esta manera se garantiza que el cliente recibe código HTML ya interpretado y únicamente el servidor se encarga de procesar el código fuente. De esta manera se garantiza la seguridad de las aplicaciones y la portabilidad del código. Cualquier plataforma soportando un navegador WEB es capaz de ser un cliente de una aplicación WEB en PHP.

d) "Sofware" libre , abierto y soporte múltiples plataformas

PHP se caracteriza por ser un software libre (sin costo alguno) y abierto (se encuentra disponible el código fuente en lenguaje C). De esta manera es posible manejar el motor de ejecución ZEND de PHP y personalizar PHP. (El motor de PHP).
Otra de las ventanas de PHP , desde el punto de vista de desarrollo, es la disponibilidad en múltiples plataformas y la portabilidad del código debido a ser archivos php, los cuales son archivos html con "tags" embebidos php.

e) Biblioteca (librería) de funciones de PHP

PHP es un lenguaje de programación sumamente versátil. Esta versatilidad se logra a través de la librería de funciones bastante extensa. El conjunto de funciones de PHP soporta una gran variedad de operaciones para desarrollar e integrar aplicaciones.

LECCION 2 INTRODUCCION AL DESARROLLO WEB EN PHP: USO DE FORMULARIOS HTML, SERVIDORES CGI

PHP permite procesar formularios HTML. Es posible asociar con la acción del botón de procesar un formulario un programa escrito en PHP. En el siguiente ejemplo se muestra que es únicamente necesario indicar el nombre del archivo donde se encuentra el código de PHP en el tag action del formulario.

Ejemplo de formulario HTML ejecutando el código de fAlumno.php

```
<html>
<head>
<title> Sistema de manejo de Ordenes </title>
</head>

<body>

<form action="fAlumno.php" method="POST">

 Indique su nombre: <input type="text" name="nombre" />
 Indique su CI : <input type="text" name="CI" />

 <input type = "submit" value = "procesar">

</form>

</body>

</html>
```

Dentro del código de PHP es posible tener acceso a todos los campos del formulario, en este caso el campo “nombre” y el campo “CI”.
Cuando el usuario de la página Web llena este formulario y presiona el botón etiquetado "procesar", el servidor de web invoca a la página llamada fAlumno.php es llamada. En este archivo encontrará algo así:

Archivo fAlumno.php

```
<html>
<head>
<title> Sistema de manejo de Alumnos </title>
</head>
<body>
<?php
print "El nombre es <b>$REQUEST["nombre"]; </b><P>\n\n";
print "La CI es <b>$REQUEST["CI"]</b><P>\n\n";
?>
</body>
</html>
```

El resultado de este script sería por ejemplo:

```
El nombre es Gala
La CI es 00000000
```

El objetivo de este ejemplo es mostrar como obtener los valores de los campos de un HTML en una página con código HTML. PHP posee unas variables globales donde se almacenan todos los campos de las páginas HTML manejadas, estas variables se almacenan en *$_REQUEST["nombre"]* y *$_REQUEST["CI"]* . Las variables almacenadas en REQUEST consideran el caso de las variables POST y las variables GET. Se puede observar en la página HTML que se define como el método de procesamiento el modo POST; por lo tanto en este ejemplo hubiera sido valido colocar también *$_POST["nombre"]* y *$_POST["CI"]*. Si el método del formulario hubiera sido GET las variables se pueden referenciar como *$_GET["nombre"]* y *$_GET["CI"]*.

La configuración de PHP se realiza modificando el archivo php.ini. Si se desea utilizar directamente los nombres de los campos sin colocar *$_REQUEST* , se debería editar la opción de configuración **register_globals = On (por razones de seguridad no es conveniente, ya que por error se podría modificar una variable global si desearlo. Es conveniente que esta opción se muestre como Off).**

Si la opción register_globals se encuentra en On, se podría reemplazar el código de PHP por el siguiente:

Archivo fAlumno.php

```
<html>
<head>
<title> Sistema de manejo de Alumnos </title>
</head>
<body>
<?php
print "El nombre es <b>$nombre; </b><P>\n\n";
print "La CI es <b>$CI</b><P>\n\n";
?>
</body>
</html>
```

El resultado de este script sería el mismo del caso anterior

Otra alternativa es emplear la instrucción **import_request_variables**, en este caso se indican cuales son las variables del formulario a importar y se especifica un **nombre corto** para acceder a las variables. A continuación se muestra un ejemplo de la utilización de esta instrucción.

Archivo fAlumno.php

```
<html>
<head>
<title> Sistema de manejo de Alumnos </title>
</head>
<body>
<?php
import_request_variables("gP","F_");
print "El nombre es <b>$F_nombre; </b><P>\n\n";
print "La CI es <b>$F_CI</b><P>\n\n";
?>
</body>
</html>
```

El resultado de este script sería el mismo del caso anterior

En el ejemplo anterior la instrucción import_request_variables("gP","F_");, le indica a PHP que se van a importar las variables GET y POST y se va a utilizar como nombre corto F_; por lo tanto $REQUEST["nombre"] se reemplaza por $F_nombre y $REQUEST["CI"] se reemplaza por $F_CI.

SERVIDORES CGI

CGI es la abreviatura o acrónimo para Common Gateway Interface. CGI permite crear páginas WEB (HTML) dinámicas basada en la información de botones, “checkbox”, entradas de texto y cualquier elemento de HTML. Estas páginas

pueden contener imagines, sonidos y cualquier pieza de información que pueda ser transmitida en Internet. Inclusive estos elementos pueden ser referencias a otros elementos de otras páginas WEB.

CGI es un estándar para desarrollar interfaces entre aplicaciones y servidores de información como servidores Web o http. Un documento html por defecto maneja contenido estático. En cambio, un CGI es ejecutado dinámicamente ("en tiempo real") produciendo información dinámica.

Por ejemplo, se desea desarrollar un programa en CGI para que muestre en una página WEB los resultados de una consulta de la base de datos dependiendo de los datos introducidos por un usuario de la aplicación. En este caso, el programa CGI captura los datos introducidos por el usuario, se comunica con la Base de Datos, ejecuta la consulta y genera un documento HTML con los resultados producidos el cual es enviado al usuario.

Desde el punto de vista de implementación, un programa CGI es un programa ejecutable. Por lo tanto, cuando se desarrolla un CGI se le está permitiendo a los usuarios de una aplicación WEB ejecutar el programa CGI en la máquina donde se encuentra almacenado el servidor de páginas WEB. Por lo tanto, la programación de CGI debe considerar varios aspectos relacionados con la seguridad. En primer lugar, los programas CGI deben residir en un directorio específico dentro de los servidores de WEB, de modo tal que el servidor conoce como ejecutar el programa para producir los resultados y únicamente enviar al cliente el resultado de la operación. El control de este directorio lo tiene el administrador del sitio Web de modo de controlar el acceso y la instalación de programas CGI. Por lo general los nombres de estos directorios son similares a /cgi-bin.

Un programa CGI puede ser escrito en cualquier lenguaje que pueda ser ejecutado en la máquina donde se encuentra instalado el servidor de páginas WEB. En este caso los programas CGI serán desarrollados en PHP.

Por lo general , los programas de CGI son considerados programas de "scripts". Existen otros lenguajes aparte de PHP soportando el desarrollo de "scripts" CGI, entre estos se encuentran:

- C/C++
- Fortran
- PERL
- TCL
- Cualquier "shell" de Unix
- Visual Basic
- AppleScript

LECCION 3 CONFIGURACIÓN DE PHP

La configuración de PHP determina el comportamiento de ejecución del mismo permitiendo la personalización del ambiente de desarrollo de PHP según las características de un equipo de desarrollo. Adicionalmente, la configuración determina el modo de ejecución de PHP.

La configuración de PHP puede realizarse a través de un archivo con extensión .ini. En PHP este archivo se llama php.ini. El archivo .ini es leído cuando arranca PHP. En el caso de utilizar PHP como un servidor de contenido dinámico en WEB, este archivo se lee cuando arranca el servidor WEB. Por lo tanto, es necesario reiniciar el servicio de servidor de web al realizarse modificaciones en este archivo.
Los archivos php.ini contienen una serie de comandos e instrucciones (parámetros de configuración). Afortunadamente, el archivo php.ini se encuentra bien documentado y se especifica claramente la semántica del parámetro a configurar. Por otro lado, el archivo de configuración es bastante extenso lo cual es ofrece poder de configuración pero requiere manejar muchos detalles. En la página de documentación de php; www.php.net se pueden conseguir todos los parámetros de configuración.

Los archivos php.ini son archivos de texto y pueden ser editados con cualquier editor de texto. Se pueden especificar comentarios en este archivo colocando ";".

A continuación se muestra un ejemplo, de varios parámetros de configuración.

```
engine = On

short_open_tag = On

asp_tags = On

precision    =  12
```



```
y2k_compliance = On

output_buffering = Off

zlib.output_compression = Off

implicit_flush = Off

unserialize_callback_func=

serialize_precision = 100

allow_call_time_pass_reference = On

safe_mode = Off

safe_mode_gid = Off

safe_mode_include_dir =

safe_mode_exec_dir =

safe_mode_allowed_env_vars = PHP_

safe_mode_protected_env_vars = LD_LIBRARY_PATH

;open_basedir =

disable_functions =

disable_classes =

;highlight.string  = #DD0000

;highlight.comment = #FF9900

;highlight.keyword = #007700

;highlight.bg      = #FFFFFF

;highlight.default = #0000BB

;highlight.html    = #000000

expose_php = On

max_execution_time = 30

max_input_time = 60

memory_limit = 8M

error_reporting  = E_ALL;

display_errors = On

display_startup_errors = Off
```

```
log_errors = Off

log_errors_max_len = 1024

ignore_repeated_errors = Off

ignore_repeated_source = Off

report_memleaks = On

track_errors = Off

html_errors = Off
```

Se puede mostrar en el navegador los valores de los parámetros de configuración empleando las instrucciones de PHP phpinfo().

int **phpinfo** (void)

Devuelve la información sobre el estado actual de PHP. Incluye información sobre las opciones de compilación, las extensiones de PHP, la versión PHP, la información de ambiente , la información del servidor, el entorno PHP, la versión del Sistema de Operación, los caminos de los directorios, las opciones de configuración maestras y locales, los "headers", y la Licencia Pública GNU.
El valor de un parámetro de configuración puede ser actualizado empleando la instrucción ini_set().

string **ini_set** (string nombre_var, string nuevo_valor)

Actualiza el valor de un parámetro de configuración. Retorna el valor antiguo en caso de éxito, **FALSE** en caso de falla. La opciÓn de configuración mantendrá este nuevo valor durante la ejecución del script, y **será restablecido** al final del script.

También se puede consultar el valor de un parámetro de configuración empleando la instrucción ini_get().

string **ini_get** (string nombre_var)

Retorna el valor del parámetro de configuración en caso de éxito. En caso de producirse un fallo, por ejemplo al realizar una consulta de un parámetro inexistente, devolverá una cadena de caracteres vacía.

Ejemplo php.ini

```
; Permite documentar
; los archivos php.ini vienen en inglés
; registro variables globales
register_globals = off

; traza de errores

track_errors = yes


; estilo linux

include_path = ".:/usr/local/lib/php"


; estilo windows

include_path = ".;c:\php\lib"
```


LECCION 4 INTRODUCCIÓN AL LENGUAJE DE PROGRAMACIÓN PHP: SINTAXIS BASICA, TIPOS Y ESTRUCTURAS DE DATOS, VARIABLES, CONSTANTES, EXPRESIONES, OPERADORES

PHP es un lenguaje de programación bastante **flexible y poderoso**. La sintaxis de PHP es muy similar a la sintaxis de los lenguajes C, C++ y Java; por lo tanto cualquier programador de cualquiera de estos lenguajes puede comenzar a programar en PHP inmediatamente.

El código escrito en el lenguaje de programación PHP es **interpretado**. Por lo tanto, las instrucciones de un programa en PHP son ejecutadas una por una hasta que se produzca un error o hasta que se finalice la ejecución. Al ser PHP interpretado se facilita su implementación en múltiples plataformas. Los lenguajes de sintaxis similar (C, C++, Java) mencionados anteriormente son todos compilados, en este caso se generan archivos .class o .obj y se compila todo el programa en lugar de interpretarse línea por línea. Java es un caso muy particular porque el código fuente se compila en archivos .class, pero luego la máquina virtual de Java interpreta línea por línea el contenido de un .class.

PHP es un lenguaje **no fuertemente tipado**. En PHP no es necesario asignar un tipo a una variable, no es necesario declarar las variables, una variable a lo largo de la vida de un programa puede almacenar variables de distintos tipos.

PHP soporta las técnicas de **programación estructurada** empleando function() para descomponer un problema en subproblemas; también soporta las técnicas de programación modular a través de la instrucción **incluye** y la programación orientada por objetos con los constructores del lenguaje **class**. En el caso de la programación orientada por objetos, la funcionalidad es implementada totalmente en PHP 5, en la versión sólo se ofrece el constructor de clase y el manejo de herencia (no soporta el polimorfismo ni el encapsulamiento).

PHP ofrece un modelo para **manejar los errores** de la aplicación manejando una traza de los errores, permitiendo a un programa en PHP disparar errores dependientes del dominio de la aplicación, soportando la facilidad de programar un manejador de errores específico para ciertas porciones del programa. En la versión 5 de PHP se ofrece el manejo de excepciones similar al de C++ y Java con las instrucciones del **tipo try .... catch**.

PHP maneja el concepto de **referencias** tanto a variables como a funciones, permitiendo implementar el concepto de variables pasadas como referencia en parámetros a funciones y variables funciones. Las variables funciones son lo que se conoce en C++ como los apuntadores a las funciones. En general referencia se refiere a un apuntador.

La principal ventaja de PHP como lenguaje de programación es la **amplia librería de funciones ofrecidas**. Dentro de esta librería se tienen: operaciones de cadenas de caracteres, operaciones matemáticas, funciones manejo de bases de datos, funciones de manejo de red, funciones de manipulación de imágenes, funciones de seguridad en Internet, funciones de correo electrónico, funciones de comercio electrónico, funciones de manejo de archivos, entre otras.

A continuación se muestra una tabla con el **grupo de funciones soportadas por PHP**.

Funciones específicas de Apache
Funciones avanzadas de depuración de PHP
Funciones de matrices
Funciones matemáticas de precisión arbitraria BCMath
Funciones del compilador de bytecode de PHP
Funciones de compresión Bzip2
Funciones de calendario
Funciones del API de CCVS
Funciones de Clases/Objetos
Funciones COM y .Net (Windows)
Funciones ClibPDF
Funciones de “Crack”

Funciones de Tipo de Caracter

Funciones CURL (Librería de cliente URL)

Funciones de pago electrónico

Funciones de Administración CYRUS IMAP

Funciones de Fecha y Hora

Funciones de la capa de abstraccion de bases de datos (tipo dbm)

Funciones para dBase

Funciones DB++

Funciones dbx

Funciones de acceso directo a E/S

Funciones de Directorio

Funciones DOM

Funciones DOM XML

Funciones .NET

Funciones de Gestión de Errores y Registros

Funciones de Ejecución de Programas

Funciones Exif

Funciones para el Monitor de Archivos

Funciones FrontBase

Funciones del Formato de Datos de Formulario

Funciones filePro

Funciones del Sistema de Archivos

Funciones FriBiDi

Funciones FTP

Funciones de Gestión de Funciones

Funciones Gettext

Funciones GMP

Funciones HTTP

Funciones para Hyperwave

Funciones InterBase

Funciones iconv

Funciones ID3

Funciones de Informix

Funciones de Administración IIS

Funciones para imágenes

Funciones IMAP, POP3 y NNTP

Opciones e Información de PHP

Funciones Ingres II

Funciones IRC Gateway

Integración de Java y PHP

Funciones LDAP

Funciones libxml

Funciones LZF

Funciones de Correo

Funciones mailparse

Funciones matemáticas

Funciones manejo String Multibyte

Funciones MCAL

Funciones de Cifrado Mcrypt

Funciones pago MCVE

Funciones Memcache

Funciones Mhash

Funciones Mimetype

Funciones Ming para Flash

Funciones para mnoGoSearch

Funciones mSQL

Funciones de Microsoft SQL Server

Functions muscat

Funciones MySQL

Extensión mejorada de MySQL

Funciones de Control de Pantalla con Terminal Ncurses

Funciones de Red

Funciones NIS

Funciones Lotus Notes

Funciones NSAPI-specific

Funciones de Agregración y Composición

Funciones de Oracle 8

Funciones OpenSSL

Funciones Oracle

Funciones de Control de Salida

Propiedades de los objetos y llamadas a métodos sobrecargados

Funciones Ovrimos SQL

Funciones Parsekit

Funciones de Control de Procesos

Funciones de Expresiones Regulares (Compatibles con Perl)

Funciones PDF

Funciones PDO

Funciones Verisign Payflow Pro

Funciones PostgreSQL

Funciones POSIX

Funciones de impresión

Funciones Pspell

Funciones qtdom

Funciones Rar

Funciones GNU

Funciones de Expresiones Regulares (POSIX Extendido)

Funciones Semáforo y de memoria compartida

Funciones base de datos SESAM

Funciones para el manejo de sesiones

Funciones de Memoria Compartida

Funciones XML simples

Funciones SNMP

Funciones SOAP

Funciones de Socket

Funciones de librería estandar PHP

Funciones SQLite

Funciones Secure Shell2

Funciones de Secuencias

Funciones de Cadenas

Funciones Shockwave Flash

Funciones de Sybase

Funciones TCP Wrappers

Funciones Tidy

Funciones Tokenizer

Funciones ODBC

Funciones de URL

Funciones de Variables

Funciones vpopmail

Funciones W32api

Funciones WDDX

Funciones xattr

Functions xdiff

Funciones de intérprete XML

Funciones XML-RPC

Funciones XSL

Funciones XSLT

YAZ Funciones de manejo de archivos Zip (sólo lectura) Funciones de Compresión Zlib

A continuación se cubrirán los constructores del lenguaje PHP.

- Tipos
- Variables
- Constantes
- Expresiones
- Operadores
- Estructuras de Control
- Funciones
- Clases y Objetos
- Excepciones
- Referencias

TIPOS Y ESTRUCTURAS DE DATOS

Tipos de datos

PHP soporta ocho tipos primitivos o tipos bases.

Cuatro tipos simples:

- **boolean**: Representa un valor de tipo lógico. Puede ser **TRUE** o **FALSE**. Para el caso de las variables del tipo bolean, el intérprete de PHP no es sensitivo al caso de mayúsculas y minúsculas.

```
<?php
$varLog = True; // asignar el valor TRUE a $varLog
$varLog = true; // asignar el valor TRUE a $varLog
$varLog = TRUE; // asignar el valor TRUE a $varLog
?>
```

- **integer**

Un entero representa un número entero incluyendo al cero, y en PHP se manejan los enteros con signo para manejar los números negativos.
Los enteros pueden ser especificados en notación decimal (base-10), hexadecimal (base-16) u octal (base-8), opcionalmente precedidos por un signo (- o +).
Si usa la notación octal, debe preceder el número con un *0* (cero), para usar la notación hexadecimal, preceda el número con *0x*.

Ejemplo números enteros

```
<?php
$a = 567; // numero decimal
$a = -567; // un numero negativo
```



```
$a = 0567; // numero octal (

conversión a octal 7 + (8*6) + (8*8*5))
$a = 0x1A; // numero hexadecimal (equivalente al 26 decimal)
?>
```

El tamaño de un entero (igual que en C y C++) es dependiente de la plataforma, por lo general se almacenan como enteros de 32 bits con signo. PHP no soporta enteros sin signo.

- **float** (número de punto-flotante, también conocido como '**double**')

Los números de punto flotante almacenan los números reales en PHP. Son también conocidos como "flotantes", "dobles" o "números reales". A continuación se muestran ejemplos de su utilización:

```
<?php
$a = 4.567;
$b = 4.2e3; // notación exponencial, en potencias de 10
$c = 7E-10; // notación exponencial, en potencias de 1 / 10
?>
```

- **string**

Un valor **string** es una cadena de caracteres. En PHP, un caracter se almacena como un byte, por lo tanto maneja exactamente 256 caracteres diferentes. Por lo tanto PHP no ofrece soporte Unicodee nativo. Para trabajar con el soporte Unicode se pueden utilizar las funciones **utf8_encode()** y **utf8_decode()**. Teóricamente PHP no impone ninguna limitación sobre el tamaño de las cadenas de caracteres, por lo tanto los programadores no deben ocuparse del manejo del crecimiento de las cadenas.
Unal de cadena puede especificarse en tres formas diferentes: comillas simples, comillas dobles o sintaxis heredoc. A continuación se muestra un ejemplo.

```
<?php
echo 'ejemplo de comilla simple';
$comillaSimple = 'Hola';

echo ''ejemplo de comilla doble'';
$comillaDoble = ''Hola'';


$cadena = <<<EJHERE
Ejemplo de una cadena HEREDOC
Estas cadenas pueden tener varias líneas
El inicio se específico con los tres menores y un
Nombre de tag para finalizar
que se extiende por varias líneas
EJHERE;
```

Dos tipos compuestos:

- **array**

Un array en PHP permite representar un arreglo de elementos, generalmente una matriz de múltiples dimensiones. Un arreglo en PHP es mapa ordenado. Un mapa ordenado es una estructura de datos donde los datos son almacenados por claves y se realiza un mapeo clave → valor.
En PHP, este tipo puede ser utilizado como un tipo de datos abstracto permitiendo la implementaciones de arreglos, matrices, vectores, listas, pilas, colecciones, diccionarios, árboles.
Los arreglos se cubrirán más adelante en el resto de este manual

- **object**

PHP permite desarrollar programas orientados por Objetos. En la versión 4.0 ofrece el concepto de clases y herencia. En la versión 5.0 ofrece además los conceptos de encapsulamiento y polimorfismo. PHP maneja el concepto de clases y el concepto de instancias de las clases u objetos.
Los objetos se cubrirán más adelante en el resto del presente manual.

Adicionalmente, PHP maneja dos tipos especiales.

- **resource**

Un recurso es una variable especial conteniendo una referencia a un recurso externo. Los recursos son creados y usados por funciones especiales.
PHP ofrece un conjunto de funciones para el manejo de los recursos. También existen funciones que crean, usan o destruyen recursos PHP. La función **is_resource()** puede ser usada para determinar si una variable es un recurso y **get_resource_type()** devolverá el tipo de recurso.

- **NULL**

En PHP **NULL** indica que una variable no tiene valor. **NULL** es el único valor posible para este tipo de datos. Una variable es considerada NULL si se ha asignado la constante **NULL** a la variable, no ha sido definida con valor alguno, ha sido eliminada con la instrucción **unset()**.

Conversión de Tipos de datos

En PHP se pueden realizar conversiones explícitas sobre las variables, empleando la función **settype()** sobre la variable. También se puede tomar el tipo de datos de una variable con la función gettype().

Otra manera de realizar conversiones explícitas sobre las variables es a través de operaciones de "casting".

```
<?php
$cad = "10";          // $cad es una cadena
$numS = "$cad";       // $numS es una cadena
$numC = (int) $numS;    // $numC es un entero
                  // se fuerza al tipo entero (casting)
$val = 10;

// En este caso los valors y los tipos son lo mismo
if ($numC === $val) {
   echo "Ejemplo de casting";
}
?>
```

Existen funciones para determinar si una variable contiene algún tipo de datos en especial. Estas funciones devuelven valores enteros, 0 indica que la variable no corresponde con el tipo de datos consultados, 1 indica que si corresponde.

is_array

is_bool

is_double

is_float

is_int

is_integer

is_long

is_null

is_numeric

is_object

is_real

is_resource

is_scalar

is_string

VARIABLES

PHP no es un lenguaje de programación fuertemente tipado; por lo tanto, no es necesario declarar las variables ni tampoco requiere asociarle un tipo de datos. En PHP, es posible que una variable tome el valor de diferentes tipos de datos durante la ejecución de un programa. En PHP las variables se representan con el carácter dólar ($) seguido por el nombre de la variable. PHP es un lenguaje "case sensitive", por lo tanto es importante considerar si el nombre de la variable contiene minúsculas y mayúsculas.

Los nombres de variables en PHP siguen las siguientes reglas de formación. Un nombre de variable valido debe empezar con una letra o "underscore", seguido de cualquier número de letras, números y rayas. A continuación se ejemplifica la utilización de variables.

```
<?php
$nombre = "Juan";
$Nombre = "Juan 2";
echo "$nombre, $Nombre";      // produce como salida "Juan, Juan 2"

$3invalida = 'prueba';    // invalido el nombre, comienza con un numero
$_4valida = 'prueba';    // valido, comienza con un underscore
$válida = 'prueba';    // valido, comienza con 'ä' ASCII 228 (Extendido)
?>
```

En PHP, las variables siempre se asignan **por valor**. Cuando una variable es asignada por valor se crea una copia del valor de la asignación en la variable. Por lo tanto al cambiarse el valor de la variable sólo cambia la variable asignada y no la expresión original.
PHP ofrece adicionalmente otra forma de asignar valores a las variables; **por referencia**. En este caso la variable asignada almacena la referencia (apuntador) a la variable original. Los cambios en cualquiera de las dos variables afectan a ambas. Desde el punto de vista de eficiencia, al no copiarse los valores de las variables la asignación ocurre más rápidamente.

