



PHP. Nivel II

abril, 2019



Objetivos del nivel

- Aprender a programar orientado a objetos en PHP
- Generar documentos PDF dinámicamente
- Manejo de sesiones
- Manejo de cookies

Hacia quien está Orientado el curso

Desarrolladores y programadores que deseen profundizar en herramientas basadas en PHP a partir del paradigma POO.

Prerrequisitos del nivel

- PHP Nivel I
- Programación Orientada a Objetos Nivel II

Acerca de este manual

Este manual pertenece al Centro de Asesoramiento y Desarrollo Informático C.A. (CADI F1). Para obtener más información sobre este u otros cursos visite nuestra sitio Web www.cadif1.com, escribanos a la dirección de correo cadi@cadif1.com o visítenos en nuestra sede ubicada en la Av. Pedro León Torres con calle 59, Centro Comercial Sotavento, piso 2 oficina 27, Barquisimeto estado Lara, Venezuela. Tlf. 0251-7179247, 0251-4410268.

Las marcas mencionadas en este manual son propiedad de sus respectivos dueños. Copyright 2019. Todos los derechos reservados.



Contenido del nivel

Capítulo 1. Funciones

- 1.1.- Definición de Funciones en php.
- 1.2.- Llamar una Función.
- 1.3.- Paso de Parámetros a Las Funciones.

Capítulo 2. Manejo de Fechas

- 2.1.- Zonas Horarias.
- 2.2.- La Función Getdate().
- 2.3.- La Función Date.
- 2.4.- Time.

Capítulo 3. Librerías

- 3.1.- Creación de Librerías.
- 3.2.- Incluir Librerías en Páginas.
- 3.3.- Namespaces.

Capítulo 4. Cookies

- 4.1.- Qué es Una Cookie.
- 4.2.- Guardar Una Cookie.
- 4.3.- Recuperar Una Cookie.
- 4.4.- Modificar y Eliminar Una Cookie.

Capítulo 5. Manejo de Sesiones. Parte 1

- 5.1.- Concepto de una Sesión.
- 5.2.- Crear una Variable de Sesión.
- 5.3.- Destruir Una Variable de Sesión.

Capítulo 6. Manejo de Sesiones. Parte 2

- 6.1.- Formulario.
- 6.2.- Validación.
- 6.3.- Verificación.

Capítulo 7. Declaración de Clases en Php

- 7.1.- Definición de Una Clase.
- 7.2.- Creación y Uso Objetos.
- 7.3.- Niveles de Acceso.

Capítulo 8. Métodos. Parte 1

- 8.1.- Definición de Métodos.
- 8.2.- Llamado de Los Métodos.
- 8.3.- Acceso a Los Atributos.

Capítulo 9. Métodos. Parte 2

- 9.1.- Getter y Setter.
- 9.2.- Constructores.
- 9.3.- Métodos Estáticos.

Capítulo 10. Herencia

- 10.1.- Herencia.
- 10.2.- Herencia en Constructores.
- 10.3.- Funciones Relacionadas.

Capítulo 11. Clases Abstractas

- 11.1.- Clases Abstractas.
- 11.2.- Métodos Abstractos.
- 11.3.- Interfaces.

Capítulo 13. Generación de Archivos Pdf. Parte 1

- 13.1.- ¿Qué es FPDF?.
- 13.2.- Creación de un archivo PDF.
- 13.3.- Agregar Contenido a un Documento.

Capítulo 14. Generación de Archivos Pdf. Parte 2

- 14.1.- Agregar Imágenes.
- 14.2.- Encabezado y Pie de Página.

Capítulo 15. Envío de Archivos

- 15.1.- Campo de Archivos.
- 15.2.- Procesar Archivos Subidos.
- 15.3.- Configuración de Php.