Para trabajar con valores por referencia, es necesario colocar el símbolo ampersand "&" al comienzo de la variable cuyo valor se está asignando. A continuación se muestra un ejemplo:

```
<?php
$nombre1 = 'Henry';            // Asigna el valor 'Henry' a $nombre1
$nombre2 = &$nombre1;          // $nombre2, referencia a $nombre1
$nombre2 = "El nombre es $nombre2";  // Modifica también $nombre1
echo $nombre1;              // Salida: El nombre es Henry
echo $nombre2;              // Salida: El nombre es Henry
?>
```

Variables predefinidas

En PHP existen variables predefinidas (llamadas también globales), las cuales se encuentran disponibles para cualquier "script" en ejecución.
Las variables **globales** en PHP se almacenan en arreglos (en futuras secciones de este manual serán cubiertos los arreglos). A continuación se muestran ejemplos de estas variables.

$GLOBALS

Contiene una referencia a cada variable global. Los índices de esta matriz (claves) son los nombres de las variables globales.

$_SERVER

Variables definidas por el servidor web .

$_GET

Variables obtenidas en el “script” a través del modo GET del protocolo http.

$_POST

Variables obtenidas en el “script” a través del modo POST del protocolo http.
.

$_COOKIE

Variables obtenidas en el “script” a través de las “cookies” en http.

$_FILES

Variables obtenidas en el “script” a través de la lectura de archivos via http.

$_ENV

Variables definidas por el ambiente o entorno de programación.

$_REQUEST

Variables obtenidas en el “script” a través de cualquier modo (GET o POST) del protocolo http.

$_SESSION

Variables registradas en la sesión del “script”.

Alcance o ámbito de las variables

El alcance de una variable es el contexto donde la variable se encuentra definida y puede ser utilizada:

```
<?php
$var = 1;
// resto del código puede utilizar la variable $var, siempre
// y cuando no se creen functions
?>
```

Aquí, la variable *$var* se considera como una variable global a todo el “script” y puede ser utilizada en cualquier parte del mismo. En el caso de declarar una variable local en una función , el alcance es local a la función y no puede ser utilizada fuera de la función (las funciones serán cubiertas en secciones posteriores a este manual) . A continuación se muestra un ejemplo.

```
<?php
$varG = 1; /* alcance global */

function prueba()
{
  echo $varG; /* referencia local a una variable global */
}

prueba();
?>
```

Este script no producirá salida, ya que la instrucción echo asume que la variable $varG es local a la función prueba. Se debe recordar que las variables en PHP no se declaran. Si se requiere utilizar la variable $varG dentro de la función, se codifica el siguiente script.


```
<?php
$varG = 1; /* alcance global */

function prueba()
{
  global $varG; // ahora la variable global se encuentra disponible

  echo $varG; /* referencia local a una variable global */
}

prueba();
?>
```

Este caso producirá como salida 1.
Otra manera de acceder a las variables globales es utilizando la matriz *$GLOBALS*. Esto se demuestra en el siguiente ejemplo

```
<?php
$var1 = 1;
$var2 = 2;

function prueba()
{
  $GLOBALS["var1"] = $GLOBALS["var2"] + $GLOBALS["var2"];
}

prueba();
echo $var2;
?>
```

Este caso producirá como salida 3.
En PHP, también se pueden utilizar variables estáticas empleando la palabra clave ***static***. Las variables estáticas no pierden los últimos valores almacenados. A continuación se muestra un ejemplo:

```
<?php
function prueba ()
{
  $var = 0;
  echo $var;
  $var--;
}
?>
```

En este caso, esta función cada vez que se ejecuta asigna a *$var* el valor *0*. La instrucción *$var--*, que decrementa la variable, no es útil, porque al finalizar la función la variable *$var* desaparece. Si la variable $var se define como estática:

```
<?php
function prueba()
{
  static $var = 0;
  echo $var;
  $var--;
}
?>
```

En este caso, cada vez que se ejecuta la función prueba(), se decrementará el valor de *$var*.

Variables doblemente referenciadas

En PHP, se conocen como variables variables. A continuación se ilustra con un ejemplo

```
<?php
$var1 = 3;
$var2 = 'var1'; // asigna a una variable, el nombre de otra variable
echo $$var2; // $$ es una variable doblemente referenciada, dos singos $
?>
```

En este caso, se produce como pantalla el resultado 3.

CONSTANTES

Una constante es dato cuyo valor no cambia durante la ejecución de un programa. Las constantes también pueden ser utilizadas para mejorar la legibilidad de un programa. Las constantes se definen igual que las variables, pero por convención suelen asociarse nombres en mayúsculas

El alcance de una constante es global.

En PHP, una constante se define empleando la función **define()**. Una vez definida, no puede ser modificada ni eliminada .
Solo se puede definir como constantes valores de tipo primitivo (**boolean**, **integer**, **float** y **string**).
Para obtener el valor de una constante solo es necesario especificar su nombre. A diferencia de las variables, *no* se debe especificar el singo *$*. Se puede utilizar la función **constant()**, para obtener el valor de una constante. Para obtener la lista de todas las constantes definidas se utiliza la función **get_defined_constants()**.

La función **defined()** permite comprobar la existencia de una constante.
A continuación, se enumeran las las diferencias entre constantes y variables

- Las constantes no son precedidas por un símbolo de dolar ($)
- Las constantes solo pueden ser definidas usando la función ***define()*** .
- Las constantes son visibles en cualquier contexto.
- Las constantes no pueden cambiar de valor una vez definidas
- Las constantes sólo almacenan valores escalares

A continuación se muestra un ejemplo

```
<?php
define("IVA",15.0);
echo IVA; // imprime 15.0
?>
```

EXPRESIONES

Las expresiones son las instrucciones más básicas de la programación en PHP. Las expresiones más simples consisten en asignación de variables y pueden ser agrupadas en funciones, módulos y clases en el caso de la programación orientada por objetos.

Las funciones permite agrupar un conjunto de expresiones o instrucciones. A continuación, se muestra la siguiente función:

```
<?php
function f1 () {
  $a = 5;
  $b = 3;
  return $a*$b;
}
?>
```

Las funciones en PHP, son expresiones que al evaluarse tienen el valor que retornan. Como f1() devuelve 15, el valor de la expresión 'f1()' es 15.

Los valores de las expresiones en PHP no se limitan únicamente a valores simples (esclalares) también pueden trabajar con arreglos y con objetos.

El símbolo **";"** sirve como separador de expresiones (instrucciones) en PHP.

Al igual que los lenguajes C y Java, PHP ofrece los operadores de incremento y decremento. Por ejemplo **$x = $cont++** (**post-incremento**, evalúa la expresión primero y después incrementa, en el caso de una asignación no

tiene ningún efecto; pero si se tratara de una expresión condicional como **if ($cont++ == 3)**, evalúa primero la expresión y después incrementa el valor de $cont). En el caso de pre-incremento **$x = ++$cont**, se realiza primero el incremento antes de evaluar la expresión. Las expresiones de decremento **"—"**.

PHP ofrece **expresiones del tipo lógico**, estas devuelven valores **TRUE** o **FALSE**. Estas expresiones realmente son evaluadas a valores 0 o 1 **FALSE o TRUE**, respectivamente. PHP soporta las **expresiones relacionales** > (mayor que), >= (mayor o igual que), == (igual que), != (distinto), < (menor que) y <= (menor o igual que). Estas expresiones al evaluarse devuelven valores lógicos u por lo tanto son utilizadas en combinación con **expresiones condicionales** (**if**) y repetitivas (**while**) .

Otro tipo de expresiones disponibles en PHP son la **combinación de los operadores con la asignación**. Por ejemplo, en PHP **$x+=2**; es equivalente a $x = $x + 2; . Cualquier operación binaria puede ser usada de esta manera, por ejemplo '**$x -= 5**' (restar 5 del valor de $x), '**$x *= 7**' (multiplicar el valor de $x por 7), etc.

PHP maneja un tipo de expresión condicional también disponible en los lenguajes de Programación C y Java:

```
<?php
($x > 0) ? $y = 1: $y = 2
?>
```

Si el valor de la primera sub-expresión (la identificada antes del "?") es verdadero, entonces se evalúa la segunda sub-expresión (la identificada antes de los : ") , si no, se evalúa la tercera sub-expresión.

A continuación se muestra un ejemplo de la utilización de expresiones:

```
<?php
function duplicar($i) {
  return $i*2;
}
$x = $y = 5;        /* asignar el valor cinco a las variables $x y $y */
$z = $x++;          /* postincremento
$w = $s = ++$y;     /* preincremento
/* en este punto, tanto $w como $z y $s,  son iguales a 6 */

$d = duplicar($z++);  /* asignar el doble del valor de $z antes
                del incremento, 2*6 = 12 */
$g = duplicar(++$z);  /* asignar el doble del valor de $z después
                del incremento, 2*7 = 14 a $g */
$h = $g += 10;      /* $g y $h, valen 24
?>
```

OPERADORES

Los operadores en PHP se clasifican según el número de operadores manejados. Existen operadores **unarios** como el "-", $x = -3; o el operador de negación "!". En segundo lugar se encuentran los operadores **binarios** , los cuales son la mayoría de las expresiones. El tercer tipo de operadores son los **ternarios**, un solo operador el condicional (op1 ? op2: op3).

La **precedencia de un operador** indica el orden de evaluación de los operadores en una expresión. Por ejemplo el operador * tiene mayor precedencia que el operador +; por lo tanto si se coloca $z = 3 + 2 * 5 (devuelve el valor de 13). En caso de tener duda de la precedencia de los operadores en PHP o asignar un orden de evaluación requerido se pueden utilizar paréntesis; por ejemplo $x = (3+2)*5 (devuelve el valor de 25).

La siguiente tabla indica la precedencia de los operadores en PHP, con aquellos de mayor precedencia listados al comienzo de la tabla. Los operadores en la misma línea tienen la misma precedencia, en cuyo caso la asociatividad decide el orden para evaluarlos.

Precedencia de Operadores

Asociatividad	Operadores
no-asociativo	new

Asociatividad	Operadores
derecha	[
no-asociativos	+ + --
no-asociativos	! ~ - (int) (float) (string) (array) (object) @
izquierda	* / %
izquierda	+ - .
izquierda	<< >>
no-asociativos	< <= > >=
no-asociativos	== != === !==
izquierda	&
izquierda	^
izquierda	|
izquierda	&&
izquierda	||
izquierda	? :
derecha	= += -= *= /= .= %= &= |= ^= <<= >>=
izquierda	and
izquierda	xor
izquierda	or
izquierda	,

La asociatividad de izquierda quiere decir que la expresión es evaluada de izquierda a derecha, la asociatividad de derecha indica lo contrario.

Operadores matemáticos en PHP

En la siguiente tabla se muestran los operadores matemáticos de PHP.

Ejemplo	Nombre	Resultado
-$x	Cambio de signo	$x, cambiado de signo
$x + $y	Adición	Suma de $x y $y.
$x - $y	Substracción	Diferencia entre $x y $y.
$x * $y	Multiplicación	Producto de $x y $y.
$x / $y	División	Cociente de $x y $y.
$x % $y	Módulo	Resto de $x dividido por $y. (operador modulo)

Operadores de Asignación

PHP define operadores de asignación, estos por lo general asignan el valor de la variable.

```
<?php

$x = ($y = 2) + 3; // $x es igual a 5, y $y es igual a 2.

?>
```

El operador de asignación se puede combinar con cualquier operador binario (matemático, lógico, etc), por ejemplo para el caso de los matemáticos.

Ejemplo	Nombre	Resultado
$x += $y	Adición	A $x le asigna la suma de $x y $y.
$x -= $y	Substracción	A $x le asigna la diferencia entre $x y $y.
$x *= $y	Multiplicación	A $x le asigna el producto de $x y $y.
$x /= $y	División	A $x le asigna el cociente de $x y $y.
$x %= $y	Módulo	A $x le asigna el resto de $x dividido por $y. (operador modulo)

Operadores “Bitwise” (BIT a BIT)

Los operadores bit a bit permiten en PHP trabajar la representación 0 a 1 de una carácter aplicando operaciones lógicas.

Ejemplo	Nombre	Resultado
$x & $y	Y	Los bits que están en 1 tanto en $x como en $y son activados.
$x | $y	O	Los bits que están en 1 ya sea en $x o en $y son activados.
$x ^ $y	O exclusivo (xor)	Los bits que estén en 1 en $x o $y, pero no en ambos, son activados.
~ $x	No	Los bits que estén en 1 en $x son desactivados, y vice-versa.
$x << $y	"Shift" a izquierda	Desplaza los bits de $x, $y pasos a la izquierda (cada operación de estas en binario , equivale en decimal a "multiplicar por dos") .
$x >> $y	"Shift" a derecha	Desplaza los bits de $x, $y pasos a la derecha (cada operación de estas en binario , equivale en decimal a "dividir por dos")

Operadores Relacionales

Los operadores relacionales, como su nombre indica, permiten comparar o relacionar dos valores. Estos operadores devuelven el valor de true en caso de cumplirse la condición de comparación, false en caso contrario.

Ejemplo	Nombre	Resultado
$x == $y	Igual	**TRUE** si $x es igual a $y.
$x === $y	Idéntico	**TRUE** si $x es igual a $y, y son del mismo tipo. (A partir de PHP 4)
$x != $y	Diferente	**TRUE** si $x no es igual a $y.
$x <> $y	Diferente	**TRUE** si $x no es igual a $y.
$x !== $y	No idénticos	**TRUE** si $x no es igual a $y, o si no son del mismo tipo. (A partir de PHP 4)
$x < $y	Menor que	**TRUE** si $x es menor que $y.
$x > $y	Mayor que	**TRUE** si $x es mayor que $y.
$x <= $y	Menor o igual que	**TRUE** si $x es menor o igual que $y.
$x >= $y	Mayor o igual que	**TRUE** si $x es mayor o igual que $y.

Operadores de Control de Errores

PHP ofrece soporte para un operador de control de errores. Cuando el signo de arroba (@) es colocado al comienzo de una expresión en PHP, cualquier mensaje de error generado a causa de esa expresión será ignorado.

```
<?php
$arch = @file ('archivoInexistente') or
   die ("Error: abriendo el archivo : '$php_errormsg'");
?>
```

Nota: El operador @ trabaja sólo sobre **expresiones**.

Operadores de ejecución de comandos del sistema de operación

PHP soporta un operador de ejecución: las comillas invertidas (``). El uso del operador de comillas invertidas es idéntico al de **shell_exec()**.
A continuación se muestra un comando Linux para mostrar la lista de procesos de ejecución en el sistema

```
<?php
$procesos = `ps -edaf`;
echo "<pre>$procesos</pre>";
?>
```

Operadores de Incremento/Decremento

PHP ofrece soporte de operadores de pre- y post-incremento y decremento, estilo-C o Java.

Ejemplo	Nombre	Efecto
++$x	Pre-incremento	Incrementa $x en uno, y luego devuelve $x.
$x++	Post-incremento	Devuelve $x, y luego incrementa $x en uno.

Ejemplo	Nombre	Efecto
--$x	Pre-decremento	Decrementa $x en uno, luego devuelve $x.
$x--	Post-decremento	Devuelve $x, luego decrementa $x en uno.

Operadores Lógicos

PHP ofrece operadores para trabajar sobre expresiones lógicas.

Ejemplo	Nombre	Resultado
$x and $y	Y	**TRUE** si tanto $x como $y son **TRUE**.
$x or $y	O	**TRUE** si cualquiera de $x o $y es **TRUE**.
$x xor $y	O exclusivo (Xor)	**TRUE** si $x o $y es **TRUE**, pero no ambos.
! $x	No	**TRUE** si $x no es **TRUE**.
$x && $y	Y	**TRUE** si tanto $x como $y son **TRUE**.
$x || $y	O	**TRUE** si cualquiera de $x o $y es **TRUE**.

Operadores sobre Strings (cadena de caracteres)

Existen dos operadores para datos tipo **string**. El primero es el operador de **concatenación ('.')**. El segundo es el operador de **asignación sobre concatenación ('.=')**.

```
<?php
$a = " Hola ";
$b = $a . "Chao"; // $b contiene" Hola Chao"

$a = " Hola ";
$a .= "Chao"; // $a contiene " Hola Chao"
?>
```

Operadores de Matrices

En las siguientes secciones de este manual se cubrirá en detalles el tema de arreglos y matrices. En esta sección únicamente se indican los operadores.

Ejemplo	Nombre	Resultado
$M1 + $M2	Unión	Unión de $M1 y $M2.
$M1 == $M2	Igualdad	**TRUE** si $M1 y $M2 tienen los mismos elementos.
$M1 === $M2	Identidad	**TRUE** si $M1 y $M2 tienen los mismos elementos en el mismo orden.
$M1 != $M2	No-igualdad	**TRUE** si $M1 no es igual a $M2.
$M1 <> $M2	No-igualdad	**TRUE** si $M1 no es igual a $M2.
$M1 !== $M2	No-identidad	**TRUE** si $M1 no es idéntico a $M2.

Operadores de Objetos

En las siguientes secciones de este manual se cubrirá en detalles el tema de programación orientada por objetos. En esta sección únicamente se indican los operadores.
PHP tiene un operador único de tipo: *instanceof*. *instanceof* es usado para determinar si un objeto dado es de una clase de objeto especificada.
El operador ***instanceof*** fue introducido en PHP 5. Antes de esta versión, **is_a()** era utilizado.

LECCION 5. ESTRUCTURAS CONDICIONALES

Las estructuras condicionales en PHP permiten controlar el control de ejecución de un “script”. En este tipo de instrucciones se evalúa una condición (por lo general es una expresión) y dependiendo de su valor se ejecutan o no porciones del “script”. Es un constructor muy típico en los lenguajes de programación.

Instrucción if

PHP caracteriza una estructura if similar a la del lenguaje de Programación C. A continuación se indica la sintaxis de la instrucción

```
<?php

if (expr)

   inst
?>
```

```
<?php

if (expr) {
   inst1;
   ....
   instN;
?>
```

Si el valor de la expresión “expr” es verdadero se ejecutan las instrucciones del bloque de if. Es obligatorio encerrar la expresión entre paréntesis; en caso de requerirse ejecutar más de una instrucción en el bloque del if, éstas deben agruparse entre llaves.

Las instrucciones if , soportan múltiples niveles de anidamiento. Por lo tanto se pueden colocar instrucciones if dentro de las instrucciones de un bloque de if.

Instrucción if *else*

PHP caracteriza una estructura if else similar a la del lenguaje de Programación C. A continuación se indica la sintaxis de la instrucción

```
<?php

if (expr)

   inst

else

   instXXX

?>
```

```
<?php

if (expr) {

   inst1;
   ....
   instN;
}
else {
  instXXX1;
   ....
   instXXXN;
}
?>
```

Si el valor de la expresión “expr” es verdadero se ejecutan las instrucciones del bloque de if, en caso constrario se ejecutan las instrucciones del bloque de else. Notese que los bloques de if y de else debe ir en parejas y cada cual abre y cierra su bloque.

Instrucción if else elseif

La instrucción if, permite implementar una instrucción condicional simple (caso verdadero) . La instrucción if else permite implementar una instrucción condicional doble (caso verdadero o caso falso). La instrucción elseif permite implementar un caso de múltiples opciones (mutuamente excluyentes)

```
<?php

if (expr) {

   // caso 1;

}

elseif (expr2) {

  // caso 2;

}

....

elseif (exprN) {
  // caso N;
}
else {
 // ninguno de los casos
}


?>
```

Puede haber varios *elseif*s dentro de la misma instrucción *if*. La primera expresión *elseif* (si existe) que se evalúe como **TRUE** se ejecutaría. La sentencia *elseif* se ejecuta sólo si la expresión *if* precedente y cualquier expresión *elseif* precedente se evalúan como **FALSE**, y la expresión *elseif* actual se evalúa como **TRUE**. (**mutuamente excluyente**)

Instrucción switch

La instrucción *switch* permite expresar las instrucciones de múltiples condiciones de forma más compacta. Es ideal para los casos de selección múltiple. Por lo general se utiliza para analizar los posibles valores a tomar por una variable. Dependiendo de los valores para la variable se ejecutan las acciones.

> **Nota:** Tener en cuenta que al contrario que otros lenguajes de programación, **continue** se aplica a switch y funciona de manera similar a ***break*.**

Con el siguiente ejemplo se ilustra la utilización de la instrucción switch.:

```
<?php
if ($x == 0) {
  print "x es 0";
} elseif ($x == 1) {
  print "x es1";
} elseif ($x == 2) {
  print "x es 2";
}

switch ($x) {
  case 0:
     print "x es 0";
     break;
  case 1:
     print "x es 1";
     break;
  case 2:
     print "x es 2";
     break;
  default: // se ejecuta cuando no se cumple ningún case
     print "no es ninguno de los esperados";
}
?>
```

La instrucción *switch* se ejecuta línea por línea. Cuando se encuentra una instrucción *case* con un valor que coincide con el valor de la expresión *switch*, PHP comienza a ejecutar las instrucciones. PHP continúa ejecutando las instrucciones hasta el final del bloque *switch*, o la primera vez que vea una instrucción *break*. Si no se escribe una instrucción *break* al final de una lista de sentencias case, PHP seguirá ejecutando las sentencias del siguiente case. Por ejemplo:

```
<?php
 switch ($x) {
    case 0:
       print "x es 0";
    case 1:
       print "x es 1";
    case 2:
       print "x es 2";
 }
?>
```

En este ejemplo, si $x es igual a 0, PHP ejecutaría todas las sentecias print. Si $x es igual a 1, PHP ejecutaría las últimas dos sentencias print y sólo si $x es igual a 2, sólo se ejecuta el último print.

La lista de instrucciones de un case puede también estar vacía, pasando el control a la lista de instrucciones del siguiente case.
La instrucción *case* se utiliza con cualquier expresión evaluada como un tipo simple, es decir, números enteros o de punto flotante y cadenas de texto. No se pueden usar aquí ni arreglos ni objetos a menos que se conviertan a un tipo simple.

Sintaxis alternativa de estructuras de control

PHP ofrece una sintaxis alternativa para algunas de sus estructuras de control: if, while, for, foreach, y switch. En cada caso, la forma básica de la sintaxis alternativa es cambiar la llave de apertura por dos puntos (**:**) y la llave de cierre por **endif;**, **endwhile;**, **endfor;**, **endforeach;**, o **endswitch;**, respectivamente.

```
<?php if ($a == 5): ?>

A es igual a 5

<?php endif; ?>
```

En el ejemplo anterior, el bloque HTML "A es igual a 5" se anida dentro de una sentencia if escrita en la sintaxis alternativa. El bloque HTML se mostraría solamente si $a es igual a 5.

La sintaxis alternativa también se aplica a else y elseif. El siguiente ejemplo es una estructura if con elseif y else en el formato alternativo:

```
<?php

if ($a == 5):

    echo "a igual 5";

    echo "...";

elseif ($a == 6):

    echo "a igual 6";

    echo "!!!";

else:

    echo "a no es 5 ni 6";

endif;

?>
```

Nota: PHP no soporta la mezcla de sintaxis en el mismo bloque de control.

LECCION 6. ESTRUCTURAS REPETITIVAS

Las estructuras repetitivas en PHP permiten controlar **el control de ejecución** de un "script". En este tipo de instrucciones se ejecuta un bloque de instrucciones de manera cíclica o repetitiva mientras se cumpla la condición (por lo general es una expresión). Es un constructor muy típico en los lenguajes de programación.

Instrucción while

Los ciclos while permiten ejecutar un bloque de instrucciones mientras se cumpla una condición. La condición puede ser cualquier expresión que retorne un valor bolean (o 0 o 1). La sintaxis de la instrucción while es como se muestra a continuación:

```
while (cond) {
  instruccion1;
  ....
  instruccionn;
}
```

En PHP, la condición debe especificarse obligatoriamente entre paréntesis. Si el bloque de instrucciones contiene una sola instrucción no es necesario colocar las llaves . Notar que en el caso del while se evalúa primero la condición, si el valor es TRUE ejecuta todas las instrucciones del ciclo; después de ejecutar la última vuelve a revisar la condición y así hasta que la condición no se cumpla.

El siguiente ejemplo suma los números del 1 al 10:

```
<?php
 $sum = 0;
 $i = 1;
 while ($i <= 10) {
    $sum += $i;
    $i++;

 }

 print $sum;


?>
```

Instrucción do while

La instrucción do..while es similar a la instrucción while, únicamente se diferencian en el hecho del momento cuando se evalúa la condición. En el caso del while se chequea antes de ejecutar el bloque de instrucción; mientras para el caso do..while se ejecuta primero el bloque de instrucciones y luego se revisa la condición. En el caso del *do..while* se garantiza la ejecución de el menos una vez del bloque de instrucciones.

La sintaxis de la instrucción do..while es como se muestra a continuación:

```
do {
  instruccion1;
  ....
  instruccionn;
} while (cond)
```

El siguiente ejemplo suma los números del 1 al 10, empleando un do .. while:

```
<?php

 $sum = 0;

 $i = 1;

 do {

    $sum += $i;
```

```
    $i++;

 } while ($i < 10) // fíjarse en el cambio del < por <=
 print $sum;


?>
```

Instrucción for

La sintaxis de la instrucción for se muestra a continuación:

```
for (expr1; expr2; expr3) {

  instruccion1;
  ...
  instruccionn;
}
```

El bloque de instrucción for se compone de tres partes separadas por ";". La primera parte (expr1) se ejecuta una sola vez antes de comenzar el bloque de instrucciones, aquí se deben colocar las instrucciones relacionadas con la inicialización. La segunda parte (expr2) contiene la condición de parada del ciclo, si (expr2) se evalúa como trae se ejecuta el ciclo en caso contrario se finaliza la ejecución. La tercera parte contiene las instrucciones ejecutadas después de que se ejecuta (instruccionn) después de cada iteración del ciclo.

Cualquiera de estas tres partes puede puede estar vacía. En el caso de que *expr2* sea vacía significa que el ciclo se ejecuta infinitamente. Se puede utilizar la instrucción break para finalizar la ejecución de un lazo.

El siguiente ejemplo suma los números del 1 al 10:

```
<?php

 $sum = 0;

 for ($i = 1; $i <= 10; $i++) {

    $sum += $i;


 }

 print $sum;


 $sum = 0;

 $i = 1;

 for (;$i <= 10;) {

    $sum += $i;

    $i++;

 }

 print $sum;



?>
```

Instrucción for each

Esta instrucción se encuentra disponible a partir de PHP 4. Esta instrucción permite iterar sobre los elementos de una matriz (las matrices serán cubiertas más adelante en este curso). foreach funciona únicamente con matrices y generará un error si se intenta utilizar con otro tipo de datos ó variables no inicializadas. Existen dos sintaxis para la instrucción foreach:

```
foreach(expresionMatriz as $valor) {

  instruccion1;
  ....
  instruccionn;
}


foreach(expresionMatriz as $clave => $valor) {
  instruccion1;
  ....
  instruccionn;
}
```

LECCION 7. OTRAS ESTRUCTURAS DE CONTROL

break

break termina la ejecución de la estructura actual for, foreach, while, do-while o switch.

break acepta un argumento numérico opcional el cual indica de cuantas estructuras anidadas encerradas se debe salir.

```
<?php

$arr = array('uno', 'dos', 'tres', 'cuatro', 'pare', 'cinco');

while (list(, $val) = each($arr)) {

   if ($val == 'pare') {

      break;    /* Se puede también escribir 'break 1;' aquí. */

   }

   echo "$val<br />\n";

}

/* Usando el argumento opcional. */

$i = 0;

while (++$i) {

   switch ($i) {

   case 5:

      echo "En 5<br />\n";

      break 1;  /* Sólo sale del switch. */

   case 10:

      echo "En 10; saliendo<br />\n";

      break 2;  /* Sale del switch y del while. */

   default:

      break;

   }

}

?>
```

continue

continue se utiliza dentro de las estructuras iterativas para saltar el resto de la iteración actual del bucle y continuar la ejecución en la evaluación de la condición, y luego comenzar la siguiente iteración.

Nota: La sentencia switch se considera una estructura iterativa para los propósitos de continue.

continue acepta un argumento numérico opcional, que indica a cuántos niveles de bucles encerrados se ha de saltar al final. El valor por omisión es 1, por lo que salta al final del bucle actual.

```
<?php
while (list($clave, $valor) = each($arr)) {
    if (!($clave % 2)) { // saltar los miembros impares
        continue;
    }
    hacer_algo($valor);
}

$i = 0;
while ($i++ < 5) {
    echo "Exterior<br />\n";
    while (1) {
        echo "Medio<br />\n";
        while (1) {
            echo "Interior<br />\n";
            continue 3;
        }
        echo "Esto nunca se imprimirá.<br />\n";
    }
    echo "Ni esto tampoco.<br />\n";
}
?>
```

Omitir el punto y coma después del continue puede llevar a confusión. He aquí un ejemplo de lo que no se debe hacer.

```
<?php
for ($i = 0; $i < 5; ++$i) {
    if ($i == 2)
        continue
    print "$i\n";
}
?>
```

Se esperaría que el resultado fuera:

```
0
1
3
4
```

pero la salida de este script será:

```
2
```

debido a que continue print "$i\n"; se evalúa completo como una sola expresión, y así print se llama solamente cuando $i == 2 es verdadero. (El valor de retorno de print es pasado a continue como el argumento numérico.)

declare

El constructor declare es utilizado para fijar directivas de ejecución para un bloque de código. La sintaxis de declare es similar a la sintaxis de otros constructores de control de flujo:

```
declare (directive)
    statement
```

La sección directive permite que el comportamiento de declare sea configurado. Actualmente, sólo dos directivas están reconocidas: **ticks** y **encoding**.

La parte statement del bloque declare será ejecutada dependiendo de la directiva fijada en el bloque directive.

El constructor declare también se puede utilizar en el alcance global, afectando a todo el código siguiente (sin embargo, si el archivo con el declare fue incluido entonces no afectará al archivo padre).

```
<?php
// estos son lo mismo:

// se puede usar ésto:
declare(ticks=1) {
    // script entero aquí
}

// o se puede usar ésto:
declare(ticks=1);
// script entero aquí
?>
```


Ticks

Un tick es un evento que ocurre para cada sentencia tickable N de bajo nivel ejecutada por el intérprete dentro del bloque declare. El valor para N se especifica usando ticks=N dentro del bloque de declare de la sección directive.

No todas las sentencias son tickable. Por lo general, expresiones de condición y expresiones de argumento no son tickables.

Los eventos que ocurren en cada tick se especifican mediante la función **register_tick_function().** Tener en cuenta que más de un evento puede ocurrir por cada tick.

Ejemplo #1 Ejemplo de uso del tick

```
<?php

declare(ticks=1);

// Una función llamada en cada evento tick
function tick_handler()
{
   echo "tick_handler() llamado\n";
}

register_tick_function('tick_handler');

$a = 1;

if ($a > 0) {
   $a += 2;
   print($a);
}

?>
```

Ejemplo #2 Ejemplo de uso de ticks

```
<?php

function tick_handler()
{
 echo "tick_handler() llamado\n";
}

$a = 1;
tick_handler();

if ($a > 0) {
   $a += 2;
   tick_handler();
   print($a);
   tick_handler();
}
tick_handler();

?>
```

Encoding

Una codificación de script puede ser especificada para cada script usando la directiva encoding.

Ejemplo #3 Declarando un encoding para el script

```
<?php
declare(encoding='ISO-8859-1');
// código aquí
?>
```

Precaución
Cuando se combina con espacios de nombres, la única sintaxis legal para declarar es declare(encoding='...'); donde ... es el valor del encoding. declare(encoding='...') {} resultará en un error de análisis cuando se combina con espacios de nombres.

return

Si se llama desde una función, la sentencia return inmediatamente termina la ejecución de la función actual, y devuelve su argumento como el valor de la llamada a la función. return también pondrá fin a la ejecución de una sentencia eval() o a un archivo de script.

Si se llama desde el ámbito global, entonces la ejecución del script actual termina. Si el archivo script actual fue incluido o requerido con include o require, entonces el control es pasado de regreso al archivo que hizo el llamado. Además, si el archivo script actual fue incluido con include, entonces el valor dado a return será retornado como el valor de la llamada include. Si return es llamado desde dentro del fichero del script principal, entonces termina la ejecución del script. Si el archivo script actual fue nombrado por las opciones de configuración auto_prepend_file o auto_append_file en php.ini, entonces se termina la ejecución de ese archivo script.

include

La sentencia include incluye y evalúa el archivo especificado.

require

require es idéntico a include excepto que en caso de fallo producirá un error fatal de nivel E_COMPILE_ERROR

require_once

La sentencia require_once es idéntica a require excepto que PHP verificará si el archivo ya ha sido incluido y si es así, no se incluye (require) de nuevo.

include_once

La sentencia include_once incluye y evalúa el archivo especificado durante la ejecución del script. Es un comportamiento similar al de la sentencia include, siendo la única diferencia que si el código del fichero ya ha sido incluido, no se volverá a incluir. Como su nombre lo indica, será incluido sólo una vez.

goto

El operador goto puede ser usado para saltar a otra sección en el programa. El
punto de destino es especificado mediante una etiqueta seguida de dos puntos y
la instrucción es dada como goto seguida de la etiqueta del destino deseado.
Este goto no es completamente sin restricciones. La etiqueta de destino debe estar dentro del mismo archivo y contexto, lo que significa que no se puede saltar fuera de una función o método, ni se puede saltar dentro de uno. Tampoco se puede saltar dentro de cualquier clase de estructura de bucle o switch. Se puede saltar fuera de estos y un uso común es utilizar un goto en lugar de un break multi-nivel.

LECCION 8 PROGRAMACIÓN ESTRUCTURADA EN PHP

INTRODUCCIÓN A LOS CONCEPTOS DE PROGRAMACIÓN ESTRUCTURADA

Programación Modular

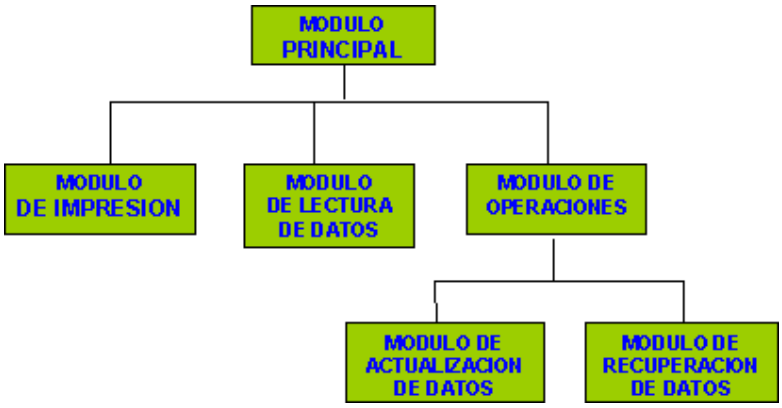
El programa se divide en módulos (partes independientes), cada una ejecuta una actividad o tarea y se codifican independientes de otros módulos.

Con esta técnica se genera un módulo denominado programa principal que controla a los otros módulos denominados subprogramas. Si la tarea asignada a un módulo es muy compleja se deberá subdividir dicho módulo en otros de menor complejidad. El proceso sucesivo de subdivisión de módulos continúa hasta que cada módulo tenga una tarea precisa que ejecutar.

Como los módulos son independientes, diferentes programadores pueden trabajar simultáneamente en diferentes partes de un mismo programa. Esto reduce los tiempos de diseño y de decodificación del programa.

Otra ventaja es que la depuración, la localización de errores y la modificación de código se hace mucho más fácil pues se puede verificar la correctitud de cada módulo por separado.

En la siguiente figura se visualiza un ejemplo de un programa dividido en módulos:



Programación Estructurada

Conjunto de técnicas que utiliza un número limitado de estructuras de control que minimizan la complejidad y los errores. La programación estructurada es el conjunto de técnicas que utiliza:

Diseño Descendente (Top - Down).

Es el proceso mediante el cual un problema se descompone en una serie de niveles o etapas. La metodología descendente consiste en efectuar una relación entre las sucesivas etapas de estructuración, relacionando unas con otras mediante entradas y salidas de información. Es decir, se descompone el problema en etapas o estructuras jerárquicas, tal que se puede considerar cada estructura desde dos puntos de vista: qué hace y cómo lo hace.

Recursos Abstractos.

La programación estructurada se auxilia de los recursos abstractos.

Descomponer un programa en términos de recursos abstractos consiste en descomponer una determinada acción compleja en función de un número de acciones más simples, capaces de ser ejecutadas por una computadora y que constituirán sus instrucciones.

Estructuras Básicas.

Las estructuras básicas son las denominadas estructuras de control clasificadas en secuenciales, selectivas y repetitivas. Un programa estructurado se puede escribir sólo con las tres estructuras de control mencionadas. Las estructuras de control tienen un solo punto de entrada y un solo punto de salida, esto hace que los programas estructurados sean más fáciles de leer y depurar.

Reglas para una buena Programación

- Diseñar algoritmos en etapas aplicando el método descendente (top-down)
- Dividir el algoritmo en partes independientes (módulos)

- Utilizar las técnicas de programación estructurada
- Dar importancia a las estructuras de datos
- Describir completa y claramente cada algoritmo/programa (documentación)
- Verificar y validar el programa con datos significativos

Introducción a los Subprogramas o Subalgoritmos

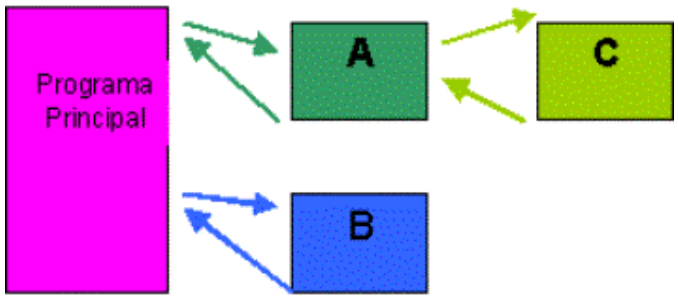
Tanto la programación modular como la programación estructurada utilizan el módulo como herramienta de descomposición de programas.

Un módulo es un conjunto de instrucciones que se procesan de una sola vez y que se refieren mediante un nombre por el que son llamados o invocados desde diferentes puntos del programa.

Los módulos, subprogramas o subalgoritmos, son un conjunto de instrucciones que constituyen parte de un programa, que conforman una unidad y tienen un objetivo específico.

Todo módulo tiene un punto de entrada, un punto de salida y puede ser una función o un procedimiento o subrutina.

En la próxima figura se muestra un ejemplo de invocación de subprogramas. El programa principal invoca al subprograma **A** el cual a su vez invoca al subprograma **C**. Cuando **C** se termina de ejecutar regresa el control del programa al módulo **A** el cual termina su ejecución y retorna a su vez el control al programa principal. Éste se continúa ejecutando hasta que invoca (llama) al subprograma **B** el cual se ejecuta y le devuelve el control. El programa principal continúa con su ejecución hasta la instrucción de finalizar.



IMPLEMENTACIÓN DE LOS CONCEPTOS DE PROGRAMACIÓN ESTRUCTURADA EN PHP

Programación Modular

PHP ofrece la programación modular a través de los módulos. Un módulo en PHP es un archivo .php (por lo general se coloca como extensión .**inc** – para abreviar incluye --) conteniendo scripts php . Un script php en la filosofía modular es un grupo de funciones.
La instrucción **include()** incluye y ejecuta las instrucciones del archivo de módulo especificado. También se puede utilizar **require()** si desea incluir un módulo.
En caso de tener programación modular donde se incluyen varios módulos, debe tener en cuenta que **un módulo sólo se puede incluir una vez**. La mejor manera de implementar esto en PHP es definir variables globales indicando si el módulo fue ya incluido antes de incluirlo.
Cuando el motor de ejecución de php encuentra la instrucción de include trabaja como si copiara todo el código del módulo en el archivo que lo incluye.


```
secundario.php
<?php

$instituto = 'gala';
$alumno = 'usted';

?>


principal.php
<?php

echo " $instituto $alumno"; // No escribe nada

include 'secundario.php';

echo "$instituto $alumno"; // Escribe gala usted


?>
```

La instrucción include puede colocarse en cualquier lugar en PHP, pero aplicando los principios de programación modular debería colocarse al principio del módulo.

```
<?php

function f1()
{
   global $instituto;

   include 'secundario.php';

   echo "Instituto $instituto ";
 // colocaría instituto gala
}

/* En este caso secundario.php se incluye dentro de f1 y su
  código no se encuentra disponible fuera de él
  */


  ?>
```

Cuando un archivo es incluido, el intérprete sale del modo PHP y entra en modo HTML en la primera línea del del archivo referenciado, y vuelve de nuevo al modo PHP después de procesar la última línea. Por esta razón, cualquier código dentro del archivo incluido que debiera ser ejecutado como código PHP debe ser encerrado dentro de los tags de comienzo y fin de PHP.

Subrutinas

PHP ofrece el concepto de funciones (functions) para descomponer un programa en sub-programas. la programación modular a través de los módulos. Un módulo en PHP es un archivo .php (por lo general se coloca como extensión .**inc** – para abreviar incluye --) conteniendo scripts php . Un script php en la filosofía modular es un grupo de funciones.

FUNCIONES EN PHP

Introducción

Una función se puede definir con la siguiente sintaxis:

```
<?php
function f1 ($par_1, $par_2, ..., $par_n)
{
   // código PHP
   // $valorretorno = ...
```

```
  return $valorretorno;
}
?>
```

Cualquier instrucción válida de PHP puede ser parte de una función, inclusive se pueden anidar funciones o definir clases dentro de funciones (las clases serán explicadas en la sección de programación orientada por objetos)
En PHP, no es necesario que las funciones sean definidas antes de realizar las llamadas.
En PHP, pueden definirse funciones condicionales.

```
<?php

$f1Ok = true;


f2();

if ($f1Ok) {
  function f1 ()
  {
   echo "Esta función no puede ser llamada hasta que se ejecute la funcion.\n";
  }
}


if ($f1Ok) // solo se puede llamar a f1 si el booleano es true
   f1();

function f2()
{
 echo "Esta función si se puede llamar de cualquier parte .\n";
}

?>
```

En PHP también se pueden anidar las funciones

```
<?php
function ppal()
{
  function secundaria()
  {
   echo "No existe hasta que se llame ppal().\n";
  }
}


ppal();

// ahora se puede llamar a secundaria
secundaria();

?>
```

PHP soporta un número variable de parámetros y parámetros por defecto. PHP ofrece las siguientes funciones predefinidas **func_num_args()**, **func_get_arg()**, y **func_get_args()** para el manejo de parámetros variables.

Parámetros en las funciones

Una función puede recibir una lista de parámetros. Los parámetros pueden ser variables o constantes separados por comas.

El tipo de parámetros por defecto de PHP es por valor. Por lo tanto al pasar una variable como parámetro a una función, la función crea una copia interna de la variable y esta no se modifica al finalizar la ejecución de la función. PHP también permite pasar parámetros por referencia.

```
<?php
function suma($x, $y)
{
  $x = $x + $y;
  return $x;
}
?>

$a = 5;
$b = 8;
$c = suma ($a,$b);
// $c vale 8, pero $a sigue valiendo 5, ya que es por valor
```

Para permitir a una función modificar sus parámetros estos deben pasarse **por referencia**.
En PHP, para pasar un parámetro por referencia debe colocarse el **símbolo ampersand (&)** al nombre del parámetro en la definición de la función:

```
<?php
function suma(&$x, &$y)
{
   $x = $x + $y;
   return $x;
}
$a = 5;
$b = 8;
$c = suma ($a,$b);
// $c vale 8, ahora $a sigue valiendo 8, ya que es por referencia
?>
```

En PHP, una función puede definir valores por defecto , para los tipos de datos primitivos

```
<?php
function f1 ($def = "hola")
{
  return "Saludar con $def.\n";
}
echo f1 ();
echo f1 ("epale");
?>
```

La salida del fragmento anterior es:

```
Saludar con hola.
Saludar con epale.
```

El valor por defecto debe ser una expresión constante.

Al utilizar parámetros por defecto, estos tienen que estar a la derecha de cualquier parámetro sin valor por defecto.

Funciones variables

PHP soporta el concepto de funciones variable. La utilidad de esta implementación es que se puede almacenar en una variable el nombre de la función y luego al evaluar la variable se ejecuta la funciòn.
Las funciones variables no funcionan con funciones predefinidas del lenguaje como con **echo()**, **print()**, **unset()**, **isset()**, **empty()**, **include()**, **require()**.

```
<?php
function f1()
{
  echo "Ejecutando f1 <br>\n";
```

```
}

function f2($x)
{
  echo "Ejecutando f2 con" . $x . " <br>\n";
}


$f = 'f1';
$f();       // llama f1()

$f = 'f2';
$f("p1");  // llama f2()

?>
```

Funciones internas (incluidas)

PHP se estandariza con muchas funciones y construcciones. También existen funciones que necesitan extensiones específicas de PHP compiladas, si no, aparecerán errores fatales "undefined function" ("función no definida"). Por ejemplo, para usar las funciones de image tales como imagecreatetruecolor(), PHP debe ser compilado con soporte para GD. O para usar mysql_connect(), PHP debe ser compilado con soporte para MySQL. Hay muchas funciones de núcleo que está incluidas en cada versión de PHP, tales como las funciones de string y de variable. Una llamada a phpinfo() o get_loaded_extensions() mostrará las extensiones que están cargadas en PHP. Muchas muchas extensiones están habilitadas por defecto.

Funciones anónimas

Las funciones anónimas, también conocidas como clausuras (closures), permiten la creación de funciones sin un nombre especificado. Son más útiles como valor de los parámetros de llamadas de retorno, pero tienen muchos otros usos.

Ejemplo #1 Ejemplo de función anónima

```
<?php

echo preg_replace_callback('~-([a-z])~', function ($coincidencia) {

    return strtoupper($coincidencia[1]);

}, 'hola-mundo');

// imprime holaMundo

?>
```

Las clausuras también se pueden usar como valores de variables; PHP automáticamente convierte tales expresiones en instancias de la clase interna Closure. Se asume que una clausura a una variable usa la misma sintaxis que cualquier otra asignación, incluido el punto y coma final:

Ejemplo #2 Ejemplo de asignación de variable de una función anónima

```
<?php
```

```
$saludo = function($nombre)

{

   printf("Hola %s\r\n", $nombre);

};



$saludo('Mundo');

$saludo('PHP');

?>
```

LECCION 9 REFERENCIAS EN PHP

Las Referencias en PHP son medios de acceder al mismo contenido de una variable mediante diferentes nombres. No son como los apuntadores de C; por ejemplo, no se puede realizar aritmética de apuntadores con ellas, realmente no son direcciones de memoria, etc. Las referencias son alias de la tabla de símbolos. En PHP el nombre de la variable y el contenido de la variable son cosas diferentes, por lo que el mismo contenido puede tener diferentes nombres. La analogía más próxima es con los archivos y los nombres de archivos de Unix - los nombres de variables son entradas de directorio, mientras que el contenido de las variables es el archivo en sí. Las referencias se pueden vincular a enlaces duros en sistemas de archivos Unix.

Hay tres operaciones básicas que se realizan usando referencias: asignar por referencia, pasar por referencia, y devolver por referencia.

Se puede pasar una variable por referencia a una función y así hacer que la función pueda modificar la variable. La sintaxis es la siguiente:

```
<?php

function foo(&$var)

{

   $var++;

}



$a=5;

foo($a);

// $a es 6 aquí

?>
```

Devolver por referencia es útil cuando se quiere usar una función para encontrar a qué variable debería estar vinculada una referencia. No se debe utilizar devolver por referencia para aumentar el rendimiento. El motor optimizará automáticamente esto por sí mismo. Se deben devolver referencias sólo cuando se tenga una razón técnicamente válida para hacerlo. Para devolver referencias use esta sintaxis (en la próxima lección se cubren los conceptos de claes para complementar este punto):


```
<?php

class foo {

   public $valor = 42;



   public function &obtenerValor() {

      return $this->valor;

   }

}



$obj = new foo;

$miValor = &$obj->obtenerValor(); // $miValor es una referencia a $obj->valor, que es 42.

$obj->valor = 2;

echo $miValor;              // imprime el nuevo valor de $obj->valor, esto es, 2.

?>
```

LECCION 10 PROGRAMACIÓN ORIENTADA POR OBJETOS EN PHP

INTRODUCCIÓN A LOS CONCEPTOS DE PROGRAMACIÓN ORIENTADA POR OBJETOS

Técnicas de programación

Los programadores han tenido que luchar con el problema de la complejidad durante mucho tiempo. Para ello se han válido de una técnica muy potente para tratar la complejidad: la abstracción.
La incapacidad de dominar en su totalidad los objetos complejos induce a la construcción de modelos mentales de partes del mismo.
Un modelo mental es una vista simplificada de cómo funciona la realidad de modo que se pueda interactuar con ella.
En esencia, el proceso de construcción de modelos es lo mismo que el diseño de software, pues éste produce un modelo pero que puede ser manipulado por una computadora.
Los mecanismos de abstracción que han conducido al desarrollo profundo de los objetos son: los procedimientos, los módulos, los tipos abstractos de datos y los objetos.

Procedimientos

Los procedimientos y funciones permiten definir tareas reunidas en una entidad que se puede reutilizar. Además proporciona la posibilidad de ocultamiento de la información.

Módulos

Es una técnica que proporciona la posibilidad de clasificar sus datos y procedimientos en una parte privada, sólo accesible dentro del módulo, y una parte pública, accesible fuera del módulo. El criterio a seguir en la construcción de un módulo es que si no se necesita algún tipo de información, no se debe tener acceso a ella. Este criterio es el ocultamiento de información.
Los módulos proporcionan un método efectivo de ocultamiento de información, pero no permiten realizar instanciación, que es la capacidad de hacer múltiples copias de las zonas de datos.

Tipos Abstractos de Datos

Un Tipo Abstracto de Datos (TAD) es un tipo de dato definido por el usuario/programador que se manipula de un modo similar a los tipos de datos definidos por el sistema. Un TAD define un conjunto de valores y de operaciones que se realizan sobre esos valores. Los TAD permiten hacer instanciación.

Objetos

Un objeto es un tipo abstracto de datos al que se le añaden compartición de código y reutilización. Los mecanismos básicos del manejo de objetos son: objetos, mensajes y métodos, clases e instancias y herencia. Una idea fundamental es la comunicación de dos objetos a través del paso de mensajes.
Llegado este momento de la evolución de los mecanismos de abstracción es que la orientación a objetos se acopla a la simulación de situaciones del mundo real.
La orientación a objetos intenta manejar la complejidad de los problemas del mundo real abstrayendo el conocimiento y encapsulándolo en objetos.