Capítulo 1. FUNCIONES

1.1.- Definición de Funciones en php

Una función se puede definir con la siguiente sintaxis:

```
<?php
    function foo ($arg_1, $arg_2, ..., $arg_n)
    {
        echo "Función de ejemplo.";
        return $retval;
    }
?>
```

El tipo de dato devuelto por la función dependerá de la variable que se coloque en la cláusula return. Cualquier instrucción válida de PHP puede aparecer en el cuerpo de la función, incluso invocar otras funciones y definiciones de clases. La función puede tener cualquier cantidad de parámetros, en caso de no tener parámetros igual se deben colocar los paréntesis.

Dentro del cuerpo de la función se pueden declarar variables, que solo existirán mientras se esté ejecutando la función. Estas variables se declaran igual que cualquier variable en PHP (inicializándola con un valor cualquiera). Por ejemplo:

```
<?php

function esMayor($edad)
{
    if ($edad >=18) return true;
    else return false;
}

?>
```

También se puede hacer referencia a variables globales en la función, pero se debe "declarar" la variable usando la palabra reservada "global", de lo contrario se tomará como una variable local. Por ejemplo:

```
<?php

$dias=array("lunes","martes","miercoles","jueves",
            "viernes","sabado","domingo");

// si no hacemos la definición de la variable $dias con la palabra global,
// ocurrirá un error como si la variable no existiera.

function mostrarDias()
{
    global $dias; // aqui estamos diciendo que vamos a usar la var global

    for ($i=0;$i<count($dias);$i++)
        echo $dias[$i]. " \n";
}
```

1.2.- Llamar una Función

Las funciones pueden llamarse en cualquier lugar del código PHP, incluso antes de definirse. Se llaman colocando el nombre de la función y entre paréntesis los parámetros que ella espera. Al llamar una función no existe sensibilidad a mayúsculas y minúsculas. Por ejemplo:

```
<?php

echo "El sueldo es " . calcularSueldo(750, 15);

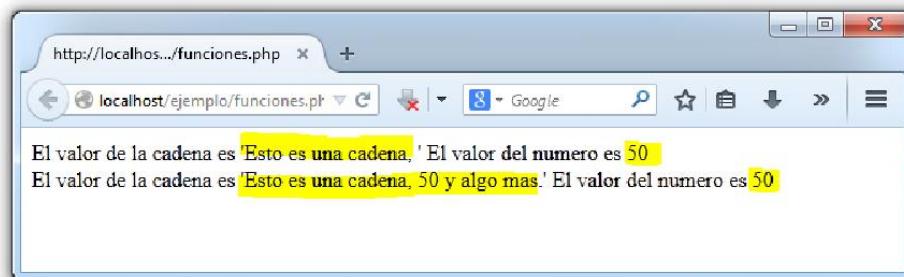
function calcularSueldo($sueldobase, $antiguedad )
{
    return $sueldobase + ($antiguedad * 20 );
}
```



1.3.- Paso de Parámetros a Las Funciones

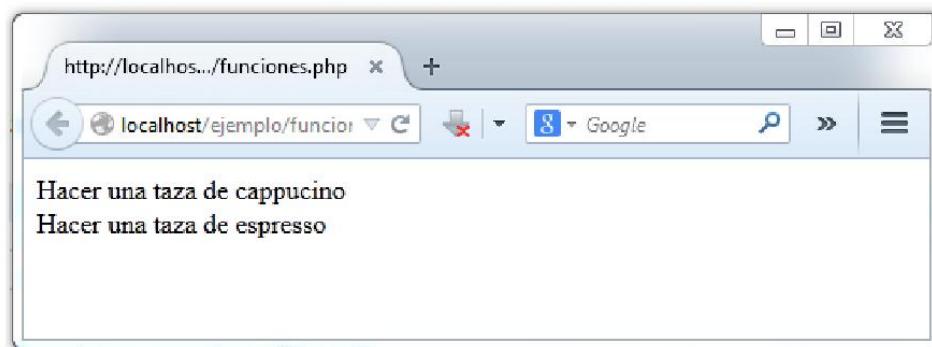
PHP soporta pasar parámetros por valor (el comportamiento por defecto), por referencia y parámetros por defecto. Mientras no se especifique lo contrario, los parámetros de una función se pasan por valor. Si se necesita que un parámetro de una función se pase por referencia, se antepone el ampersand (&) al nombre del parámetro en la definición de la función:

```
<?php
    function imprimir($str,$n)
    {
        echo "El valor de la cadena es '$str' ";
        echo "El valor del numero es $n <br>";
    }
    function agregar_extra(&$cadena,$n)
    {
        $cadena = $cadena . $n ." y algo mas.";
        $n=0;
    }
    $str = "Esto es una cadena, ";
    $n=50;
    imprimir($str,$n);
    agregar_extra($str,$n);
    imprimir($str,$n);
```