En este sentido surge un enfoque más abstracto que la programación procedimental donde es más importa el qué hace y no el cómo lo hace

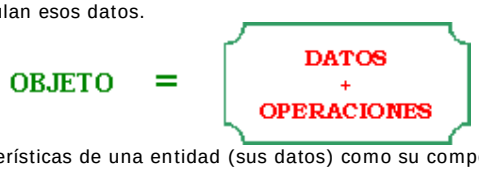
Paradigma de programación orientada por objetos

La Programación Orientada a Objetos (POO) se conoce como un nuevo paradigma de programación, ya que constituye un conjunto de teorías y métodos que representan un conocimiento, un medio de visualizar el mundo.
La orientación a objetos obliga a reconsiderar el pensamiento sobre la computación, sobre lo que significa desarrollar software y como se manipula la información dentro del computador.
La Programación Orientada a Objetos se define como el conjunto de disciplinas que desarrollan y modelan software que facilitan la construcción de sistemas complejos a partir de componentes.
Los conceptos y herramientas orientados a objetos son tecnologías que permiten que los problemas del mundo real sean expresados de modo fácil y natural, a través de:

1. **Modelización del Mundo.** La gente entiende el mundo en términos de objetos, por lo tanto, un programa escrito en términos de objetos debería ser más intuitivo y entendible que un programa estructurado de otra forma.
2. **Reuso.** Como modelos de software los objetos tienen alta cohesión porque los mismos encapsulan códigos y datos. Por lo tanto, un objeto puede ser tomado de un programa y utilizado en otros.
3. **Mantenibilidad.** Los objetos son más modulares. Los efectos de los cambios en programas son mejor localizados. Los objetos son más fáciles u económicos de implementar.
4. **Metodología de Software Unificada.** El análisis se hace en términos de objetos del mundo. Si el diseño, codificación y mantenimiento también son hechos en términos de objetos, el proceso de software completo será regido por un único concepto.

Un programa orientado objetos es una colección de objetos que interactúan entre sí.

Un **OBJETO** es todo lo que nos rodea, tanto elementos palpables como abstractos. Cumple un determinado rol dentro de los sistemas. Conoce como hacer su trabajo y recuerda su propia información.
De esta manera el objeto se constituye en la entidad central de la programación orientada a objetos.
Más formalmente se definen los **objetos** como tipos de datos que encapsulan con el mismo nombre estructuras de datos y las operaciones que manipulan esos datos.



Un objeto contiene tanto las características de una entidad (sus datos) como su comportamiento o funcionamiento (sus procedimientos).
Los objetos tienen **identidad**, ejecutan un cierto conjunto de **operaciones** y en determinado momento muestran un **estado**.

Identidad
Es el conjunto de atributos que definen y caracterizan/diferencian un objeto de otro.

Operaciones
Es el conjunto de procedimientos que permiten cambiar el estado del objeto, para procurar adaptarse al medio y/o interactuar con el medio.

Estado
Es el conjunto de valores de los atributos del objeto que definen como está el objeto en un momento dado.

Ejemplo:

Objeto : Vehículo

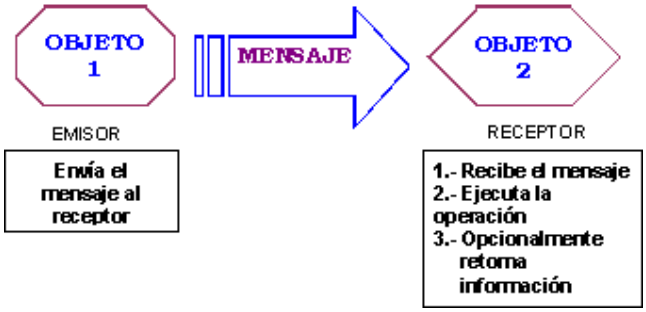
Identidad : Marca, Modelo, Tipo, Color, No. de Placa

Operaciones : Prender, Apagar, Avanzar, Detenerse, Acelerar, Frenar

Estado : Vehículo(Chevrolet, Esteem, Sedan, Verde, ABC123)

En general un objeto se compone de otros objetos. Los **atributos** son los objetos que componen el estado interno de un objeto. Establecen una jerarquía de inclusión entre los objetos. De esta manera un objeto está compuesto por uno o más atributos. Así el Objeto Vehículo se compone a su vez de los objetos: Motor, Chasis, Ruedas, etc., como se muestra en la siguiente figura:

Un objeto puede ser accedido sólo a través de su interfase pública mediante el envío de mensajes.
Un **mensaje** consiste de un Nombre y opcionalmente de uno o varios argumentos/parámetros.
Cuando un objeto manda un mensaje a otro, el emisor está requiriendo que el receptor haga una operación y opcionalmente que retorne determinada información.
Cuando el receptor recibe el mensaje, hace la operación requerida de la forma en que sabe hacerlo. El emisor no especifica cómo debe hacerse la operación.



El conjunto de mensajes a los que un objeto puede responder se conoce como el comportamiento del objeto. Cuando un objeto recibe un mensaje realiza la operación requerida mediante la ejecución de un método.
Un **método** es un algoritmo ejecutado es respuesta al envío de un mensaje. Siempre es parte de la representación privada del objeto. Nunca es parte de la interfaz pública.

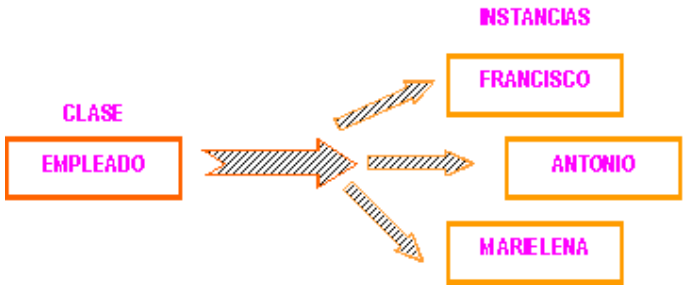
Las **PROPIEDADES BÁSICAS** que deben tener los objetos para que el modelo sea orientado a objetos son: la abstracción, el encapsulamiento, el polimorfismo y la herencia.

Abstracción

La abstracción es la propiedad que permite representar las características esenciales de un objeto, sin preocuparse de las restantes características.
Una abstracción se centra en la vista externa de un objeto, tal que propicie la separación del comportamiento esencial de un objeto, de su implementación.
Definir, entonces, una abstracción significa describir una entidad del mundo real y luego utilizar esta descripción en un programa.
La descripción abstracta de un grupo de objetos es una **Clase,** donde cada uno de los objetos descritos se diferencia por su estado específico y por la posibilidad de realizar una serie de operaciones.
Así, por ejemplo, un vehículo es un objeto que tienen un estado (azul, apagado, etc.) y sobre el cual se pueden realizar algunas operaciones (tales como, prender, acelerar, apagar, otras).
Características de las Clases:

- Permite crear objetos.
- Permite la definición de una serie de atributos
- Permite la definición de una serie de operaciones
- Permite la definición de restricciones de manera de conservar la consistencia de la Data.
- Permite almacenar información de los objetos que componen la clase. Sabe quienes la componen.
- Permite eliminar objetos.

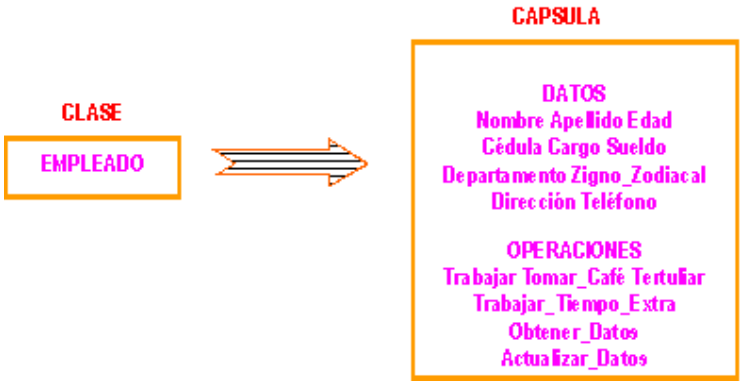
Los objetos que se comportan de la manera especificada por una clase se denominan **instancias** de esa clase.
Todos los objetos son instancias de alguna clase. Una vez que una instancia de una clase es creada, se comporta como todas las otras instancias de su clase.
Ejemplo:



Encapsulamiento

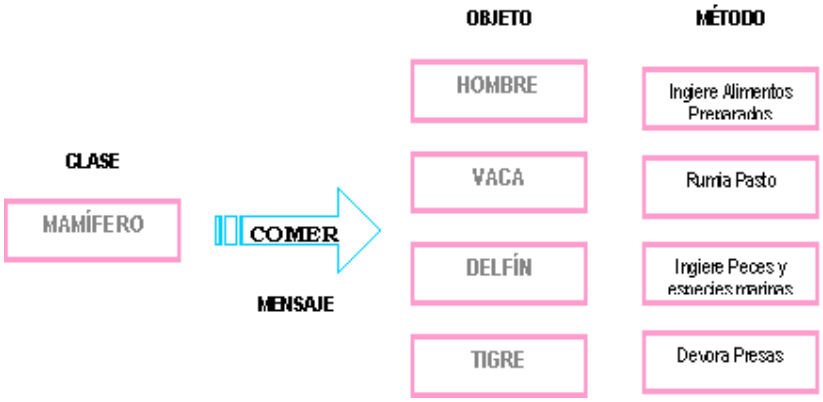
Es la propiedad que permite asegurar que el contenido de la información de un objeto está oculto al mundo exterior. El encapsulamiento permite el ocultamiento de la información y la división del programa en módulos.
Los módulos se implementan mediante clases, de forma que una clase representa la encapsulación de una abstracción. Esto significa que cada clase tiene dos partes: una interfaz y una implementación. La interfaz de una clase captura sólo su vista externa y la implementación contiene la representación de la abstracción, así como los mecanismos que realizan el comportamiento deseado.
El encapsulamiento provee las siguientes ventajas:

- Reducción de la cohesión entre distintos objetos, lo que implica la construcción de objetos independientes. Cambiar un objeto no afecta al resto, sólo al objeto en cuestión.
- Localización: la información relativa a un objeto está centrada en la definición de él mismo y no distribuida en otros objetos. Esto implica fácil modificación, fácil adaptación a los cambios, rápida ubicación de lo que hay que cambiar.
- Protección Integridad: sólo permite cambiar el estado por sí mismo, a través de sus operaciones propias. Esto asegura la integridad y la consistencia de la Data.



Polimorfismo

Es la propiedad que permite que dos o más clases de objetos respondan el mismo mensaje de diferente forma. De esta manera un objeto no necesita conocer a quién le está mandando el mensaje, sólo necesita conocer que varios tipos diferentes de objetos han sido definidos para responder ese mensaje en particular.
El polimorfismo permite reconocer y explotar las similitudes entre diferentes clases de objetos. Cuando se reconoce que varios tipos diferentes de objetos podrían responder el mismo mensaje, se visualiza más claramente la distinción entre el mensaje y el método.
Por ejemplo, cuando se describe la clase Mamíferos, se define la operación Comer. Los distintos mamíferos (hombre, vaca, delfín o tigre) efectivamente se alimentan de diferentes formas. El polimorfismo implica la posibilidad de tomar un objeto de un tipo (mamífero) e indicarle que ejecute una operación (comer). Esta acción se ejecutará de diferente forma según sea el objeto mamífero sobre el que se aplica.

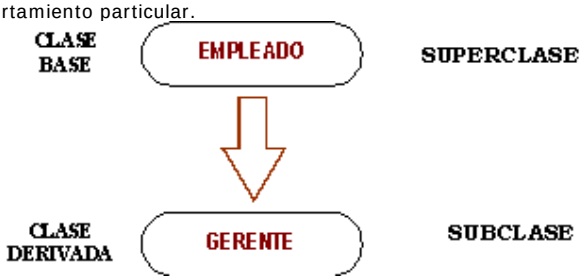


Herencia

El polimorfismo adquiere su máxima expresión en la derivación o extensión de clases, es decir, cuando se obtiene una clase a partir de una clase ya existente, mediante la propiedad de derivación de clases o herencia.
La **herencia** es un mecanismo que permite a un objeto heredar propiedades de otra clase de objetos. Permite a su vez a un objeto contener sus propios procedimientos y heredar los mismos de otros objetos.
Por ejemplo la clase **Gerente** hereda las propiedades de la clase **Empleado** e incluye nuevas propiedades.

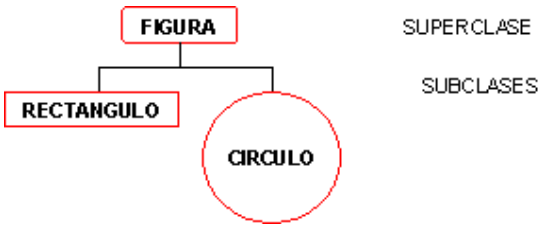
Un Gerente ES UN Empleado

La herencia permite diseñar y especificar sólo las diferencias de la clase más específica. Permite reusar tanto el comportamiento como el estado interno.
De la herencia surgen los conceptos de superclase y subclase.
Una Superclase es la clase de la cual heredan otras clases. Una Subclase es la clase que hereda su comportamiento de otra clase y define su comportamiento particular.



Ejemplo de Herencia

Definir con programación orientada a objetos la jerarquía de objetos que representa las figuras círculo y rectángulo.



Definición de las clases:

Figura:
Datos : Area, Perimetro, Forma
Métodos : CalculoArea, CalculoPerimetro, Dibujar

Circulo:
Datos : Radio
Métodos : CalculoArea, CalculoPerimetro, Dibujar

Rectangulo:
Datos : Altura, Base

Métodos : CalculoArea, CalculoPerimetro, Dibujar

Los métodos CalculoArea, CalculoPerímetro y Dibujar se declaran en la clase Figura pero se implementan en los objetos derivados, esto es dado que dichos procedimientos serán distintos según se trate de la figura círculo o rectángulo.

CalculoArea(Circulo) : Pi * $Radio^2$
CalculoPerimetro(Circulo) : 2*Pi*Radio
Dibujar(Circulo)

CalculoArea(Rectangulo) : Altura * Base
CalculoPerimetro(Rectangulo) : 2*Altura*Base
Dibujar(Rectangulo)

PROGRAMACIÓN ORIENTADA POR OBJETOS EN PHP

PHP 4.0 sólo maneja **Encapsulamiento** y **Herencia**. Aunque no permite declarar campos de datos ni funciones como privados. PHP 4.0 ofrece el concepto de clases, donde en una clase se agrupan campos de datos (variables) con campos de operaciones (funciones). PHP 4.0 no ofrece ni **Polimorfismo (sobrecarga, sobreescritura)** ni **Agregación**.

Clases en PHP

Una clase en PHP agrupa variables y funciones. La sintaxis de una clase es la siguiente:

```
<?php
class Empleado {
  var $CI;  // CI del empleado
  var $sueldo; // sueldo del empleado

  function calcularSueldo ($diasTrabajados, $bonoXDia) {
     $this->sueldo = $diasTrabajados * $bonoXdia;
  }

 }
?>
```

La sintaxis mostrada anteriormente incluye las palabras claves **class, var, this y el operador** ->. La palabra clave **class** se utiliza para definir la clase. La palabra clave **var** se indica para especificar los campos de datos o variables. La palabra clave **this** se emplea para acceder a los campos de la clase. Las palabras claves **var** y **this** son obligatorias, no se puede prescindir de ellas. En algunos lenguajes de programación el **this** se utiliza si existe confusión entre un parámetro (o una variable local de la función) con un campo de datos. El operador this es empleado para acceder a los campos de una clase. *NO* es posible separar la definición de una clase en varios archivos, o bloques PHP distintos.

```
<?php
class Empleado{
  var $CI;
  var $sueldo;

  function Empleado() {
     // código del constructor
     // el constructor es una función con el mismo
     // nombre de la clase

  }
}
?>
```

Las clases en PHP son tipos, por lo tanto una vez definida la clase se pueden trabajar con variables de esta clase. En terminología orientada por objetos cuando se utiliza una variable de una clase, se define como un objeto y debe ser **instanciada** empleando el operador **new**. Cuando PHP consigue el operador new, se ejecuta el código del constructor.

```
<?php
$emp = new Empleado;
// acceso a los campos de un objeto operador ->
$emp->calcularSueldo(30, 3500.56);

$emp2 = new Empleado;
// acceso a los campos de un objeto operador ->
$emp2->calcularSueldo(30, 6700.23);


?>
```


En el ejemplo anterior se crearon dos empleados, emp y emp2 y para ambos se invocó el método calcularSueldo.

Herencia en PHP

PHP soporta el concepto de herencia. Se utiliza la palabra clave **extends**. A continuación se muestra con un ejemplo la sintaxis de herencia.

```
<?php
class Gerente extends Empleado{
  var $bonoEjecutivo;
  function calcularSueldo($diasT, $bonoD) {
    Empleado::calcularSueldo($diasT,$bonoD); // heredado de empleado
                                  // Operador :: indica la
                                  // clase donde se ejecuta
                                  // el método
    // equivalentemente se pudo haber utilizado
    // parent::calcularSueldo($diasT,$bonoD);
    $this->sueldo += $bonoEjecutivo;
  }
}
?>
```

La clase extendida o derivada hereda todos los campos (variables y funciones de la clase base o padre). No es posible remover de la clase básica la definición de cualquier función o variable existente.
PHP no soporta herencia múltiple, por lo tanto una clase sólo puede extender una única clase.
En el ejemplo, se define un gerente como un empleado que adicionalmente maneja un bono ejecutivo.
Para invocar una operación de la clase padre o de cualquier padre en la rama de la herencia se emplea el operador **::** , en el caso de **parent::** sólo se sube una rama en la herencia.

PHP adicionalmente maneja los siguientes elementos de la programación orientada por objetos:
Constructores y Destructores (**__construct**, **__destruct**)
Visibilidad de los campos (**public**, **protected**, **private**)
Métodos estáticos (**static**)
Clase abstracta (**abstract**)
Interfaces (**interface**)
Clases selladas (**final**)
Como se observa, desde el punto de vista de lenguajes de programación PHP soporta toda la funcionalidad de un sistema totalmente orientado por objetos.

PHP HACIA FULL ORIENTACION POR OBJETOS

A partir de PHP 5, el modelo de objetos ha sido reescrito para permitir un mejor rendimiento y con más características. Este fue un cambio importante a partir de PHP 4. PHP 5 tiene un modelo de objetos completo.

Entre las características de PHP 5 están la inclusión de la visibilidad, las clases abstractas y clases y métodos finales, métodos mágicos adicionales, interfaces, clonación y tipos sugeridos.

PHP trata los objetos de la misma manera como referencias o manejadores, cada variable contiene una referencia de objeto en lugar de una copia de todo el objeto.

Propiedades

Las variables pertenecientes a clases se llaman "propiedades". También se les puede llamar usando otros términos como "atributos" o "campos". Éstas se definen usando una de las palabras clave **public, protected, o private**, seguido de una declaración normal de variable. Esta declaración puede incluir una inicialización, pero esta inicialización debe ser un valor constante, es decir, debe poder ser evaluada en tiempo de compilación y no debe depender de información en tiempo de ejecución para ser evaluada.

Con el fin de mantener la compatibilidad con PHP 4, PHP 5 continuará aceptando el uso de la palabra clave **var** en la declaración de propiedades en lugar de (o además de) public, protected, o private. Sin embargo, var ya no es necesaria.

Si se declara una propiedad utilizando var en lugar de public, protected, o private, PHP tratará dicha propiedad como si hubiera sido definida como public.

Dentro de los métodos de una clase, las propiedades no estáticas pueden ser accedidas utilizando -> (el operador de objeto): **$this->**propiedad (donde propiedad es el nombre de la propiedad). Las propiedades estáticas pueden ser accedidas utilizando **self::$propiedad**..

La pseudo-variable $this está disponible dentro de cualquier método de clase cuando éste es invocado dentro del contexto de un objeto. $this es una referencia al objeto que se invoca (usualmente el objeto al que pertenece el método, pero posiblemente sea otro objeto, si el método es llamado estáticamente desde el contexto de un objeto secundario).

```
Ejemplo #1 Declaración de propiedades
<?php
class ClaseSencilla
{
  // Declaraciones de propiedades inválidas:
  public $var1 = 'hola ' . 'mundo';
  public $var2 = <<<EOD
hola mundo
EOD;
  public $var3 = 1+2;
  public $var4 = self::myStaticMethod();
  public $var5 = $myVar;

  // Declaraciones de propiedades válidas:
  public $var6 = myConstant;
  public $var7 = array(true, false);

  // Esto se permite sólo en PHP 5.3.0 y posteriores.
  public $var8 = <<<'EOD'
hola mundo
EOD;
}
?>
```

Autocarga de clases

Muchos desarrolladores escribiendo aplicaciones orientadas a objetos crean un archivo fuente PHP para cada definición de clase. Una de las mayores molestias es hacer una larga lista de includes al comienzo de cada script (uno por cada clase).

En PHP 5 esto ya no es necesario. Se puede definir una función __autoload() que es automáticamente invocada en caso de que se esté intentando utilizar una clase/interfaz que todavía no haya sido definida. Al invocar a esta función el motor de scripting da una última oportunidad para cargar la clase antes que PHP falle con un error.

Este ejemplo intenta cargar las clases MiClase1 y MiClase2 desde los ficheros MiClase1.php y MiClase2.php respectivamente.

```
<?php

function __autoload($nombre_clase) {

   include $nombre_clase . '.php';

}



$obj  = new MiClase1();

$obj2 = new MiClase2();

?>
```

Constructores

```
void __construct ([ mixed $args [, $... ]] )
```

PHP 5 permite a los desarrolladores declarar métodos constructores para las clases. Aquellas que tengan un método constructor lo invocarán en cada nuevo objeto creado, lo que lo hace idóneo para cualquier inicialización que el objeto pueda necesitar antes de ser usado.

Nota: Constructores parent no son llamados implícitamente si la clase hija define un constructor. Para ejecutar un constructor parent, se requiere invocar a parent::__construct() desde el constructor child. Si el child no define un constructor, entonces se puede heredar de la clase padre como un método de clase normal (si no fue declarada como privada).

Ejemplo #1 Utilización de nuevos constructores unificados

```
<?php

class BaseClass {

  function __construct() {

     print "En el constructor BaseClass\n";

  }

}



class SubClass extends BaseClass {

  function __construct() {

     parent::__construct();

     print "En el constructor SubClass\n";

  }

}



class OtherSubClass extends BaseClass {

   // heredando el constructor BaseClass

}

// En el constructor BaseClass

$obj = new BaseClass();

// En el constructor BaseClass

// En el constructor SubClass

$obj = new SubClass();

// In BaseClass constructor

$obj = new OtherSubClass();

?>
```

Por motivos de compatibilidad, si PHP 5 no puede encontrar una función __construct() para una determinada clase y la clase no heredó uno de una clase padre, buscará el viejo estilo de la función constructora, mediante el nombre de la clase. Efectivamente, esto significa que en el único caso en el que se tendría compatibilidad es si la clase tiene un método llamado __construct() que fuese utilizado para diferentes propósitos.

Destructor

void __destruct (void)

PHP 5 introduce un concepto de destructor similar al de otros lenguajes orientados a objetos, tal como C++. El método destructor será llamado tan pronto como no hayan otras referencias a un objeto determinado, o en cualquier otra circunstancia de finalización.

Ejemplo #3 Ejemplo de Destructor

```
<?php

class MyDestructableClass {

  function __construct() {

     print "En el constructor\n";

     $this->name = "MyDestructableClass";

  }



  function __destruct() {

     print "Destruyendo " . $this->name . "\n";

  }

}



$obj = new MyDestructableClass();

?>
```

Como los constructores, los destructores padre no serán llamados implícitamente por el motor. Para ejecutar un destructor padre, se deberá llamar explícitamente a parent::__destruct() en el interior del destructor. También como los constructores, una clase hija puede heredar el destructor de los padres si no implementa uno propio.

El destructor será invocado aún si la ejecución del script es detenida usando exit(). Llamar a exit() en un destructor evitará que se ejecuten las rutinas restantes de finalización.

Visibilidad

La visibilidad de una propiedad o método se puede definir anteponiendo una de las palabras claves **public, protected o private** en la declaración. Miembros de clases declarados como public pueden ser accedidos de cualquier lado. Miembros declarados como protected, sólo de la clase misma, por herencia y clases parent. Aquellos definidos como private, únicamentede la clase que los definió.

Visibilidad de Propiedades

Las propiedades de clases deben ser definidas como public, private, o protected. Si se declaran usando var, serán definidas como public.

Ejemplo #1 Declaración de propiedades

```
<?php
/**
 * Definición de MyClass
 */
class MyClass
{
    public $public = 'Public';
    protected $protected = 'Protected';
    private $private = 'Private';

    function printHello()
    {
        echo $this->public;
        echo $this->protected;
        echo $this->private;
    }
}

$obj = new MyClass();
echo $obj->public; // Funciona
echo $obj->protected; // Error Fatal
echo $obj->private; // Error Fatal
$obj->printHello(); // Muestra Public, Protected y Private


/**
 * Definición de MyClass2
 */
class MyClass2 extends MyClass
{
    // Se puede redeclarar los métodos public y protected, pero no el private
    protected $protected = 'Protected2';

    function printHello()
    {
        echo $this->public;
        echo $this->protected;
        echo $this->private;
    }
}

$obj2 = new MyClass2();
echo $obj2->public; // Funciona
echo $obj2->private; // Undefined
echo $obj2->protected; // Error Fatal
$obj2->printHello(); // Muestra Public, Protected2, Undefined

?>
```

Nota: La forma de declaración de una variable de PHP 4 con la palabra clave var todavía es soportado (como un sinónimo de public) por razones de compatibilidad.

Visibilidad de Métodos

Los métodos de clases pueden ser definidos como public, private, o protected. Aquellos declarados sin ninguna palabra clave de visibilidad explícita serán definidos como public.

Ejemplo #2 Declaración de métodos

```
<?php
/**
 * Definición de MyClass
 */
class MyClass
{
    // Declaración de un constructor public
    public function __construct() { }

    // Declaración de un método public
    public function MyPublic() { }

    // Declaración de un método protected
    protected function MyProtected() { }
```

```
    // Declaración de un método private
    private function MyPrivate() { }

    // Esto es public
    function Foo()
    {
        $this->MyPublic();
        $this->MyProtected();
        $this->MyPrivate();
    }
}

$myclass = new MyClass;
$myclass->MyPublic(); // Funciona
$myclass->MyProtected(); // Error Fatal
$myclass->MyPrivate(); // Error Fatal
$myclass->Foo(); // Public, Protected y Private funcionan


/**
 * Definición de MyClass2
 */
class MyClass2 extends MyClass
{
    // Esto es public
    function Foo2()
    {
        $this->MyPublic();
        $this->MyProtected();
        $this->MyPrivate(); // Error Fatal
    }
}

$myclass2 = new MyClass2;
$myclass2->MyPublic(); // Funciona
$myclass2->Foo2(); // Public y Protected funcionan, pero Private no

class Bar
{
    public function test() {
        $this->testPrivate();
        $this->testPublic();
    }

    public function testPublic() {
        echo "Bar::testPublic\n";
    }

    private function testPrivate() {
        echo "Bar::testPrivate\n";
    }
}

class Foo extends Bar
{
    public function testPublic() {
        echo "Foo::testPublic\n";
    }

    private function testPrivate() {
        echo "Foo::testPrivate\n";
    }
}

$myFoo = new foo();
$myFoo->test(); // Bar::testPrivate
                // Foo::testPublic
?>
```

Visibilidad desde otros objetos

Los objetos del mismo tipo tendrán acceso a los miembros private y protected entre ellos aunque no sean de la misma instancia. Esto es porque los detalles específicos de implementación ya se conocen cuando se encuentra dentro de estos objetos.

Ejemplo #3 Accediendo a miembros private del mismo tipo de objeto

```
<?php
class Test
{
   private $foo;

   public function __construct($foo)
   {
      $this->foo = $foo;
   }

   private function bar()
   {
      echo 'Método private accedido.';
   }

   public function baz(Test $other)
   {
      // Se puede cambiar la propiedad private:
      $other->foo = 'hola';
      var_dump($other->foo);

      // También se puede invocar al método private:
      $other->bar();
   }
}

$test = new Test('test');

$test->baz(new Test('other'));
?>
```

El resultado del ejemplo sería:

```
string(5) "hola"
Método private accedido.
```

Herencia

La herencia es un principio de programación bien establecido y PHP hace uso de él en su modelado de objetos. Este principio afectará la manera en que muchas clases y objetos se relacionan unas con otras.

Por ejemplo, cuando se extiende una clase, la subclase hereda todos los métodos públicos y protegidos de la clase padre. A menos que una clase sobrescriba esos métodos, mantendrán su funcionalidad original.

Como se indico anteriormente se utiliza la palabra clave **extends** para utilizar herencia

Operador (::)

El Operador de Resolución de Ámbito o en términos simples, el doble dos-puntos, es un token que permite acceder a elementos estáticos, constantes, y sobrescribir propiedades o métodos de una clase.

Cuando se hace referencia a estos items desde el exterior de la definición de la clase, se utiliza el nombre de la clase.

A partir de PHP 5.3.0, es posible hacer referencia a una clase usando una variable. El valor de la variable no puede ser una palabra clave (por ej., self, parent y static).

Ejemplo #1 :: desde el exterior de la definición de la clase

```
<?php

class MyClass {

   const CONST_VALUE = 'Un valor constante';

}
```

```
$classname = 'MyClass';

echo $classname::CONST_VALUE; // A partir de PHP 5.3.0



echo MyClass::CONST_VALUE;

?>
```

Las tres palabras claves especiales **self**, **parent** y **static** son utilizadas para acceder a propiedades y métodos desde el interior de la definición de la clase.

Ejemplo #2 :: desde el interior de la definición de la clase

```
<?php

class OtherClass extends MyClass

{

   public static $my_static = 'variable estática';



   public static function doubleColon() {

      echo parent::CONST_VALUE . "\n";

      echo self::$my_static . "\n";

   }

}



$classname = 'OtherClass';

echo $classname::doubleColon(); // A partir de PHP 5.3.0



OtherClass::doubleColon();

?>
```

Cuando una clase extendida sobrescribe la definición parent de un método, PHP no invocará al método parent. Depende de la clase extendida el hecho de llamar o no al método parent. Esto también se aplica a definiciones de métodos Constructores y Destructores, Sobrecarga, y Mágicos.

Ejemplo #3 Invocando a un método parent

```
<?php

class MyClass
```

```
{
   protected function myFunc() {
      echo "MyClass::myFunc()\n";
   }
}

class OtherClass extends MyClass
{
   // Sobrescritura de definición parent
   public function myFunc()
   {
      // Pero todavía se puede llamar a la función parent
      parent::myFunc();
      echo "OtherClass::myFunc()\n";
   }
}

$class = new OtherClass();
$class->myFunc();
?>
```

Esto es útil para la definición y abstracción de la funcionalidad y permite la implementación de funcionalidad adicional en objetos similares sin la necesidad de reimplementar toda la funcionalidad compartida.

static

Declarar propiedades o métodos de clases como estáticos los hacen accesibles sin la necesidad de instanciar la clase. Una propiedad declarada como static no puede ser accedida con un objeto de clase instanciado (aunque un método estático sí lo puede hacer).

Por motivos de compatibilidad con PHP 4, si no se utiliza ninguna declaración de visibilidad, se tratará a las propiedades o métodos como si hubiesen sido definidos como public.

Debido a que los métodos estáticos se pueden invocar sin tener creada una instancia del objeto, la pseudo-variable **$this** no está disponible dentro de los métodos declarados como estáticos.

Las propiedades estáticas no pueden ser accedidas a través del objeto utilizando el **operador flecha (->).**

Como cualquier otra variable estática de PHP, las propiedades estáticas sólo pueden ser inicializadas utilizando un string literal o una constante; las expresiones no están permitidas. Por tanto, se puede inicializar una propiedad estática con enteros o arrays (por ejemplo), pero no se puede hacer con otra variable, con el valor de devolución de una función, o con un objeto.

A partir de PHP 5.3.0, es posible hacer referencia a una clase usando una variable. El valor de la variable no puede ser una palabra clave (p.ej., self, parent y static).

Ejemplo #1 Ejemplo de propiedad estática

```
<?php

class Foo

{

   public static $mi_static = 'foo';



   public function valorStatic() {

      return self::$mi_static;

   }

}

class Bar extends Foo

{

   public function fooStatic() {

      return parent::$mi_static;

   }

}

print Foo::$mi_static . "\n";

$foo = new Foo();

print $foo->valorStatic() . "\n";

print $foo->mi_static . "\n";      // "Propiedad" mi_static no definida

print $foo::$mi_static . "\n";

$nombreClase = 'Foo';

print $nombreClase::$mi_static . "\n"; // A partir de PHP 5.3.0



print Bar::$mi_static . "\n";

$bar = new Bar();
```

```
print $bar->fooStatic() . "\n";

?>
```

Ejemplo #2 Ejemplo de método estático

```
<?php

class Foo {

   public static function unMétodoEstático() {

      // ...

   }

}



Foo::unMétodoEstático();

$nombreClase = 'Foo';

$nombreClase::unMétodoEstático(); // A partir de PHP 5.3.0

?>
```

Abstracción de clases

PHP 5 introduce clases y métodos abstractos. Las clases definidas como abstract seguramente no son instanciadas y cualquier clase que contiene almenos un método abstracto debe ser definida como abstract. Los métodos definidos como abstractos simplemente declaran la estructura del método, pero no pueden definir la implementación.

Cuando se hereda de una clase abstracta, todos los métodos definidos como abstract en la definición de la clase parent deben ser redefinidos en la clase hija; adicionalmente, estos métodos deben ser definidos con la misma visibilidad (o con una menos restrictiva). Por ejemplo, si el método abstracto está definido como protected, la implementación de la función puede ser redefinida como protected o public, pero nunca como private. Por otra parte, las estructuras de los métodos tienen que coincidir; es decir, la implicación de tipos y el número de argumentos requeridos deben ser los mismos. Por ejemplo, si la clase derivada define un parámetro opcional, mientras que el método abstracto no, no habría conflicto con la estructura del método. Esto también aplica a los constructores de PHP 5.4. Antes de PHP 5.4 las estructuras del constructor podían ser diferentes.

Ejemplo #1 Ejemplo de clase abstracta

```
<?php

abstract class AbstractClass

{

   // Forzando la extensión de clase para definir este método

   abstract protected function getValue();

   abstract protected function prefixValue($prefix);



   // Método común
```

```
    public function printOut() {
        print $this->getValue() . "\n";
    }
}

class ConcreteClass1 extends AbstractClass
{
    protected function getValue() {
        return "ConcreteClass1";
    }

    public function prefixValue($prefix) {
        return "{$prefix}ConcreteClass1";
    }
}

class ConcreteClass2 extends AbstractClass
{
    public function getValue() {
        return "ConcreteClass2";
    }

    public function prefixValue($prefix) {
        return "{$prefix}ConcreteClass2";
    }
}

$class1 = new ConcreteClass1;
$class1->printOut();
echo $class1->prefixValue('FOO_') ."\n";
```

```
$class2 = new ConcreteClass2;

$class2->printOut();

echo $class2->prefixValue('FOO_') ."\n";

?>
```

El resultado del ejemplo sería:

```
ConcreteClass1

FOO_ConcreteClass1

ConcreteClass2

FOO_ConcreteClass2
```

Ejemplo #2 Ejemplo de clase abstracta

```
<?php

abstract class AbstractClass

{

    // El método abstracto sólo necesita definir los parámetros requeridos

    abstract protected function prefixName($name);


}


class ConcreteClass extends AbstractClass

{


    // La clase derivada puede definir parámetros opcionales que no estén en la estructura del prototipo

    public function prefixName($name, $separator = ".") {

        if ($name == "Pacman") {

            $prefix = "Mr";

        } elseif ($name == "Pacwoman") {

            $prefix = "Mrs";

        } else {

            $prefix = "";
```

```
        }

        return "{$prefix}{$separator} {$name}";

    }

}


$class = new ConcreteClass;

echo $class->prefixName("Pacman"), "\n";

echo $class->prefixName("Pacwoman"), "\n";

?>
```

El resultado del ejemplo sería:

```
Mr. Pacman

Mrs. Pacwoman
```

Códigos antiguos que no tengan clases o funciones definidas por el usuario llamadas 'abstract' deberían ejecutarse sin modificaciones.

Interfaces

Las interfaces permiten crear código con el cual se especifica qué métodos deben ser implementados por una clase, sin tener que definir cómo estos métodos son manipulados.

Las interfaces son definidas utilizando la palabra clave **interface**, de la misma forma que con clases estándar, pero sin métodos que tengan su contenido definido.

Todos los métodos declarados en una interfaz deben ser **public**, ya que ésta es la naturaleza de una interfaz.

implements

Para implementar una interfaz, se utiliza el operador implements. Todos los métodos en una interfaz deben ser implementados dentro de la clase; el no cumplir con esta regla resultará en un error fatal. Las clases pueden implementar más de una interfaz si se deseara, separándolas cada una por una coma.

Notas:

Una clase no puede implementar dos interfaces que compartan nombres de funciones, puesto que esto causaría ambigüedad.

Las interfaces se pueden extender al igual que las clases utilizando el operador extends.

La clase que implemente una interfaz debe utilizar exactamente las mismas estructuras de métodos que fueron definidos en la interfaz. De no cumplir con esta regla, se generará un error fatal.

Constantes

Es posible tener constantes dentro de las interfaces. Las constantes de interfaces funcionan como las constantes de clases excepto porque no pueden ser sobrescritas por una clase/interfaz que las herede.

Ejemplos

Ejemplo #1 Ejemplo de interfaz

```
<?php
// Declarar la interfaz 'iTemplate'
interface iTemplate
{
    public function setVariable($name, $var);
    public function getHtml($template);
}
// Implementar la interfaz
// Ésto funcionará
class Template implements iTemplate
{
    private $vars = array();
    public function setVariable($name, $var)
    {
        $this->vars[$name] = $var;
    }

    public function getHtml($template)
    {
        foreach($this->vars as $name => $value) {
            $template = str_replace('{' . $name . '}', $value, $template);
        }
        return $template;
    }
}
// Ésto no funcionará
// Error fatal: La Clase BadTemplate contiene un método abstracto
// y por lo tanto debe declararse como abstracta (iTemplate::getHtml)
```

```
class BadTemplate implements iTemplate
{
   private $vars = array();
   public function setVariable($name, $var)
   {
      $this->vars[$name] = $var;
   }
}
?>
```

Ejemplo #2 Interfaces extensibles

```
<?php
interface a
{
   public function foo();
}
interface b extends a
{
   public function baz(Baz $baz);
}
// Ésto sí funcionará
class c implements b
{
   public function foo()
   {
   }
   public function baz(Baz $baz)
   {
   }
}
// Ésto no funcionará y resultará en un error fatal
```

```
class d implements b
{
    public function foo()
    {
    }
    public function baz(Foo $foo)
    {
    }
}
?>
```

Ejemplo #3 Herencia múltiple de interfaces

```
<?php
interface a
{
    public function foo();
}
interface b
{
    public function bar();
}
interface c extends a, b
{
    public function baz();
}
class d implements c
{
    public function foo()
    {
    }
    public function bar()
```

```
    {

    }

    public function baz()

    {

    }

}

?>
```

Traits

Desde PHP 5.4.0, PHP implementa una metodología de reutilización de código llamada Traits.

Los traits (rasgos) son un mecanismo de reutilización de código en lenguajes de herencia simple, como PHP. El objetivo de un trait es el de reducir las limitaciones propias de la herencia simple permitiendo que los desarrolladores reutilicen a voluntad conjuntos de métodos sobre varias clases independientes y pertenecientes a clases jerárquicas distintas. La semántica a la hora combinar Traits y clases se define de tal manera que reduzca su complejidad y se eviten los problemas típicos asociados a la herencia múltiple y a los Mixins.

Un Trait es similar a una clase, pero con el único objetivo de agrupar funcionalidades muy específicas y de una manera coherente. No se puede instanciar directamente un Trait. Es por tanto un añadido a la herencia tradicional, y habilita la composición horizontal de comportamientos; es decir, permite combinar miembros de clases sin tener que usar herencia.

Ejemplo #1 Ejemplo de Trait

```
<?php

trait ezcReflectionReturnInfo {

    function getReturnType() { /*1*/ }

    function getReturnDescription() { /*2*/ }

}

class ezcReflectionMethod extends ReflectionMethod {

    use ezcReflectionReturnInfo;

    /* ... */

}

class ezcReflectionFunction extends ReflectionFunction {

    use ezcReflectionReturnInfo;

    /* ... */

}

?>
```

Los miembros heredados de una clase base se sobrescriben cuando se inserta otro miembro homónimo desde un Trait. De acuerdo con el orden de precedencia, los miembros de la clase actual sobrescriben los métodos del Trait, que a su vez sobrescribe los métodos heredados.

Ejemplo #2 Ejemplo de Orden de Precedencia

Se sobrescribe un miembro de la clase base con el método insertado en MiHolaMundo a partir del Trait DecirMundo. El comportamiento es el mismo para los métodos definidos en la clase MiHolaMundo. Según el orden de precedencia los métodos de la clase actual sobrescriben los métodos del Trait, a la vez que el Trait sobrescribe los métodos de la clase base.

```
<?php

class Base {

   public function decirHola() {

      echo '¡Hola ';

   }

}



trait DecirMundo {

   public function decirHola() {

      parent::decirHola();

      echo 'Mundo!';

   }

}



class MiHolaMundo extends Base {

   use DecirMundo;

}

$o = new MiHolaMundo();

$o->decirHola();

?>
```

El resultado del ejemplo sería:

¡Hola Mundo!

Ejemplo #3 Ejemplo de Orden de Precedencia #2

```
<?php

trait HolaMundo {
```

```
    public function decirHola() {
        echo '¡Hola Mundo!';
    }
}
class ElMundoNoEsSuficiente {
    use HolaMundo;
    public function decirHola() {
        echo '¡Hola Universo!';
    }
}
$o = new ElMundoNoEsSuficiente();
$o->decirHola();
?>
```

El resultado del ejemplo sería:

```
¡Hola Universo!
```

Se pueden insertar múltiples Traits en una clase, mediante una lista separada por comas en la sentencia **use**.

Ejemplo #4 Uso de Múltiples Traits

```
<?php
trait Hola {
    public function decirHola() {
        echo 'Hola ';
    }
}
trait Mundo {
    public function decirMundo() {
        echo 'Mundo';
    }
}
class MiHolaMundo {
    use Hola, Mundo;
```

```
    public function decirAdmiración() {
        echo '!';
    }
}
$o = new MiHolaMundo();
$o->decirHola();
$o->decirMundo();
$o->decirAdmiración();
?>
```

El resultado del ejemplo sería:

```
Hola Mundo!
```

Si dos Traits insertan un método con el mismo nombre, se produce un error fatal, siempre y cuando no se haya resuelto explicitamente el conflicto.

Para resolver los conflictos de nombres entre Traits en una misma clase, se debe usar el operador **insteadof** para elejir unívocamente uno de los métodos conflictivos.

Como esto sólo permite excluir métodos, se puede usar el operador as para permitir incluir uno de los métodos conflictivos bajo otro nombre.

Ejemplo #5 Resolución de Conflictos

En este ejemplo, Talker utiliza los traits A y B. Como A y B tienen métodos conflictos, se define el uso de la variante de smallTalk del trait B, y la variante de bigTalk del traita A.

Aliased_Talker hace uso del operador as para poder usar la implementación de bigTalk de B, bajo el alias adicional talk.

```
<?php
trait A {
    public function smallTalk() {
        echo 'a';
    }
    public function bigTalk() {
        echo 'A';
    }
}
trait B {
    public function smallTalk() {
```

```
        echo 'b';
    }
    public function bigTalk() {
        echo 'B';
    }
}
class Talker {
    use A, B {
        B::smallTalk insteadof A;
        A::bigTalk insteadof B;
    }
}
class Aliased_Talker {
    use A, B {
        B::smallTalk insteadof A;
        A::bigTalk insteadof B;
        B::bigTalk as talk;
    }
}
?>
```

Al usar el operador **as**, se puede también ajustar la visibilidad del método en la clase exhibida.

Ejemplo #6 Modificar la Visibilidad de un Método

```
<?php
trait HolaMundo {
    public function decirHola() {
        echo 'Hola Mundo!';
    }
}
// Cambiamos visibilidad de decirHola
class MiClase1 {
```

```
    use HolaMundo { decirHola as protected }

}

// Método alias con visibilidad cambiada

// La visibilidad de decirHola no cambia

class MiClase2 {

  use HolaMundo { hacerHolaMundo as private decirHola }

}

?>
```

Al igual que las clases, los Traits también pueden hacer uso de otros Traits. Al usar uno o más traits en la definición de un trait, éste puede componerse parcial o completamente de miembros de finidos en esos otros traits.

Ejemplo #7 Traits Compuestos de Traits

```
<?php

trait Hola {

    public function decirHola() {

        echo 'Hola ';

    }

}

trait Mundo {

    public function decirMundo() {

        echo 'Mundo!';

    }

}

trait HolaMundo {

    use Hola, Mundo;

}

class MiHolaMundo {

    use HolaMundo;

}

$o = new MiHolaMundo();

$o->decirHola();
```

```
$o->decirMundo();

?>
```

El resultado del ejemplo sería:

```
Hola Mundo!
```

Los traits soportan el uso de métodos abstractos para imponer requisitos a la clase a la que se exhiban.

Ejemplo #8 Expresar Resquisitos con Métodos Abstractos

```
<?php

trait Hola {

    public function decirHolaMundo() {

        echo 'Hola'.$this->obtenerMundo();

    }

    abstract public function obtenerMundo();

}

class MiHolaMundo {

    private $mundo;

    use Hola;

    public function obtenerMundo() {

        return $this->mundo;

    }

    public function asignarMundo($val) {

        $this->mundo = $val;

    }

}

?>
```

Los métodos de un trait pueden referenciar a variables estáticas, pero no puede ser definido en el trait. Los traits, sin embargo, pueden definir método estáticos a la clase a la que se exhiben.

Ejemplo #9 Variables estáticas

```
<?php

trait Contador {

    public function inc() {
```

```
        static $c = 0;
        $c = $c + 1;
        echo "$c\n";
    }
}
class C1 {
    use Contador;
}
class C2 {
    use Contador;
}
$o = new C1(); $o->inc(); // echo 1
$p = new C2(); $p->inc(); // echo 1
?>
```

Ejemplo #10 Métodos Estáticos

```
<?php
trait EjemploEstatico {
    public static function hacerAlgo() {
        return 'Hacer algo';
    }
}
class Ejemplo {
    use EjemploEstatico;
}
Ejemplo::hacerAlgo();
?>
```

Los traits también pueden definir propiedades.

Ejemplo #11 Definir Propiedades

```
<?php
trait PropiedadesTrait {
```

```
    public $x = 1;

}

class EjemploPropiedades {

    use PropiedadesTrait;

}

$ejemplo = new EjemploPropiedades;

$ejemplo->x;

?>
```

Si un trait define una propiedad una clase no puede definir una propiedad con el mismo nombre, si no se emitirá un error.Éste de tipo E_STRICT si la definición de la clase es compatible (misma visibilidad y valor inicial) o de otro modo un error fatal.

Ejemplo #12 Resolución de Conflictos

```
<?php

trait PropiedadesTrait {

    public $misma = true;

    public $diferente = false;

}

class EjemploPropiedades {

    use PropiedadesTrait;

    public $misma = true; // Estándares estrictos

    public $diferente = true; // Error fatal

}

?>
```

Sobrecarga

La sobrecarga en PHP ofrece los medios para "crear" dinámicamente propiedades y métodos. Estas entidades dinámicas se procesan por los métodos mágicos que se pueden establecer en una clase para diversas acciones.

Se invoca a los métodos de sobrecarga cuando se interactúa con propiedades o métodos que no se han declarado o que no son visibles en el ámbito activo.

Todos los métodos sobrecargados deben definirse como **public**.

Nota:

No se puede pasar ninguno de los parámetros de estos métodos mágicos por referencia.

OBSERVACION CONCEPTUAL: La interpretación de PHP de "overloading" es distinta de la mayoría de los lenguajes orientados a objetos. La sobrecarga tradicionalmente ofrece la capacidad de tener múltiples métodos con el mismo nombre, pero con un tipo o un número distinto de parámetros.

Sobrecarga de propiedades

public void __set (string $name , mixed $value)

public mixed __get (string $name)

public bool __isset (string $name)

public void __unset (string $name)

__set() se ejecuta al escribir datos sobre propiedades inaccesibles.

__get() se utiliza para consultar datos a partir de propiedades inaccesibles.

__isset() se lanza al llamar a isset() o a empty() sobre propiedades inaccesibles.

__unset() se invoca cuando se usa unset() sobre propiedades inaccesibles.

El parámetro $name es el nombre de la propiedad con la que se está interactuando. En el método __set() el parámetro $value especifica el valor que se debe asignar a la propiedad $name.

La sobrecarga de propiedades sólo funciona en contextos de objetos. Estos métodos mágicos no se lanzarán en contextos estáticos. Por esa razón, no se deben declarar como estáticos. Desde PHP 5.3.0, se emite un aviso si alguno de los métodos de sobrecarga es declarado como static.

Nota:

Debido a la forma en que PHP procesa el operador de asignación, el valor que devuelve __set() se ignora. Del mismo modo, nunca se llama a __get() al encadenar asignaciones como ésta:

$a = $obj->b = 8;

Nota:

No se puede utilizar una propiedad sobrecargada en ninguna construcción del lenguaje que no sea isset(). Esto significa que si se invoca a empty() en una propiedad sobrecargada, no se llamará al método de sobrecarga.

Para solventar esta limitación, se debe copiar la propiedad sobrecargada en una variable de ámbito local que empty() pueda manejar.