Una función puede definir valores por defecto para los parámetros, por ejemplo:

```
<?php  
  
    function makecoffee ($type = "cappuccino")  
    {  
        return "Hacer una taza de $type <br>";  
    }  
  
    echo makecoffee();  
    echo makecoffee("espresso");
```



El valor por defecto tiene que ser una expresión constante y no una variable ó llamada a una función. Cabe destacar que cuando se usan parámetros por defecto, estos tienen que estar a la derecha de cualquier parámetro sin valor por defecto.

En ocasiones podriamos necesitar alguna función que permita operar sin conocer la cantidad exacta de parámetros que debemos pasarles. Esto se conoce como funciones con parámetros variables. Dentro de las funciones podemos utilizar otras funciones propias de PHP como:

func_num_args Devuelve el numero de argumentos pasados

func_get_arg Devuelve un argumento

func_get_args Devuelve todos los argumentos en una matriz

Por ejemplo, si necesitamos una función que concatene palabras y que muestre la frase final, se puede hacer de la siguiente forma:

```

function conector(){
    $datos="";
    $argumentos= func_get_args();
    for($i=0;$i<func_num_args();$i++){
        $datos.=$argumentos[$i]." ";
    }
    echo $datos;
}

conector("hola","a todos");
// La salida es: hola a todos

```

Es importante no confundir los conceptos de funciones con parámetros definidos y funciones con parámetros variables, ya que si a una función se le han definido parámetros y al momento de su llamada no recibe los parámetros que está esperando, esto producirá un error:

```

<?php

function promediar_notas($arreglo_notas){
    $suma=0;
    foreach($arreglo_notas as $valor){
        $suma+=$valor;
    }
    return $suma/count($arreglo_notas);
}
$notas=array(15,12,18);
echo promediar_notas();

```



Warning: Missing argument 1 for promediar_notas(), called in C:\Users\asesor\Documents\proyecto\funciones.php on line 11 and defined in C:\Users\asesor\Documents\proyecto\funciones.php on line 3

Capítulo 2. MANEJO DE FECHAS

2.1.- Zonas Horarias

Antes de trabajar con las funciones de fecha y hora debemos establecer la zona horaria que necesitamos usar. Si no establecemos la zona horaria correcta, los valores de fecha y hora obtenidos pueden ser diferentes a los que esperamos. La zona horaria la establecemos con la función `date_default_timezone_set("zona_horaria")`. La zona horaria en nuestro país (Venezuela) es America/Caracas. Para no tener que colocar esta función antes de cualquier función de fecha y hora, podemos establecer en el `php.ini` el valor `date.timezone = mizona`.

La lista completa de los posibles valores de zona horaria la podemos encontrar en la url

<http://php.net/manual/es/timezones.php>. Con la función `date_default_timezone_get()` podemos determinar la zona horaria que se está usando. Veamos un ejemplo:

```
<?php  
  
echo "La zona horaria por defecto es: ".date_default_timezone_get()."<br>";  
date_default_timezone_set("America/caracas");  
echo "La nueva zona horaria es: ".date_default_timezone_get()."<br>";
```



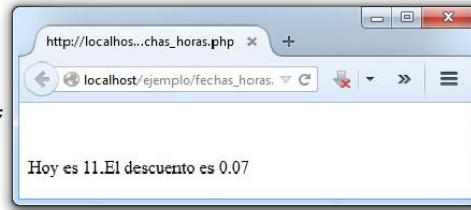
2.2.- La Función Getdate()