Ejemplo #1 Sobrecarga de propiedades mediante los métodos __get(), __set(), __isset() y __unset()

```
<?php

class PropertyTest

{

   /**  Localización de los datos sobrecargados.  */

   private $data = array();



   /**  La sobrecarga no se usa en propiedades declaradas.  */

   public $declared = 1;
```

```
/**  La sobre carga sólo funciona aquí al acceder desde fuera de la clase.  */
private $hidden = 2;
public function __set($name, $value)
{
   echo "Estableciendo '$name' a '$value'\n";
   $this->data[$name] = $value;
}
public function __get($name)
{
   echo "Consultando '$name'\n";
   if (array_key_exists($name, $this->data)) {
      return $this->data[$name];
   }
   $trace = debug_backtrace();
   trigger_error(
      'Propiedad indefinida mediante __get(): ' . $name .
      ' en ' . $trace[0]['file'] .
      ' en la línea ' . $trace[0]['line'],
      E_USER_NOTICE);
   return null;
}
/**  Desde PHP 5.1.0  */
public function __isset($name)
{
   echo "¿Está definido '$name'?\n";
   return isset($this->data[$name]);
}
/**  Desde PHP 5.1.0  */
public function __unset($name)
{
```

```
        echo "Eliminando '$name'\n";
        unset($this->data[$name]);
    }
    /**  No es un método mágico, esta aquí para completar el ejemplo.  */
    public function getHidden()
    {
        return $this->hidden;
    }
}
echo "<pre>\n";
$obj = new PropertyTest;
$obj->a = 1;
echo $obj->a . "\n\n";
var_dump(isset($obj->a));
unset($obj->a);
var_dump(isset($obj->a));
echo "\n";
echo $obj->declared . "\n\n";
echo "Vamos a probar con la propiedad privada que se llama 'hidden':\n";
echo "Las propiedades privadas pueden consultarse en la clase, por lo que no se usa __get()...\n";
echo $obj->getHidden() . "\n";
echo "Las propiedades privadas no son visibles fuera de la clase, por lo que se usa __get()...\n";
echo $obj->hidden . "\n";
?>
```

El resultado del ejemplo sería:

```
Estableciendo 'a' a '1'
Consultando 'a'
1
¿Está definido 'a'?
bool(true)
```

Eliminando 'a'

¿Está definido 'a'?

bool(false)

1

Vamos a probar con la propiedad privada que se llama 'hidden':

Las propiedades privadas pueden consultarse en la clase, por lo que no se usa __get()...

2

Las propiedades privadas no son visibles fuera de la clase, por lo que se usa __get()...

Consultando 'hidden'

Notice: Propiedad indefinida mediante __get(): hidden en <file> en la línea 69 in <file>en la línea 28

Sobrecarga de métodos

public mixed __call (string $name , array $arguments)

public static mixed __callStatic (string $name , array $arguments)

__call() es lanzado al invocar un método inaccesible en un contexto de objeto.

__callStatic() es lanzado al invocar un método inaccesible en un contexto estático.

El parámetro $name corresponde al nombre del método al que se está llamando. El parámetro $arguments es un array enumerado que contiene los parámetros que se han pasado al método $name.

Ejemplo #2 Sobrecarga de métodos mediante los métodos __call() and __callStatic()

```
<?php

class MethodTest

{

   public function __call($name, $arguments)

   {

      // Nota: el valor $name es sensible a mayúsculas.

      echo "Llamando al método de objeto '$name' "

          . implode(', ', $arguments). "\n";

   }

   /**  Desde PHP 5.3.0  */

   public static function __callStatic($name, $arguments)

   {
```

```
        // Nota: el valor $name es sensible a mayúsculas.
        echo "Llamando al método estático '$name' "
            . implode(', ', $arguments). "\n";
    }
}
$obj = new MethodTest;
$obj->runTest('en contexto de objeto');
MethodTest::runTest('en contexto estático');  // Desde PHP 5.3.0
?>
```

El resultado del ejemplo sería:

```
Llamando al método de objeto 'runTest' en contexto de objeto
Llamando al método estático 'runTest' en contexto estático.
```

Métodos mágicos

Los nombres de método __construct(), __destruct(), __call(), __callStatic(), __get(), __set(), __isset(), __unset(), __sleep(), __wakeup(), __toString(), __invoke(), __set_state() y __clone() son mágicos en las clases PHP. No se puede tener métodos con estos nombres en cualquiera de las clases a menos que desee la funcionalidad mágica asociada a estos.

PHP se reserva todos los nombres de los métodos que comienzan con __ como mágicos. Se recomienda no utilizar los nombres de métodos con __ en PHP a menos que desee alguna funcionalidad mágica documentada.

Final

PHP 5 introduce la nueva palabra clave **final**, que impide que las clases hijas sobrescriban un método, antecediendo su definición con la palabra final. Si la propia clase se define como final, entonces no se podrá heredar de ella.

Ejemplo #1 Ejemplo de métodos Final

```
<?php
class BaseClass {
  public function test() {
    echo "llamada a BaseClass::test()\n";
  }

  final public function moreTesting() {
    echo "llamada a BaseClass::moreTesting()\n";
  }
```

```
}

class ChildClass extends BaseClass {

  public function moreTesting() {

     echo "llamada a ChildClass::moreTesting()\n";

  }

}

// Devuelve un error Fatal: Cannot override final method BaseClass::moreTesting()

?>
```

Ejemplo #2 Ejemplo de clase Final

```
<?php

final class BaseClass {

  public function test() {

     echo "llamada a BaseClass::test()\n";

  }

  // Aquí no importa si se define una función como final o no

  final public function moreTesting() {

     echo "llamada a BaseClass::moreTesting()\n";

  }

}


class ChildClass extends BaseClass {

}

// Devuelve un error Fatal: Class ChildClass may not inherit from final class (BaseClass)

?>
```

Nota: Las propiedades no pueden declararse como final. Sólo pueden las clases y los métodos.

LECCION 11 ESPACIOS DE NOMBRES

En su definición más amplia los espacios de nombres son una manera de encapsular elementos. Se puede ver como un concepto abstracto en muchos aspectos. Por ejemplo, en cualquier sistema de operación los directorios sirven para agrupar archivos relacionados, y actúan como espacios de nombres para los archivos que hay en ellos. Como ejemplo, el archivo foo.txt puede existir en los directorios /home/greg y /home/otro, pero no pueden existir dos copias de foo.txt en el mismo directorio. Además, para acceder al archivo foo.txt fuera del directorio /home/greg se debe añadir el nombre del directorio delante del nombre del archivo usando el separador de directorios para obtener /home/greg/foo.txt. Este mismo principio se extiende a los espacios de nombres en el mundo de la programación.

En el mundo de PHP, los espacios de nombres están diseñados para solucionar dos problemas que los autores de bibliotecas y aplicaciones se encuentran cuando crean elementos de código reusable tales como clases o funciones:

El conflicto de nombres entre el código que se crea y las clases/funciones/constantes internas de PHP o las clases/funciones/constantes de terceros.

La capacidad de apodar (o abreviar) Nombres_Extra_Largos diseñado para aliviar el problema en primer lugar, mejorando la legibilidad del código fuente.

Los espacios de nombres en PHP proporcionan una manera para agrupar clases, interfaces, funciones y constantes relacionadas. Un ejemplo de la sintaxis de los espacios de nombres de PHP:

Ejemplo #1 Ejemplo de sintaxis de espacios de nombres

```
<?php

namespace mi\nombre; //

class MiClase {}

function mifunción() {}

const MICONSTANTE = 1;

$a = new MiClase;

$c = new \mi\nombre\MiClase;

$a = strlen('hola');

$d = namespace\MICONSTANTE;

$d = __NAMESPACE__ . '\MICONSTANTE';

echo constant($d); //

?>
```

Nota:

Los nombres de espacio de nombres PHP y php, y los nombres compuestos a partir de estos nombres (como PHP\Classes) están reservados para el uso del lenguaje interno y no deben ser utilizados en el código.

Definir espacios de nombres

Aunque cualquier código de PHP válido puede estar contenido dentro de un espacio de nombres, sólo son afectados por los espacios de nombres cuatro tipos de código: clases, interfaces, funciones y constantes.

Los espacios de nombres se declaran usando la palabra clave **namespace**. Un archivo que contiene un espacio de nombres debe declararlo al inicio del archivo, antes que cualquier otro código - con una excepción: la palabra clave declare.

Ejemplo #1 Declarar un único espacio de nombres

```
<?php

namespace MiProyecto;

const CONECTAR_OK = 1;

class Conexión { /* ... */ }

function conectar() { /* ... */  }

?>
```

La única construcción de código permitida antes de la declaración de un espacio de nombres es la sentencia declare, para declarar la codificación de un archivo fuente. Además, algo que no sea código de PHP no puede preceder a la declaración del espacio de nombres, incluyendo espacios en blanco extra:

Ejemplo #2 Declarar un único espacio de nombres

```
<html>

<?php

namespace MiProyecto; // error fatal - el espacio de nombres debe ser la primera sentencia del script

?>
```

Además, a diferencia de otras construcciones de PHP, se puede definir el mismo espacio de nombres en múltiples archivos, permitiendo la separación de contenido de espacios de nombres a través del sistema de archivos.

Declarar subespacios de nombres

Al igual que los directorios y archivos, los espacios de nombres de PHP también tienen la capacidad de especificar una jerarquía de nombres de espacios de nombres. Así, un nombre de un espacio de nombres se puede definir con subniveles:

Ejemplo #1 Declarar un único espacio de nombres con jerarquía

```
<?php
namespace MiProyecto\Sub\Nivel;

const CONECTAR_OK = 1;
class Conexión { /* ... */ }
function conectar() { /* ... */  }

?>
```

El ejemplo de arriba crea la constante MiProyecto\Sub\Nivel\CONECTAR_OK, la clase MiProyecto\Sub\Nivel\Conexión y la función MiProyecto\Sub\Nivel\conectar.

LECCION 12 EXCEPCIONES

PHP 5 tiene un modelo de excepciones similar al de otros lenguajes de programación. Una excepción puede ser lanzada (**thrown**), y atrapada ("**catch**ed") dentro de PHP. El código puede estar dentro de un bloque **try**, para facilitar la captura de excepciones potenciales. Cada bloque try debe tener al menos un bloque catch correspondiente. Se pueden usar múltiples bloques catch para atrapar diferentes clases de excepciones. La ejecución normal (cuando no es lanzada ninguna excepción dentro del bloque try, o cuando un bloque catch que coincide con la clase de la excepción lanzada no está presente) continuará después del último bloque catch definido en la sencuencia. Las excepciones pueden ser lanzadas (o relanzadas) dentro de un bloque catch.

Cuando una excepción es lanzada, el código siguiente a la declaración no será ejecutado, y PHP intentará encontrar el primer bloque catch conicidente. Si una excepción no es capturada, se emitirá un Error Fatal de PHP con un mensaje "Uncaught Exception ..." ("Excepción No Capturada"), a menos que se haya definido un gestor con **set_exception_handler**().

En PHP 5.5 y posteriores, se puede utilizar un bloque finally después de los bloques catch. El código de dentro del bloque finally siempre se ejecutará después de los bloques try y catch, independientemente de que se haya lanzado una excepción o no, y antes de que el flujo normal de ejecución continúe.

El objeto lanzado debe ser una instancia de la clase Exception o de una subclase de Exception. Intentar lanzar un objeto que no lo es resultará en un Error Fatal de PHP.

Nota:

Las funciones internas de PHP utilizan principalmente Información de Errores, sólo las extensiones Orientadas a objetos modernas utilizan excepciones. Sin embargo, los errores se pueden traducir a excepciones simplemente con ErrorException.

Ejemplo #1 Lanzar una Excepción

```
<?php
function inverso($x) {
    if (!$x) {
        throw new Exception('División por cero.');
    }
    else return 1/$x;
}

try {
    echo inverso(5) . "\n";
    echo inverso(0) . "\n";
} catch (Exception $e) {
    echo 'Excepción capturada: ',  $e->getMessage(), "\n";
}

// Continuar la ejecución
echo 'Hola Mundo';
?>
```

El resultado del ejemplo sería:

```
0.2
Excepción capturada: División por cero.
Hola Mundo
```

Ejemplo #2 Manejo de excepciones con un bloque finally

```
<?php
function inverse($x) {
    if (!$x) {
        throw new Exception('División por cero.');
    }
    else return 1/$x;
}

try {
    echo inverse(5) . "\n";
} catch (Exception $e) {
    echo 'Excepción capturada: ',  $e->getMessage(), "\n";
} finally {
    echo "Primer finally.\n";
}

try {
    echo inverse(0) . "\n";
} catch (Exception $e) {
    echo 'Excepción capturada: ',  $e->getMessage(), "\n";
} finally {
    echo "Segundo finally.\n";
}

// Continuar ejecución
echo 'Hola Mundo';
?>
```

El resultado del ejemplo sería:

```
0.2
Primer finally.
Excepción capturada: División por cero.
```

```
Segundo finally.
Hola Mundo

Ejemplo #3 Excepciones Anidadas
<?php

class MiExcepción extends Exception { }

class Prueba {
   public function probar() {
      try {
         try {
            throw new MiExcepción('foo!');
         } catch (MiExcepción $e) {
            /* relanzarla */
            throw $e;
         }
      } catch (Exception $e) {
         var_dump($e->getMessage());
      }
   }
}

$foo = new Prueba;
$foo->probar();

?>
El resultado del ejemplo sería:
string(4) "foo!"
```

LECCION 13 FUNCIONES PREDEFINIDAS DE PHP

Una de las ventajas de PHP sobre los otros lenguajes de programación es la completa y operacional librería de funciones predefinidas. En esta sección se mencionarán los grupos de funciones más importantes soportados en PHP.

Funciones matemáticas

abs -- Valor absoluto

acos -- Arco coseno

acosh – Arco Coseno hiperbólico

asin -- Arco seno

asinh – Arco Seno hiperbólico

atan -- Arco tangente

atanh – Arco Tangente hiperbólica

base_convert -- Convierte un número entre dos bases de numeración arbitrarias

BinDec -- binario a decimal

ceil – techo

cos -- coseno

cosh -- Coseno hiperbólico

DecBin -- decimal a binario

DecHex -- decimal a hexadecimal

DecOct -- decimal a octal

deg2rad -- Convierte el número en grados a su equivalente en radianes

exp – exponencial elevado a...

floor – piso

fmod – Operación mòdulo

getrandmax -- Muestra el mayor valor aleatorio posible

HexDec -- hexadecimal a decimal

is_nan -- Encuentra si un valor no es un número

log10 -- Logaritmo en base-10

log -- Logaritmo natural

max -- encuentra el valor mayor

min -- encuentra el valor menor

mt_rand -- genera un valor aleatorio (algoritmo mejorado)

mt_srand -- Introduce la semilla del generador de números aleatorios (algoritmo mejorado)

OctDec -- octal a decimal

pi -- devuelve el valor de pi

pow -- expresión exponencial

rad2deg -- Convierte el número en radianes a su equivalente en grados

rand -- genera un valor aleatorio

round -- Redondea un float.

sin -- seno

sinh -- Seno hiperbólico

sqrt -- Raíz cuadrada

srand -- introduce la semilla del generador de números aleatorios

tan -- tangente

tanh -- Tangente hiperbólica

Funciones del sistema de archivos

copy -- Copia un archivo

dirname -- Devuelve la parte del path correspondiente al directorio

disk_free_space -- Devuelve el espacio disponible en el directorio

disk_total_space -- Devuelve el tamaño total de un directorio

diskfreespace -- Devuelve el espacio disponible en un directorio

fclose -- Cierra el “file pointer” de un archivo abierto

feof -- Verifica si el “file pointer” de un archivo está al final del archivo (end-of-file)

fflush -- Vacía la salida hacia un archivo

fgetc -- Obtiene un caracter del archivo

fgetcsv -- Obtiene una línea del archivo en formato CSV (comma sepparated values)

fgets -- Obtiene una línea desde el “file pointer” de archivo

fgetss -- Obtiene una línea desde el “file pointer” de archivo y elimina las etiquetas HTML

file_exists -- Verifica si un archivo o directorio existe

file_get_contents -- Lee un archivo entero y lo almacena en una cadena

file_put_contents -- Escribir una cadena sobre un archivo

file -- Lee un archivo entero y lo almacena en una matriz

filesize -- Obtiene el tamaño del archivo

fopen -- Abre un archivo o URL

fputcsv -- Escribe un texto en un archivo con formato CSV

fputs -- Alias de **fwrite()**

fread -- Lectura de archivos en formato binario

fscanf -- Procesa la entrada desde un archivo de acuerdo a un formato

fseek -- Realiza una búsqueda sobre un “file pointer”

fstat -- Obtiene información sobre un archivo usando un “file pointer”

ftell -- Indica la posición de lectura/escritura del “file pointer”

ftruncate -- Trunca un archivo a la longitud dada

fwrite -- Escritura sobre archivos en formato binario

is_dir -- Indica si el nombre de archivo es un directorio

mkdir -- Crea un directorio

rewind -- Retrocede la posición de un “file pointer”

rmdir -- Elimina un directorio

touch -- Establece la hora de acceso y modificación de un archivo

unlink -- Elimina un archivo

Funciones de manejo de fecha y hora

checkdate -- valida una fecha u hora

date -- formatea a la fecha/hora local

getdate -- obtiene información de fecha y hora

gettimeofday -- obtiene la hora actual

idate -- formatea la fecha/hora como un integer

localtime -- Obtener la hora local

microtime -- devuelve el valor timestamp UNIX actual con microsegundos

mktime -- obtiene el timestamp UNIX de una fecha

time -- devuelve el timestamp UNIX actual

Funciones de manejo de correo

mail -- Enviar correo

bool **mail** (string para, string asunto, string mensaje [, string cabeceras_adicionales [, string parametros_adicionales]])

Funciones de manejo de bases de datos mysql

mysql_affected_rows -- Devuelve el número de filas afectadas de la última operación MySQL

mysql_close -- cierra el enlace con MySQL

mysql_connect -- Abre una conexión a un servidor MySQL

mysql_Crea_db -- Crea una base MySQL

mysql_db_query -- Envía una consulta MySQL al servidor

mysql_drop_db -- Borra una base de datos MySQL

mysql_fetch_array -- Extrae el registro de resultado como una matriz asociativa, una matriz numérica o ambas

mysql_fetch_assoc -- Recupera un registro de resultado como una matriz asociativa

mysql_fetch_objeto -- Extrae el registro de resultado como un objeto

mysql_fetch_row -- Devuelve un registro de resultado como matriz

mysql_list_fields -- Lista los campos del resultado de MySQL

mysql_list_tables -- Lista las tablas en una base de datos MySQL

mysql_num_fields -- devuelve el número de campos de un registro de MySQL

mysql_num_rows -- Devuelve el número de filas de un resultado

mysql_query -- Envía una consulta de MySQL

mysql_result -- Devuelve datos de un resultado

mysql_select_db -- Selecciona un base de datos MySQL

Funciones de manejo de XML

xml_get_current_byte_index -- obtiene el índice del byte actual para un analizador XML

xml_get_current_column_number -- Obtiene el número de columna actual para un analizador XML.

xml_get_current_line_number -- obtiene el número de línea actual de un analizador XML

xml_parse_into_struct – Realiza el parsing de datos XML en un arreglo

xml_parse -- comienza a analizar un documento XML

xml_parser_Crea_ns -- Crea un parser de XML con soporte para espacio de nombres

xml_parser_Crea -- crea un analizador de XML

xml_parser_free -- Libera un analizador XML

xml_set_character_data_handler -- Establece manejadores de datos de caracteres

xml_set_element_handler -- establece manejadores de los elementos principio y fin

xml_set_objeto -- Usa un analizador XML dentro de un objetoo

xml_set_processing_instruction_handler -- Establece el manejador de instrucciones de procesado (PI)

Funciones de manejo de DOM

DOMCharacterData->appendData() -- Agrega la cadena al final del nodo

DOMCharacterData->deleteData() -- Elimina una cadena del nodo nodo

DOMCharacterData->insertData() -- Inserta una cadena

DOMCharacterData->replaceData() -- Reemplaza una cadena

DOMCharacterData->substringData() -- Extrae una subacadena del nodo

DOMDocument->__construct() -- Crea un nuevo objeto DOMDocument

DOMDocument->CreaAttribute() -- Crea un nuevo atributo

DOMDocument->CreaCDATASection() -- Crea un nuevo nodo cdata

DOMDocument->CreaComment() -- Crea nuevo nodo comment

DOMDocument->CreaDocumentFragment() -- Crea un nuevo document fragment

DOMDocument->CreaElement() -- Crea un nuevo elemento

DOMDocument->CreaEntityReference() -- Crea un nuevo entity reference

DOMDocument->CreaProcessingInstruction() -- Crea un nuevo nodo PI

DOMDocument->CreaTextNodo() -- Crea nuevo nodo text

DOMDocument->getElementById() – Busca un elemento por el id

DOMDocument->getElementsByTagName() -- Busca los elementos dado un tag

DOMDocument->importNodo() -- Importa un nodo en un documento

DOMDocument->load() -- Carga XML de un archivo

DOMDocument->loadHTML() -- Carga HTML de una cadena

DOMDocument->loadHTMLFile() -- Carga HTML de un archivo

DOMDocument->loadXML() -- Carga XML de una cadena

DOMDocument->save() -- Almacena un árbol XML en un archivo

DOMDocument->saveHTML() -- Almacena un documento en una cadena en formato HTML

DOMDocument->saveHTMLFile() -- Almacena un documento en un archivo en formato HTML

DOMDocument->saveXML() -- Almacena un árbol XML en un archivo

DOMDocument->schemaValidate() -- Valida un documento basado en un esquema

DOMDocument->schemaValidateSource() -- Valida un documento basado en un esquema

DOMDocument->validate() -- Valida el documento basado en un DTD

DOMElement->getAttribute() – Retorna el valor de un atributo

DOMElement->getAttributeNodo() -- Returna el atributo nodo

DOMImplementation->CreaDocument() -- Crea un objeto DOMDocument

DOMImplementation->CreaDocumentType() -- Crea un objeto DOMDocumentType vacio

DOMNodo->appendChild() -- Agrega un nuevo hijo

DOMNodo->cloneNodo() -- Clona un nodo

DOMNodo->hasAttributes() -- Chequea si un nodo tiene atributos

DOMNodo->hasChildNodos() -- Chequea si un nodo tiene hijos

DOMNodo->insertBefore() -- Inserta un nuevo hijo antes de un nodo de reference

DOMNodo->removeChild() -- Remueve un hijo

DOMNodo->replaceChild() -- Reemplaza un hijo

Funciones de comercio electrónico

PHP permite procesar tarjetas de crédito y otras transacciones financieras utilizando Verisign Payment Services.

pfpro_cleanup – Finaliza la librería de Payflow Pro.

pfpro_init -- Finaliza la librería de Payflow Pro .

pfpro_process_raw – Procesa una transacción a bajo nivel con Payflow Pro

pfpro_process -- Procesa una transacción con Payflow Pro

LECCION 14 ARREGLOS

Un arreglo en PHP es un mapa ordenado. Un mapa es una estructura de datos donde se almacenan los datos en pares (clave, valor) . Este tipo de datos es implementado en varias formas: como un arreglo, una lista (vector), una tabla de hashing, un diccionario, una colección, una pila, una cola.

Un **arreglo** puede ser creado con la instrucción de lenguaje **array()**. Ésta toma un número de pares ***clave => valor*** separadas con coma (el operador => es obligatorio).

```
array( [clave =>] valor
    , ...
    )
// clave puede ser un integer o string
// valor puede ser cualquier valor
// notar que la calve es auxiliar
```

```
<?php
$matriz = array("cadena" => “hola", 1 => “número”);

echo $matriz["cadena"]; // “hola”
echo $matriz[1];    // “número”

?>
```

Una *clave* puede ser un *integer* o un **string**.
Un valor puede ser de cualquier tipo en PHP.
Se pueden combinar índices enteros y de cadena. Se pueden declarar arreglos de arreglos.

```
<?php
$matriz = array("1eraD" => array(1 => 3, 2 => 8, "l" => 42));

echo $matriz["1eraD"][1];    // 3
echo $matriz["1eraD "][2];  // 8
echo $matriz["1eraD "]["l"];  // 42
```



```
?>
```

Si no especifica una clave para un valor dado, PHP toma el valor máximo del índice utilizado, y la nueva clave será ese valor máximo incrementado en 1. Si especifica una clave que ya tiene un valor asignado, ése valor será sobrescrito.

```
<?php
// Esta matriz es idéntica a
array(0 => 33, 34, 35, "l" => 36);

// ...esta matriz
array(0 => 33, 1 => 34, 2 => 35, "l" => 36);
?>
```

También se puede utilizar el **operador [],** para agregar elementos en un arreglo.

```
<?php

// Esta matriz es idéntica a
$arreglo = array(0 => 33, 1 => 34, 2 => 35);


$arreglo[] = 33;
$arreglo[] = 34;
$arreglo[] = 35;
?>
```

Si desea remover un par clave/valor, se utiliza la instrucción **unset()**.

```
<?php

$matriz[] = 56;
$matriz[5] = 23;

unset($matriz[5]); // Elimina el elemento de la matriz

unset($matriz);    // Elimina la matriz completa
?>
```

Instrucciones de arreglos

```
<?php
// Crea una matriz con los números del 1 al 5.
$matriz = array(1, 2, 3, 4, 5);
print_r($matriz); // imprime la matriz

// Ahora imprime los ítems uno por uno:
foreach ($matriz as $i => $valor) { // en foeach va la clave y el
                                    // valor
  echo($matriz[$i]);
}


?>
```

Funciones de arreglos

array_change_key_case -- Devuelve una matriz con todas las claves de las cadenas convertidas a mayúsculas o minúsculas

array_chunk -- Divide una matriz en segmentos

array_combine -- Crea una nueva matriz, usando una matriz para las claves y otra para sus valores

array_count_values -- Cuenta todos los valores de una matriz

array_diff_assoc -- Comprueba las diferencias entre matrices teniendo en cuenta los índices

array_diff_key -- Calcula la diferencia de matrices usando las claves para la comparación

array_diff_uassoc -- Calcula la diferencia entre matrices con un chequeo adicional de índices, el cual es realizado por una llamada de retorno entregada por el usuario

array_diff -- Comprueba las diferencias entre matrices

array_fill -- Llena una matriz con valores

array_filter -- Filtra elementos de una matriz mediante una función "callback"

array_flip -- Intercambia los valores de una matriz con sus índices

array_intersect_key -- Calcula la intersección de matrices usando las claves para la comparación

array_intersect_uassoc -- Calcula la intersección de matrices con chequeo de índices adicional por una función de usuario

array_intersect_ukey -- Calcula la intersección de matrices usando una función de usuario para la comparación de los índices

array_key_exists -- Comprueba si el índice o clave dada existe en la matriz

array_keys -- Devuelve todas las claves de una matriz

array_map -- Aplica la llamada de retorno especificada a los elementos de las matrices dadas

array_merge_recursive -- Une dos o más matrices recursivamente

array_merge -- Combina dos o más matrices

array_multisort -- Ordena múltiples matrices, o matrices multi-dimensionales

array_pad -- Rellena una matriz con un valor hasta el tamaño especificado

array_pop -- Extrae el último elemento de la matriz

array_push -- Inserta uno o más elementos al final de la matriz

array_rand -- Selecciona una o más entradas aleatorias de una matriz

array_reduce -- Reduce iterativamente una matriz a un solo valor usando una función llamada de retorno

array_reverse -- Devuelve una matriz con los elementos en orden inverso

array_search -- Busca un valor determinado en una matriz y devuelve la clave correspondiente en caso de éxito

array_shift -- Extrae un elemento del comienzo de la matriz

array_slice -- Extrae una porción de la matriz

array_splice -- Suprime una porción de la matriz y la sustituye por otra cosa

array_sum -- Calcula la suma de los valores en una matriz

array_udiff_assoc -- Calcula la diferencia entre matrices con un chequeo de índices adicional, comparando los datos con una llamada de retorno

array_udiff_uassoc -- Calcula la diferencia entre matrices con un chequeo de índices adicional, comparando los datos y los índices con una llamada de retorno

array_udiff -- Calcula la diferencia entre matrices, usando una llamada de retorno para la comparación de datos

array_uintersect_assoc -- Calcula la intersección de matrices con chequeo adicional de índices, comparando los datos por una función del usuario

array_uintersect_uassoc -- Calcula la intersección de matrices con chequeo adicional de índices, compara los datos y los índices por una función del usuario

array_uintersect -- Calcula la intersección de matrices, compara los datos con una función del usuario

array_unique -- Remueve valores duplicados de una matriz

array_unshift -- Introduce uno o más elementos al principio de la matriz

array_values -- Devuelve todos los valores de una matriz

array_walk_recursive -- Aplicar una función de usuario recursivamente a cada miembro de una matriz

array_walk -- Aplica una función del usuario a cada elemento de una matriz.

array -- Crear una matriz

arsort -- Ordena una matriz en orden inverso y mantiene la asociación de índices

asort -- Ordena una matriz y mantiene la asociación de índices

compact -- Crea una matriz que contiene variables y sus valores

count -- Cuenta los elementos de una matriz o propiedades de un objeto

current -- Devuelve el elemento actual de una matriz

each -- Devuelve el siguiente par clave/valor de una matriz y avanza el apuntador

end -- Mueve el apuntador interno de una tabla al último elemento

extract -- Importa variables a la tabla de símbolos desde una matriz

in_array -- Revisa si un valor existe en una matriz

key -- Obtiene una clave de una matriz asociativa

krsort -- Ordena una matriz por clave en orden inverso

ksort -- Ordena una matriz por clave

list -- Asigna variables como si fueran una matriz

natcasesort -- Ordena una matriz usando un algoritmo de "orden natural" sin distinguir mayúsculas de minúsculas

natsort -- Ordena una matriz usando un algoritmo de "orden natural"

next -- Avanza el puntero interno de una matriz

pos -- Alias de **current()**

prev -- Rebobina el puntero interno de una matriz

range -- Crea una matriz que contiene un rango de elementos

reset -- Fija el puntero interno de una matriz a su primer elemento

rsort -- Ordena una matriz en orden inverso

shuffle -- Mezcla una matriz

sizeof -- Alias de **count()**

sort -- Ordena una matriz

uasort -- Ordena una matriz mediante una función de comparación definida por el usuario y mantiene la asociación de índices

uksort -- Ordena una matriz por claves mediante una función definida por el usuario

usort -- Ordena una matriz por sus valores usando una función de comparación definida por el usuario

LECCION 15 FECHAS

Las funciones para el manejo de fechas son parte del núcleo de PHP. No se necesita ninguna instalación para usar estas funciones. Estas funciones le permiten obtener la fecha y hora del servidor en donde están siendo ejecutados los scripts PHP. Estas funciones pueden ser utilizadas para formatear las fechas y horas de múltiples maneras.

checkdate -- valida una fecha u hora

int checkdate (int month, int day, int year)

Devuelve un valor verdadero si la fecha dada es válida; en caso contrario, devuelve un valor falso. Una fecha es válida si:

- el año está entre 0 y 32767, ambos incluidos
- el mes está entre 1 y 12, ambos incluidos
- el día está en el rango permitido para el mes dado. Se tienen en consideración los años bisiestos.

date -- formatea la fecha/hora local, de acuerdo con un formato

string **date** (string format [, int timestamp])

Retorna una cadena con la fecha formateada de acuerdo con la cadena de formateo pasada como parámetro. Esta función utiliza el valor de *timestamp* dado o la hora local actual en caso de no pasarse como parámetro.
En PHP existen las siguientes opciones de formateo:

- a - "am" o "pm"
- A - "AM" o "PM"
- d - día del mes, de "01" a "31"
- D - día de la semana, en texto, con tres letras; por ejemplo, "Fri"
- F - mes, en texto, completo; por ejemplo, "January"
- h - hora, de "01" a "12"
- H - hora, de "00" a "23"
- g - hora, sin ceros, de "1" a "12"
- G - hora, sin ceros; de "0" a "23"
- i - minutos; de "00" a "59"
- j - día del mes sin cero inicial; de "1" a "31"
- l ('L' minúscula) - día de la semana, en texto, completo; por ejemplo, "Friday"
- L - "1" o "0", según si el año es bisiesto o no

- m - mes; de "01" a "12"
- n - mes; de "1" a "12"
- M - mes, en texto, 3 letras; por ejemplo, "Jan"
- s - segundos; de "00" a "59"
- S - sufijo ordinal en inglés, en texto, 2 caracteres; por ejemplo, "th", "nd"
- t - número de días del mes dado; de "28" a "31"
- U - segundos desde el valor de 'epoch' (1 de enero de 1970)
- w - día de la semana, en número, de "0" (domingo) a "6" (sábado)
- Y - año, cuatro cifras; por ejemplo, "1999"
- y - año, dos cifras; por ejemplo, "99"
- z - día del año; de "0" a "365"
- Z - diferencia horaria en segundos (de "-43200" a "43200")

getdate -- obtiene información de fecha y hora

array getdate (int timestamp)

Devuelve un arreglo asociativo conteniendo la información de fecha del valor timestamp con los siguientes elementos:

- "seconds" - segundos
- "minutes" - minutos
- "hours" - horas
- "mday" - día del mes
- "wday" - día de la semana, en número
- "mon" - mes, en número
- "year" - año, en número
- "yday" - día del año, en número; por ejemplo, "299"
- "weekday" - día de la semana, en texto, completo; por ejemplo, "Friday"
- "month" - mes, en texto, completo; por ejemplo, "January"

gettimeofday -- obtiene la hora actual

array gettimeofday (void)

Devuelve un arreglo asociativo conteniendo los datos devueltos.

- "sec" - segundos
- "usec" - milisegundos
- "minuteswest" - minutos al oeste de Greenwich
- "dsttime" - tipo de corrección dst

localtime -- Obtener la hora local

array **localtime** ([int muestra_de_tiempo [, bool es_asociativo]])

Los nombres de las diferentes claves del vector asociativo se encuentran a continuación:

- "tm_sec" - segundos
- "tm_min" - minutos
- "tm_hour" - horas
- "tm_mday" - día del mes
- "tm_mon" - mes del año, empezando en 0 que es Enero
- "tm_year" - Años que hacen desde 1900
- "tm_wday" - Día de la semana
- "tm_yday" - Día del año
- "tm_isdst" - Si el cambio de hora para el ahorro energético tiene efecto o no

time -- devuelve el timestamp UNIX actual

int **time** ()

Devuelve la hora actual como número de segundos transcurridos desde las 00:00:00 del 1 de enero de 1970 GMT (Unix Epoch).

LECCION 16 MANEJO DE STRINGS

PHP soporta funciones para manejara cadenas de caracteres o "strings".

Las funciones aquí indicadas son las más genéricas, PHP ofrece también funciones más especializadas para manejar expresiones regulares y administrar URLs.

En secciones previas e este manual se indicaron las tres maneras de especificar strings, empleando comillas simples, dobles o la sintaxis HEREDOC.

El manejo de "strings" forma parte del núcleo de PHP.

AddCSlashes -- Marca una cadena con el carácter "\", estilo del lenguaje C

AddSlashes -- Marca una cadena con barras

chop -- Elimina espacios sobrantes al final

chr -- Devuelve un caracter específico

chunk_split -- Divide una cadena en trozos más pequeños

convert_uudecode -- Descifra una cadena codificada mediante uuencode

convert_uuencode -- Codifica, mediante uuencode, una cadena

count_chars -- Devuelve información sobre los caracteres usados en una cadena

crc32 -- Calcula el polinomio crc32 de una cadena

echo – Envía a la salida “Standard” a una o más cadenas

explode -- Divide una cadena por otra

fprintf -- Escribir una cadena con formato a un archivo

get_html_translation_table -- Devuelve la tabla de traducción utilizada por htmlspecialchars() y htmlentities()

html_entity_decode -- Convertir todas las entidades HTML a sus caracteres correspondientes

htmlentities -- Convierte todos los caracteres aplicables a entidades HTML

htmlspecialchars -- Convierte caracteres especiales a entidades HTML

implode -- Unir elementos de un arreglo mediante una cadena

join -- Une elementos de una tabla mediante una cadena

ltrim -- Elimina los espacios en blanco del principio de una cadena

money_format -- Da formato a un número como una cadena de moneda

nl2br -- Inserta saltos de linea HTML antes de cada salto de línea

number_format -- Dar formato a un número

ord -- Devuelve el valor ASCII de un caracter

parse_str -- Divide la cadena en variables (similar al tokenizer en Java)

print – Produce como salida una cadena

printf -- Imprimir una cadena con formato

rtrim -- Elimina espacios en blanco al final de la cadena.

similar_text -- Calcula la similitud entre dos cadenas

sprintf -- Devuelve una cadena con formato

sscanf – Divide en variables la información contenida en una cadena según un formato dado

str_ireplace -- Versión no sensible a mayúsculas y minúsculas de str_replace().

str_pad -- Rellena una cadena con otra hasta una longitud dada

str_repeat -- Repite una cadena

str_replace -- Sustituye todas las ocurrencias de una “subcadena” por otra “subcadena” en la cadena

str_shuffle -- Reordena aleatoriamente una cadena

str_split -- Convertir una cadena en un arreglo

str_word_count -- Devolver información sobre las palabras usadas en una cadena

strcasecmp -- Comparación de cadenas no sensible a mayúsculas y minúsculas y segura en modo binario

strchr -- Encuentra la primera aparición de un caracter

strcmp -- Comparación de cadenas con seguridad binaria

strcoll -- Comparación de cadenas basada en la localidad

strcspn -- Encuentra la longitud del elemento inicial que no coincide con una máscara

strip_tags -- Elimina los tags HTML y PHP de una cadena

stripcslashes -- Desmarca la cadena marcada con addcslashes()

stripos -- Encontrar la posición de la primera ocurrencia de una cadena, insensible a mayúsculas y minúsculas

stripslashes -- Desmarca la cadena marcada con addslashes()

stristr -- strstr() sin tener en cuenta mayúsculas o minúsculas

strlen -- Obtiene la longitud de la cadena

strnatcasecmp -- Comparación de cadenas no sensible a a mayúsculas y minúsculas usando un algoritmo de "orden natural"

strnatcmp -- Compara cadenas usando un algoritmo de "orden natural"

strncasecmp -- Comparación de los primeros n caracteres de cadenas, segura con material binario y no sensible a mayúsculas y minúsculas

strncmp -- Comparación de los n primeros caracteres de cadenas, con seguridad binaria

strpbrk -- Busca una cadena por cualquiera de los elementos de un conjunto de caracteres

strpos -- Encuentra la posición de la primera aparición de una cadena

strrchr -- Encuentra la última aparición de un caracter en una cadena

strrev -- Invierte una cadena

strripos -- Encontrar la posición de la última ocurrencia de una cadena en otra, no sensible a mayúsculas y minúsculas

strrpos -- Encuentra la posición de la última aparición de un caracter en una cadena

strspn -- Encuentra la longitud del segmento inicial que coincide con la máscara

strstr -- Encuentra la primera aparición de una cadena

strtok -- Divide una cadena en elementos

strtolower -- Convierte a minúsculas una cadena

strtoupper –- Convierte a mayúsculas una cadena

strtr -- Traduce algunos caracteres

substr_compare -- Comparación de 2 cadenas, opcionalmente no sensible a mayúsculas y minúsculas, a partir de un desplazamiento, y hasta un número límite de caracteres

substr_count -- Cuenta el número de apariciones de la subcadena

substr_replace -- Sustituye texto en una parte de una cadena

substr -- Devuelve parte de una cadena

trim -- Elimina espacios del principio y final de una cadena

ucfirst -- Convertir a mayúsculas el primer caracter de una cadena

ucwords -- Convertir en mayúsculas el primer caracter de cada palabra de una cadena

vfprintf – Escribe una cadena formateada en un “stream”

vprintf -- Imprimir una cadena con formato

vsprintf -- Devuelve una cadena con formato

wordwrap -- Divide una cadena en un número dado de caracteres usando un caracter que indica donde se va a dividir la cadena.

LECCION 17 ENTRADA Y SALIDA A TRAVÉS DE ARCHIVOS

PHP soporta funciones para manejar archivos.

Los archivos son manejados como en el lenguaje de programación “C”. En el caso de los archivos a bajo nivel (cadena de “streams”); el administrador del archivo es una variable del tipo entero (“file descriptor”). En el caso de los archivos a alto nivel (texto, strings); el administrador del archivo es una variable del tipo resource (“file pointer”).

Manejo a través de file descriptors

PHP incluye soporte para funciones de acceso directo a E/S. Estas funciones permiten realizar operaciones de E/S a bajo nivel nivel. Las funciones de acceso directo a E/S solo deberían emplearse cuando se requiere un control directo de un determinado dispositivo. En todos los demás casos, es más adecuado el empleo de las funciones estándar. **Nota:** Esta extensión no está disponible en plataformas Windows

dio_close

Cierra el archivo (recibe como parámetro un file descriptor)

void dio_close (int fd)

dio_open

Abre un archivo cuyo nombre indica el parámetro "nombre_archivo" con las opciones indicadas por "flags" y los permisos establecidos con "modo"

int dio_open (string nombre_archivo, int flags [, int modo])

Los flags, pueden ser:

O_RDONLY - abre el archivo para acceso de solo lectura.
O_WRONLY - abre el archivo para acceso de solo escritura.
O_RDWR - abre el archivo para acceso tanto de lectura como de escritura.
O_CREAT - crea el archivo si no existía previamente.
O_EXCL - si se indican de forma simultanea los valores O_CREAT y O_EXCL, la función **dio_open()** falla si el archivo existía previamente.
O_TRUNC - si el archivo existe y se permiten las operaciones de escritura sobre el, se elimina todo el contenido anterior del archivo y su tamaño se coloca en cero.
O_APPEND - las operaciones de escritura sobre el archivo escriben los datos al final del archivo.
O_NONBLOCK - el archivo se pone en modo no-bloqueante.

dio_read

Lee hasta *n* bytes del archivo cuyo descriptor es *fd* y los devuelve. Si no se le indica el valor de *n*, se leen hasta 1024 bytes.

string dio_read (int fd [, int n])

dio_seek

Cambia el apuntador de la posición actual en el archivo cuyo descriptor es *fd* a traves de los parámetros *pos* y *whence*

int dio_seek (int fd, int pos, int whence)

dio_stat

Obtiene la información sobre el archivo cuyo descriptor es *fd*

arreglo dio_stat (int fd)

dio_truncate

Trunca el tamaño del archivo cuyo descriptor es *fd* hasta un valor de *offset* bytes

bool dio_truncate (int fd, int offset)

dio_write

Escribe datos en el archivo cuyo descriptor es *fd*

int dio_write (int fd, string data [, int len])

Manejo a través de file pointers

fclose

Cierra el “filepointer” de un archivo abierto

int fclose (int fp)

feof

Verifica si el “filepointer” de un archivo está al final del archivo (end-of-file)

int feof (int fp)

fflush

Vacía la salida hacia un archivo

bool fflush (int fp)

fgetc

Obtiene un caracter del archivo a través del “filepointer”

string fgetc (int fp)

fgets

Obtiene una línea del archivo a través del “filepointer”

string fgets (resource fd [, int longitud])

fopen

Abre un archivo o URL y retorna el “filepointer”

int fopen (string nombre_archivo, string modo [, bool usar_ruta_inclusion])

La lista de modos posibles se muestra a continuación.

modo	Descripción
'r'	Apertura para sólo lectura; ubica el “file pointer” al comienzo del mismo. (READ ONLY)
'r+'	Apertura para lectura y escritura; ubica el “file pointer” al comienzo del mismo. (READ WRITE)
'w'	Apertura para sólo escritura; ubica el “file pointer” al comienzo de éste y lo trunca a una longitud de cero. Si el archivo no existe, intenta crearlo. (WRITE ONLY)

modo	Descripción
'w+'	Apertura para lectura y escritura; ubica el "file pointer" al comienzo de éste y lo trunca a una longitud cero. Si el archivo no existe, intenta crearlo. (READ WRITE)
'a'	Apertura para sólo escritura; ubica el "file pointer" al final del mismo. Si el archivo no existe, intenta crearlo. (APPEND)
'a+'	Apertura para lectura y escritura; ubica el "file pointer" al final del mismo. Si el archivo no existe, intenta crearlo. (APPEND + READ)
'x'	Creación y apertura para sólo escritura; ubica el "file pointer" al comienzo de éste. Si el archivo ya existe, la llamada a **fopen()** fallará devolviendo **FALSE** y generando un error de nivel **E_WARNING**. Si el archivo no existe, intenta crearlo. Esto es equivalente a especificar las banderas *O_EXCL|O_CREAT* en la llamada de sistema *open(2)* interna. Esta opción es soportada en PHP 4.3.2 y versiones posteriores, y sólo funciona con archivos locales.
'x+'	Creación y apertura para lectura y escritura; ubica el "file pointer" al comienzo de éste. Si el archivo ya existe, la llamada a **fopen()** fallará devolviendo **FALSE** y generando un error de nivel **E_WARNING**. Si el archivo no existe, intenta crearlo. Esto es equivalente a especificar las banderas *O_EXCL|O_CREAT* en la llamada de sistema *open(2)* interna. Esta opción es soportada en PHP 4.3.2 y versiones posteriores, y sólo funciona con archivos locales.

fputs

Alias de fwrite()

fscanf

Procesa la entrada desde un archivo de acuerdo a un formato, a través del "filepointer"

mixed fscanf (int fp, string formato [, mixed &v1...])

El formato indica con el símbolo porcentaje (%) el tipo de datos de las varialbles, y las variables deben ser pasadas por referencia (la sintaxis es similar a lenguaje C)

Ejemplo

fscanf(fp,"%s%d%lf",&varCadena,&varEntero,&varDouble);

Lee del archivo una cadena, un entero y un double)

fseek

Realiza una búsqueda en un archivo empleando el "filepointer"

int fseek (int fp, int desplazamiento [, int desde])

desplazamiento se especifica en bytes

desde, se define con las constantes

SEEK_SET - Define la posición igual a *desplazamiento* bytes.
SEEK_CUR - Define la posición como la posición actual más *desplazamiento*.
SEEK_END - Define la posición como el final-de-archivo más *desplazamiento*. (Para moverse a una posición anterior al final-de-archivo, es necesario pasar un valor negativo en *desplazamiento*.)
Si no se especifica *desde*, se asume que sea SEEK_SET.

fwrite

Escribe en un archivo empleando el “filepointer”

int **fwrite** (int fp, string cadena [, int longitud])

LECCION 18 MANEJO DE IMÁGENES

PHP ofrece funciones para la manipulación de imágenes. Las funciones básicas de PHP permiten obtener el tamaño de imágenes JPEG, GIF, PNG, SWF, TIFF y JPEG2000. Para crear y manipular imágenes es necesario instalar la librería **GD** (disponible en http://www.boutell.com/gd/).

Los formatos de imágenes a manipular dependen de la versión de GD instalada y de cualquier otra librería requerida por GD para acceder a estos formatos. Las versiones de GD anteriores a la GD-1.6 soportan imágenes en formato gif y no soportan png, en cambio las versiones superiores a la GD-1.6 soportan el formato png y no el gif.

Para habilitar las funciones de lectura y escritura de imágenes en formato jpeg, se debe instalar jpeg-6b (disponible en ftp://ftp.uu.net/graphics/jpeg/).

Para añadir el soporte de fuentes Type 1, puede instalar t1lib (disponible en ftp://sunsite.unc.edu/pub/Linux/libs/graphics/).

A continuación se indica un listado de las funciones para el manejo de imágenes.

gd_info

Obtener información de la librería de GD instalada

getimagesize

Obtener el tamaño de una imagen

image_type_to_extension

Obtener la extensión del archive para un tipo de imagen

image2wbmp

Desplegar la imagen en el navegador o almacenarla en un archive

imagealphablending

Actualizar el modo de “blending” para una imagen

ImageArc

Dibujar un arco

ImageChar

Dibujar un carácter horizontalmente

ImageCharUp

Dibujar un carácter verticalmente

ImageColorAllocate

Reservar un color para una imagen

ImageColorAt

Obtener el índice del color de un pixel

ImageColorClosest

Obtener el índice del color más cercano al color especificado

ImageColorExact

Obtener el índice del color especificado

ImageColorSet

Establecer el color para el índice de la paleta especificado

ImageColorsForIndex

Obtener los colores de un índice

ImageColorTransparent

Definir un color como transparente

imagecopy

Copiar parte de una imagen

imagecopymerge

Copiar y unir parte de una imagen

ImageCopyResized

Copiar y redimensionar parte de una imagen

ImageCreate

Crear una nueva imagen

imagecreatefromgd2

Crear una nueva imagen a partir de un archivo GD2 o un URL

imagecreatefromgd2part

Crear una nueva imagen a partir de parte de un archivo GD2 o un URL

imagecreatefromgd

Crear una nueva imagen a partir de un archivo GD o un URL

imagecreatefromgif

Crear una nueva imagen a partir de un archivo gif o un URL

imagecreatefromjpeg

Crear una nueva imagen a partir de un archivo jpeg URL

imagecreatefrompng

Crear una nueva imagen a partir de un archivo png o un URL

imagecreatefromwbmp

Crear una nueva imagen a partir de un archivo bmp o un URL

ImageDashedLine

Dibujar una línea discontinua

ImageDestroy

Destruir una imagen

imageellipse

Dibujar una elipse

ImageFill

Rellenar una imagen

imagefilledarc

Dibujar un arco relleno

imagefilledellipse

Dibujar una elipse rellena

ImageFilledPolygon

Dibujar un polígono relleno

ImageFilledRectangle

Dibuja un rectángulo relleno

imagefilter

Aplicar un filtro a una imagen

ImageFontHeight

Retornar la altura de una fuente

ImageFontWidth

Retornar la anchura de una fuente

imagegif

Desplegar en un navegador una imagen gif o almacenarla en un archivo

imagejpeg

Desplegar en un navegador una imagen jpeg o almacenarla en un archivo

ImageLine

Dibujar una línea

imageloadfont

Cargar una fuente nueva

imagepng

Desplegar en un navegador una imagen png o almacenarla en un archivo

ImagePolygon

Dibujar un polígono

ImageRectangle

Dibujar un rectángulo

imagerotate

Rotar una imagen dado un ángulo

imagesetbrush

Determinar el pincel para dibujar líneas

ImageSetPixel

Dibujar un pixel

imagesetstyle

Determinar el pincel para dibujar líneas

imagesetthickness

Determinar el grosor para dibujar líneas

ImageString

Dibujar una cadena de texto horizontalmente

ImageStringUp

Dibujar una cadena de texto verticalmente

ImageSX

Obtener el ancho de la imagen

ImageSY

Obtener la altura de la imagen

LECCION 19 CONEXIÓN A BASES DE DATOS

PHP ofrece una amplia gama de funciones para realizar la integración con Bases de Datos Relacionales. Soporta un manejador de bases de datos nativos, también contiene integración via ODBC y un soporte directo con el manejador de base de datos “open source” mysql.

Funciones DMB

Estas funciones permiten operar con registros almacenados en una base de datos estilo dbm. Este tipo de base de datos almacena pares (clave , valor) en lugar de los registros soportados por las bases de datos relacionales.

Para usar estas funciones es necesario compilar PHP con soporte para una base de datos base.

A continuación se muestra un listado de las funciones de DBM

dbmclose

Cerrar una base de datos dbm

dbmdelete

Eliminar un valor de una clave de una base de datos dbm

dbmexists

Determinar si existe un valor para una clave dada en la base de datos dbm

dbmfetch

Obtener un valor para una clave desde la base de datos dbm

dbmfirstkey

Obtener la primera clave de una base de datos dbm

dbminsert

Insertar un valor para una clave en la base de datos dbm

dbmnextkey

Obtener la siguiente clave de una base de datos dbm

dbmopen

Abrir una base de datos DBM

dbmreplace

Sustituir el valor de una clave en la base de datos dbm

Funciones ODBC

Estas funciones ofrecen en PHP el soporte de Bases de Datos ODBC, el cual es un estándar soportado por la mayoría de los manejadores de Bases de Datos comerciales.

odbc_autocommit

Indicar si se realiza o no “auto commit

odbc_close_all

Cerrar todas las conexiones ODBC

odbc_close

Cerrar una conexion ODBC

odbc_columns

Listar los nombres de las columnas de una tabla

odbc_commit

Comando commit

odbc_connect

Establecer conexión con la Base de Datos

odbc_cursor

Obtener el nombre del cursor

odbc_data_source

Obtener información sobre la conexión actual

odbc_do

Similar a **odbc_exec()**

odbc_error

Obtener el código del último error producido

odbc_errormsg

Obtener el mensaje del último error producido

odbc_exec

Preparar o ejecutar un comando SQL

odbc_execute

Ejecuta un comando SQL preparado

odbc_fetch_array

Realizar fetch de un registro como un arreglo asociativo

odbc_fetch_into

Buscar un registro de resultados dentro de un vector

odbc_fetch_object

Realizar fetch de un registro como un objeto

odbc_fetch_row

Buscar un registro

odbc_field_len

Obtener la longitud de un campo

odbc_field_name

Obtener el nombre de campo

odbc_field_num

Obtener el numero de campo

odbc_field_precision

Similar a **odbc_field_len()**

odbc_field_type

Tipo de datos de un campo

odbc_foreignkeys

Obtener la lista de claves foráneas de una tabla

odbc_free_result

Liberar los recursos asociados a un resultado

odbc_next_result

Determinar si existen múltiples resultados

odbc_num_fields

Obtener número de campos de un resultado

odbc_num_rows

Obtener número de registros de un resultado

odbc_pconnect

Crear conexión permanente son la base de datos

odbc_prepare

Preparar un comando SQL para su ejecución

odbc_primarykeys

Obtener las claves primarias de una tabla

odbc_procedures

Obtener la lista de procedimientos almacenados

odbc_result_all

Mostrar los resultados como una tabla HTML

odbc_result

Obtener información de un campo

odbc_rollback

Comando rollback

odbc_statistics

Obtener estadísticas de una tabla

odbc_tableprivileges

Listar las tablas y los privilegios asociados con una tabla

odbc_tables

Listar las tablas

Funciones mysql

Estas funciones ofrecen en PHP el soporte para el manejo de Bases de Datos mysql.
En PHP 4, el soporte para MySQL se encuentra habilitado por defecto. En Windows no existe DLL, simplemente es parte de PHP 4. Sin embargo en PHP 5 mysql no se encuentra disponible por defecto.

mysql_affected_rows

Obtener el número de filas afectadas de la última operación MySQL

mysql_change_user

Cambiar el usuario conectado en la conexión activa

mysql_close

Cerrar el enlace con MySQL

mysql_connect

Abrir una conexión a un servidor MySQL

mysql_create_db

Crear una base de datos MySQL

mysql_db_query

Enviar un comando MySQL al servidor

mysql_drop_db

Eliminar una base de datos MySQL

mysql_errno

Obtener el número del mensaje de error de la última operación MySQL

mysql_error

Obtener el texto del mensaje de error de la última operación MySQL

mysql_fetch_array

Obtener un registro de resultado como un arreglo asociativo, un arreglo numérico o ambas

mysql_fetch_assoc

Obtener un registro de resultado como un arreglo asociativo

mysql_fetch_field

Obtener la información de una columna como un objeto.

mysql_fetch_lengths

Obtener la longitud de cada resultado

mysql_fetch_object

Obtener un registro de resultado como un objeto

mysql_fetch_row

Obtener un registro de resultado como un arreglo

mysql_info

Obtener información sobre la consulta más reciente

mysql_list_dbs

Listar las bases de datos disponibles en el servidor MySQL

mysql_list_fields

Listar los campos del resultado de MySQL

mysql_list_tables

Listar las tablas en una base de datos MySQL

mysql_num_fields

Obtener el número de campos de un resultado

mysql_num_rows

Obtener el número de filas de un resultado

mysql_pconnect

Abrir una conexión persistente al servidor MySQL

mysql_query

Enviar una consulta de MySQL

mysql_result

Obtener datos de un resultado

mysql_select_db

Seleccionar un base de datos MySQL

Otros manejadores de Bases de Datos soportados

Adicionalmente a los mencionados en esta sección. PHP soporta los siguientes manejadores de bases de datos :

- Oracle
- SQL Server
- Dbase
- Informix
- Ingres II
- PostgreSQL
- Sybase

LECCION 20 MANEJO DE CORREO

Las funciones de correo están más orientadas a los servidores **Linux con apache**. Para que las funciones de Correo se encuentren disponibles, PHP debe tener acceso al binario ***sendmail*** en su sistema durante tiempo de compilación. Si usa otro programa de correo, como qmail o postfix, asegúrese de usar Los "envelopes" sendmail apropiadas que vienen con tales sistemas de correo.

Por lo PHP buscará sendmail primero en su *PATH*, y luego en los siguientes sitios: */usr/bin:/usr/sbin:/usr/etc:/etc:/usr/ucblib:/usr/lib*. Es bastante recomendable contar con el programa sendmail disponible en su *PATH*. Asimismo, el usuario que compile PHP debe tener permiso para acceder al binario sendmail.

Se puede enviar correo desde **html** configurando el texto de **mailto:** (dinámicamente).

mail

Enviar correo

Disponible en (PHP 3, PHP 4 , PHP 5)

bool **mail** (string para, string asunto, string mensaje [, string cabeceras_adicionales [, string parametros_adicionales]])

mail() envía automáticamente por correo el mensaje especificado en *mensaje* al destinatario especificado en *para*. Es posible especificar múltiples destinatarios colocando una coma entre cada dirección en la cadena *para*. Es posible enviar correo electrónico con archivos adjuntos y tipos especiales usando esta función. Esto se consigue mediante el uso de codificación MIME.

LECCION 21 SEGURIDAD EN PHP

Introducción

PHP es un poderoso lenguaje de programación interpretado en cualquiera de sus tres facetas: incluido como parte de un servidor web, en forma de módulo o ejecutado como un binario CGI independiente. PHP ofrece posibilidades de acceder a archivos, ejecutar comandos y abrir conexiones de red en el servidor. Estas facilidades de PHP se pueden tornar en un problema de seguridad debido a la posibilidad de realizar operaciones directamente en el servidor. PHP está diseñado específicamente para ser un lenguaje más seguro para el desarrollo programas CGI que Perl o C. Para tornar PHP seguro, se debe aplicar la selección correcta de parámetros de configuración en tiempo de compilación y ejecución y deben seguirse prácticas correctas de programación. PHP puede ser tan seguro y al mismo tiempo tan flexible como se desee, el problema es que la seguridad dependen del programador de PHP.

Debido a la versatilidad de desarrollo en PHP, existen múltiples opciones de configuración diseñadas para controlar su comportamiento. Un amplio rango de opciones garantiza la utilización de PHP en varios tipos de aplicaciones con muchos propósitos distintos; pero el programador debe controlar la generación de un código y un ambiente de producción seguro.

El nivel de flexibilidad en la configuración de PHP se compara quizás solo con su flexibilidad de desarrollo. PHP puede ser utilizado para escribir aplicaciones completas de servidor, con todo el poder de acceder a las funciones del servidor,

o puede ser empleado para inclusiones simples del lado del servidor con muy poco riesgo controlando totalmente todas las operaciones.

Consideraciones de seguridad

Desarrollar un sistema completamente seguro no es una tarea fácil. La política más empleada es lograr el balance adecuado entre riesgo y funcionalidad. Si cada variable enviada por un usuario utilizara un mecanismo de validación muy sofisticado (revisar la retina o un análisis dactilar) , el sistema sería realmente seguro pero muy inflexible para su utilización. Por ejemplo, podría tomar media hora llenar los datos de un formulario razonablemente complejo podría incentivando a los usuarios a buscar métodos para evadir los mecanismos de seguridad.

Mecanismos de seguridad

La **seguridad del código fuente** en PHP no puede ser vulnerada directamente desde los clientes, debido a que el cliente sólo puede ver el código HTML enviada por el servidor. Las primeras versiones de asp tenían un "bug# que permitía obtener el código fuente de una página asp. Al tipear desde un cliente http://www.institutogala.com/codigo.asp::$DATA; se obtenía en el cliente el código fuente del script.

La **encriptación** es un mecanismo utilizado frecuentemente para proteger el contenido de información transmitida por Internet. En PHP se utiliza el módulo de encriptación **mcrypt** el cual consta de las siguientes funciones básicas (las cuatro más importantes) :

mcrypt_cbc

Encripta/desencripta datos en modo CBC

mcrypt_cfb

Encripta/desencripta datos en modo CFB

mcrypt_ecb

Encripta/desencripta datos en modo ECB

mcrypt_ofb

Encripta/desencripta datos en modo OFB

En el servidor de WEB se pueden **proteger los directorios WEB** contra ataques asociando permisologías de acceso a los usuarios y a los programas. Se pueden utilizar las funciones de **autenticación y autorización** disponibles en el sistema de operación.

Se pueden definir adicionalmente **passwords** propios de las aplicaciones PHP, de modo de controlar el acceso a nivel de aplicaciones y no a nivel del servidor de web o a nivel del sistema de operación de la máquina donde este se encuentra operando.

Se debe evitar a toda costa **realizar llamadas a comandos del sistema** parametrizables con datos introducidos por el usuario del sistema. Por ejemplo en Linux ofrecer una posibilidad de eliminar un archivo donde el nombre del archivo se saca del formulario y el usuario de manera mal intencionada escriba "/etc/passwd"; de esta manera estaría eliminando todas las cuentas del sistema de operación donde opera el servidor de web.

LECCION 22 AUTENTICACIÓN HTTP CON PHP

Conceptos

Autenticación es el proceso utilizado para determinar si el usuario es quién dice ser. El mencanismo más sencillo es utilizar una clave con un "password", pueden ser utilizados mecanismos más sofisticados que monitorean el comportamiento de un usuario durante la utilización de una aplicación.

Autorización es el proceso utilizado para otorgar la permisología de utilización de los recursos del sistema.

Consideraciones

Las características de autentificación HTTP en PHP solo están disponibles cuando PHP se ejecuta como un módulo en Apache. En un script PHP como módulo de Apache, se puede utilizar la función **header()** para enviar un mensaje de "Autentificación requerida" al navegador cliente . De esta manera cuando se invoca el cliente se muestra una solicitando nombre de usuario y contraseña. Una vez el usuario haya colocado los valores para el nombre y la contraseña, el URL conteniendo el script PHP será invocado de nuevo con las variables predefinidas *PHP_AUTH_USER*, *PHP_AUTH_PW*, y *AUTH_TYPE* asignadas con el nombre de usuario, la contraseña y el tipo de autentificación respectivamente.

A continuación se muestra un ejemplo

```
<?php
 if (!isset($_SERVER['PHP_AUTH_USER'])) {
  header('WWW-Authenticate: Basic realm="My Realm"');
  header('HTTP/1.0 401 Unauthorized');
  echo 'Prueba autenticación';
  exit;
 } else {
  echo "<p>Bienvenido {$_SERVER['PHP_AUTH_USER']}.</p>";
  echo "<p>Usted colocó {$_SERVER['PHP_AUTH_PW']} como su clave.</p>";
 }
?>
```

PHP_LIB

PHP_LIB es una librería de clases en PHP. Esta librería de clases ofrece las clases **Auth** y **Perm** las cuales tienen funciones ofreciendo las facilidades de Autenticación y Autorización respectivamente.

Un enlace al código fuente de la clase Auth puede ser conseguido en **http://php.rufy.com/Auth.class.phps**

Autenticación personalizada

Las aplicaciones PHP pueden contener sus propios mecanismos de autenticación. La estrategia más común consiste en almacenar usuarios con sus respectivos “passwords” en una base de datos interna del sistema. También se pueden utilizar “cookies” y “secciones” los cuales serán cubiertos en las siguientes secciones.

LECCION 23 COOKIES

PHP soporta de manera nativa el manejo de cookies HTTP. Las cookies son un mecanismo utilizado para almacenar datos en el navegador utilizado por el cliente (usuario remoto de la aplicación). El objetivo de los cookies es poder identificar al usuario la próxima vez que utilice la aplicación (o para guardar información previamente introducida).

La función que permite crear las coookies es **setcookies()**. Las cookies forman parte del encabezado de HTTP, por tanto la función **setcookie()** debe ser llamada antes de producirse cualquier salida al navegador. Esta limitación es la misma de la función **header()**.
Cualquier cookie enviada a un script PHP por el cliente automáticamente es interpretada como una variable en PHP del mismo modo como se ocurre con los métodos de datos GET y POST. Una cookie también puede ser trabajada como un arreglo de modo de manejar múltiples valores añadinedo simplemente *[]* a el nombre del cookie.
En PHP 4.1.0 y posteriores, la matriz auto-global *$_COOKIE* será siempre actualizada con cualquier cookie mandada por el cliente. *$HTTP_COOKIE_VARS* es tambien actualizada en versiones anteriores de PHP cuando la variable de configuración *track_vars* se encuentre activada. (Siempre activada a partir de PHP 4.0.3.).

Función setcookie

Disponible en (PHP 3, PHP 4 , PHP 5)

int **setcookie** (string nombre, string valor, int expiracion, string camino, string dominio, int seguridad)

setcookie() define una cookie para ser enviada con el resto de la información del encabezado. Las cookies deben enviarse *antes* de mandar cualquier otra información del 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 se borrará del cliente remoto. También puede sustituir cualquier parámetro por una cadena de texto vacía (*""*) y saltar así ese parámetro. Los parámetros *expiracion* y *seguridad* son números enteros y no se pueden saltar con una cadena de texto vacía. En su lugar utilice un cero (*0*). El parámetro *expiracion* es un entero de tiempo típico de UNIX tal como lo devuelven las funciones. El parámetro *securidad* indica que la cookie se debe transmitir única y exclusivamente sobre una conexión segura HTTPS.

Consideraciones:
Las cookies no se hacen visibles hasta la siguiente carga de la página.
Las llamadas múltiples a **setcookie()** en el mismo script se ejecutarán en orden inverso. Si está intentando borrar una cookie antes de insertar otra, debe situar la llamada de inserción antes de lamamda de borrado.

A continuación se muestran algunos ejemplos::

```
setcookie("Prueba","Valor de Preuba ");
setcookie("Prueba ",$valor,time()+3600);  /* expira en 1 hora */
setcookie("Prueba ",$valor,time()+3600,"/~profe/",
"www.institutiogala.com",1);
```

Para utilizar de un cookie de prueba en un script se utiliza la siguiente sintáxis:

```
echo $HTTP_COOKIE_VARS["Prueba"];
```

A continuación se utiliza el cookie como un arreglo.

```
setcookie( "cookie[3]", "cookietres" );
setcookie( "cookie[2]", "cookiedos" );
setcookie( "cookie[1]", "cookieuno" );
if ( isset( $cookie ) ) {
  while( list( $nombre, $valor ) = each( $cookie ) ) {
    echo "$nombre == $valor\n";
  }
}
```

LECCION 24 ENVÍO DE ARCHIVOS PUT Y POST

Envío de archivos POST

PHP es capaz de recibir archivos de cualquier navegador soportando la norma RFC-1867 (entre los que se incluyen Netscape Navigator 3 o posterior, Microsoft Internet Explorer 3 o posterior sin éste). Ésta característica permite a los usuarios enviar archivos de texto y binarios. Mediante la autentificación y funciones de manejo de archivos de PHP, es posible restringir el envío y el procesamiento de los archivos.

Una página de envío de archivos se puede crear mediante un formulario de la siguiente manera:

```
<form enctype="multipart/form-data" action="_URL_" method="post">

<input type="hidden" name="MAX_FILE_SIZE" value="1000">

Enviar el archive : <input name="userfile" type="file">

<input type="submit" value="Send File">

</form>
```

La _URL_ debe tener como destino un script PHP. El input oculto MAX_FILE_SIZE debe encontrarse antes del input de tipo "file" para indicar el tamaño máximo de archivo que se puede enviar en bytes.
Las variables utilizadas en los archivos son :

$HTTP_POST_FILES['userfile']['name']

El nombre original del archivo en la máquina cliente.

$HTTP_POST_FILES['userfile']['type']

El tipo mime del archivo (si el navegador lo proporciona). Un ejemplo podría ser *"image/gif"*.

$HTTP_POST_FILES['userfile']['size']

El tamaño en bytes del archivo recibido.

$HTTP_POST_FILES['userfile']['tmp_name']

El nombre del archivo temporal utilizado para almacenar en el servidor el archivo recibido.

Una manera elegante de asignar las variables si fueron enviadas vía POST o vía GET es la siguiente instrucción

```
<?php
$PARAMS = ( isset($HTTP_POST_VARS))
? $HTTP_POST_VARS : $HTTP_GET_VARS;
?>
```

Envío de archivos PUT

PHP soporta el método HTTP PUT utilizado en aplicaciones como Netscape Composer y Amaya del W3C. Las peticiones PUT son más sencillas que el método POST. A continuación se muestra Un ejemplo:

```
PUT /path/filename.html HTTP/1.1
```

Esta instrucción se interpreta como la petición de un cliente remoto para salvar el contenido como: /path/filename.html en la página web donde se encuentra el servidor. Lógicamente no es una buena idea que la gente pueda escribir en la página WEB.

LECCION 25 TRABAJANDO CON ARCHIVOS REMOTOS

Al tener habilitada la opción ***allow_url_fopen*** php.ini, se pueden trabajar con URLs HTTP y FTP en la mayoría de las funciones manejando como parámetros un archivo. Además URLs pueden ser usadas con **include()**, **include_once()**, **require()** y **require_once()**.

Nota: Las versiones para windows de PHP anteriores a PHP 4.3 no soportaban acceso remoto a archivos en las funciones siguientes: **include()**, **include_once()**, **require()**, **require_once()**.

A continuación se muestra un ejemplo.

```
<?php
$url = fopen ("http://www.institutogala.com/", "r");
if (!$url) {
   echo "<p>No es posible abrir el archive remoto.\n";
   exit;
}
while (!feof ($url)) {
   $linea = fgets ($url, 1024);
```



```
   /* Esto trabaja si el tag “title” se encuentra en la linea */
   if (eregi ("<title>(.*)</title>", $linea, $res)) {
      $titulo = $res[1];
      break;
   }
}
fclose($url);
?>
```

También se puede escribir archivos en un servidor FTP (siempre que se conecte como un usuario con los privilegios de acceso de acceso). Solamente se pueden crear nuevos archivos usando este método; si se intenta sobreescribir un archivo ya existente, la función **fopen()** fallará
Para conectar como un usuario distinto de 'anonymous', se necesita especificar el nombre de usuario (y posiblemente contraseña) dentro de la URL, tales como 'ftp://usuario:clave@ftp.ejemplo.com/camino/a/archivo'. (Se puede utilizar la misma clase de sintaxis para acceder a archivos vía HTTP cuando se requería una autenticación).
A continuación se muestra un ejemplo

```
<?php
$arch = fopen ("ftp://ftp.institutogala.com/recibidos/archivo", "w");
if (!arch) {
   echo "<p>No se puede abrir el archive remoto para escritura.\n";
   exit;
}
/*escritura. */
fwrite ($arch, $_SERVER['HTTP_USER_AGENT'] . "\n");
fclose ($arch);
?>
```

LECCION 26 MANEJO DE CONEXIONES TRABAJANDO CON ARCHIVOS REMOTOS

El manejo de conexiones de PHP se implementa después de la versión 3.0.7.
Internamente en PHP se mantiene el estado de la conexión. Hay 3 posibles estados:

- 0 - NORMAL
- 1 - ABORTED (Abortado)
- 2 - TIMEOUT (Fuera de tiempo)

Cuando un script PHP se ejecuta se activa el estado NORMAL. Si el cliente remoto se desconecta, se pasa al estado ABORTED. Esto suele ocurrir cuando el usuario pulsa en el botón STOP del navegador. Si se alcanza el límite de tiempo impuesto por PHP (con la instrucción **set_time_limit()**), se pasa al estado TIMEOUT.
El comportamiento por defecto de un script es el aborto cuando un cliente remoto se desconecta. Este comportamiento puede ser configurado vía la directiva ignore_user_abort en el archvio php.ini, o empleando la función **ignore_user_abort()**.

Los scripts también se puede terminar cuando se cumple el tiempo controlado por un temporizador interno. El timeout por defecto es de 30 segundos. Se puede cambiar utilizando la directiva max_execution_time en el archivo php.ini o la correspondiente directiva php_max_execution_time en la configuración del servidor de páginas Apache, como también con la función **set_time_limit()**.

LECCION 27 MODO SEGURO EN SERVIDORES COMPARTIDOS

El Modo Seguro de PHP es un intento de resolver el problema de la seguridad en un servidor compartido. Tratar de resolver este problema al nivel de PHP es incorrecto desde el punto de vista de arquitectura . Aunque son utilizadas porque las alternativas disponibles en un servidor web y a

niveles de sistemas operativos no son tan realistas. Los proveedores de servicios de Internet ISP, utilizan estas funciones.

Las directivas de Configuración que controlan el Modo Seguro son:

Directiva	Valor por Omisión
safe_mode	*Off*
safe_mode_gid	*0*
safe_mode_include_dir	*""*
safe_mode_exec_dir	*1*
open_basedir	*""*
safe_mode_allowed_env_vars	*PHP_*
safe_mode_protected_env_vars	*LD_LIBRARY_PATH*
disable_functions	*""*

Cuando **safe_mode** está en On, PHP verifica si el dueño del script actual coincide con el dueño del archivo a ser operado por una función de archivo.
Al establecerse un modo seguro es posible que ciertas funciones no puedan ser ejecutadas.

LECCION 28 PHP DESDE LA LÍNEA DE COMANDOS

A partir de la versión 4.3.0, *PHP* soporta un nuevo tipo de *SAPI* (Interfaz De Programación De Uso Del Servidor) llamada *CLI* . CLI es la abreviatura de *interfaz de línea de comando (Command Line Interface)*. Como el nombre implica, este tipo de *SAPI* se focaliza en la creación de aplicaciones para ser ejecutadas desde la línea de comando (o desde el "desktop "también) con *PHP*.

La interfaz llamada *CLI SAPI* fue introducida con *PHP 4.2.0*, pero todavía se encuentra en estado experimental y tiene que ser activada explícitamente con *--enable-cli* utilizando *./configure*. Desde *PHP 4.3.0* la interfaz *CLI SAPI* es activada automáticamente. Se puede utilizar *--disable-cli* para de-activarla.

Los archivos de *PHP 4.2.0* y *PHP 4.2.3* distribuían el CLI como php-cli.exe, y los mantenía en el mismo directorio que el CGI php.exe. A partir de *PHP 4.3.0* el archivo para windows distribuye el CLI como php.exe en un directorio llamado cli; o sea *cli/php.exe*.
Desde la línea de comando, ejecutando *php -v* indicará si *php* es *CGI* o *CLI*.
Se puede ejecutar php –h para conocer las opciones disponibles de la versión de php instalado

```
Usage: php [options] [-f] <file> [args...]

    php [options] -r <code> [args...]

    php [options] [-- args...]

 -s            Display colour syntax highlighted source.

 -w            Display source with stripped comments and whitespace.

 -f <file>       Parse <file>.

 -v            Version number

 -c <path>|<file> Look for php.ini file in this directory

 -a            Run interactively

 -d foo[=bar]     Define INI entry foo with value 'bar'

 -e            Generate extended information for debugger/profiler

 -z <file>       Load Zend extension <file>.

 -l            Syntax check only (lint)

 -m            Show compiled in modules

 -i            PHP information
```



```
-r <code>       Run PHP <code> without using script tags <?..?>

-h              This help


args...         Arguments passed to script. Use -- args when first argument

                starts with - or script is read from stdin
```

Para indicar a php la ejecución de un script .

```
php script.php


php -f script.php
```

Con la opción de la línea de comandos –r se puede ejecutar una instrucción directamente.

```
php -r 'print_r(get_defined_constants());'
```

LECCION 29 RECOLECCION DE BASURA EN PHP

Tradicionalmente, los mecanismos que contabilizan las referencias en memoria, tal como el que usaba PHP anteriormente, fallaban al manejar las fugas de memoria en referencias cíclicas. Sin embargo, desde PHP 5.3.0 implementa el algoritmo síncrono de Recolección de Ciclos Concurrentes en Sistemas de Contabilidad de Referencias resolviendo este asunto.

Una explicación detallada del funcionamiento del algoritmo queda más allá del objetivo de este curso, pero aquí se explicaremos el mecanismo básico. Antes de nada, se deben establecer unas reglas del juego. Si se incrementa un refcount, entonces sigue en uso, no es basura. Si se decrementa el refcount, y alcanza a cero, el zval puede eliminarse. Esto significa que la recolección de ciclos sólo puede llevarse a cabo cuando un parámetro refcount se decrementa a un valor que no sea cero. En segundo lugar, en la recolección de ciclos de basura, es posible averiguar qué partes son basura comprobando si se puede decrementar en uno sus refcount, para después comprobar cuáles han alcanzado a cero.

REFERENCIAS

www.php.net