Devuelve un valor array asociativo que contiene información sobre la fecha y la hora local actual. Los valores que devuelve el `getdate` son:

Clave	Descripción	Ejemplo de valores devueltos
"seconds"	Representación numérica de segundos	0 a 59
"minutes"	Representación numérica de minutos	0 a 59
"hours"	Representación numérica de horas	0 a 23
"mday"	Representación numérica del día del mes	1 a 31
"wday"	Representación numérica del día de la semana	0 (para el Domingo) a 6 (para el Sábado)
"mon"	Representación numérica de un mes	1 a 12
"year"	Una representación numérica completa de un año, 4 dígitos	Ejemplos: 1999 o 2003
"yday"	Representación numérica del día del año	0 a 365
"weekday"	Una representación textual completa del día de la semana	Sunday a Saturday
"month"	Una representación textual completa de un mes, como January o March	January a December
0	Segundos desde el Epoch Unix, similar a los valores devueltos por time() y usados por date() .	Depende del sistema, típicamente -2147483648 a 2147483647.

Un ejemplo de como usar el getdate es:

```
<?php
date default timezone set("America/caracas");
$fecha=getdate();
echo "Hoy es ".$fecha['mday'];
if ($fecha['mday']<10) $descuento=0.05;
else
    if ($fecha['mday']<20) $descuento=0.07;
    else $descuento=0.1;
echo ".El descuento es $descuento";
?>
```



La función getdate también devuelve el día de la semana en número o en palabra, con los valores "weekday" y "wday". El día de la semana en palabras es retornado en inglés. Una manera de solventar esto podría ser:

```
<?php  
date_default_timezone_set("America/caracas");  
$dias=array("domingo","Lunes","Martes","Miercoles","Jueves","Viernes","Sabado");  
$fecha=getdate();  
echo "Hoy es ".$fecha['weekday']." (en ingles) <br>";  
echo "Hoy es ".$fecha['wday']." (en numero) <br>";  
echo "Hoy es ".$dias[$fecha['wday']]." (en castellano) ";
```



?>

Es importante recordar que el código php se ejecuta en el servidor y por lo tanto la fecha y la hora que se consigue es la del servidor. Este aspecto se debe tomar en cuenta cuando se está desarrollando una aplicación donde alguna funcionalidad crítica se base en la fecha y la hora.

En muchos casos la zona horaria del servidor no va a ser la misma que la del cliente. Aunque casi nunca interesa la hora del cliente, hay manera de averiguarla. La forma de conocer la fecha y la hora del cliente es usando la función Date de javascript.

2.3.- La Función Date

La función Date devuelve un string con la fecha o la hora local con un formato especificado. La forma de usar el date es \$fecha=date("formato"). Algunos ejemplos de como usar la función date son:

```
<?php

$fechahoy=date("d/m/Y"); // fecha de hoy en formato dd/mm/yyyy
$fechaMysql=date("Y-m-d"); // fecha de hoy en el formato de mysql

// fecha larga en el formato Lunes, 2 de Noviembre de 2008
$fechaLarga=date("l \, d \d\le F \d\le Y");

echo "$fechahoy | $fechaMysql | $fechaLarga";
```

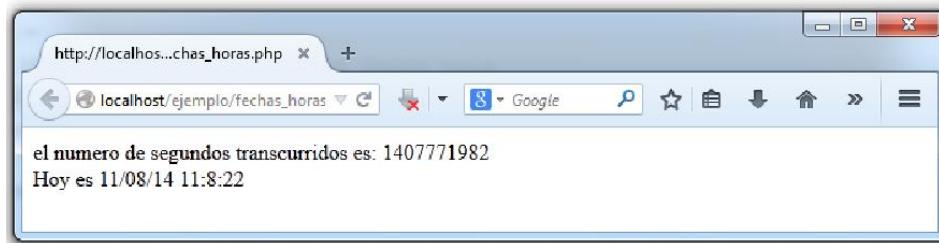
La cadena con el formato puede contener cualquiera de los siguientes caracteres:

<i>d</i>	Día del mes, 2 dígitos con ceros iniciales	01 a 31
<i>D</i>	Una representación textual de un día, tres letras	Mon a Sun
<i>j</i>	Día del mes sin ceros iniciales	1 a 31
<i>l</i> ("L" minúscula)	Una representación textual completa del día de la semana	Sunday a Saturday
<i>w</i>	Representación numérica del día de la semana	0 (para el Domingo) a 6 (para el Sábado)
<i>z</i>	El día del año (comenzando en 0)	0 a 365
<i>F</i>	Una representación textual completa de un mes, como January o March	January a December
<i>m</i>	Representación numérica de un mes, con ceros iniciales	01 a 12
<i>M</i>	Una representación textual corta de un mes, tres letras	Jan a Dec
<i>n</i>	Representación numérica de un mes, sin ceros iniciales	1 a 12
<i>t</i>	Número de días en el mes dado	28 a 31
<i>Y</i>	Una representación numérica completa de un año, 4 dígitos	Ejemplos: 1999 o 2003
<i>y</i>	Una representación de dos dígitos de un año	Ejemplos: 99 o 03
<i>a</i>	Ante meridiano y Post meridiano en minúsculas	am o pm
<i>g</i>	formato de 12-horas de una hora sin ceros iniciales	1 a 12
<i>G</i>	formato de 24-horas de una hora sin ceros iniciales	0 a 23
<i>h</i>	formato de 12-horas de una hora con ceros iniciales	01 a 12
<i>H</i>	formato de 24-horas de una hora con ceros iniciales	00 a 23
<i>i</i>	Minutos con ceros iniciales	00 a 59
<i>s</i>	Segundos, con ceros iniciales	00 a 59

2.4.- Time

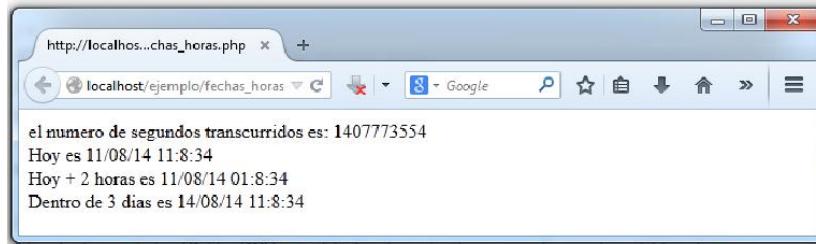
Otra forma de obtener valores del tiempo es con la función time(). Ésta devuelve un número entero que representa el número de segundos que han transcurrido desde la fecha 1 de Enero de 1970 00:00:00 (GMT), conocida también como la época Unix. Este valor es especialmente útil para hacer operaciones con fechas y horas, tales como compararlas, sumarle valores como días, segundos, minutos, calcular la diferencia entre fechas, etc. Veamos un ejemplo sencillo:

```
<?php  
// se establece la zona horaria  
date_default_timezone_set("America/caracas");  
// se determina la/hora en segundos  
$fechahora = time();  
echo "el numero de segundos transcurridos es: $fechahora <br>";  
// se imprime la fecha y hora de hoy con formato entendible  
echo "Hoy es ".date("d/m/y h:m:s")."<br>";
```



Al valor returnedo por la función time() se le puede hacer operaciones aritméticas. Por ejemplo sumarle 2 horas sería algo así como time() + 2 * 60 * 60, porque cada hora tiene 60 minutos y cada minuto tiene 60 segundos. El valor en segundos se puede mostrar en su correspondiente formato de fecha normal (d/m/y) u hora (h:m:s) o ambos inclusive, con la función "date", a la cual se le puede pasar el formato y el valor de fecha Unix. Veamos unos ejemplos:

```
<?php  
// se establece la zona horaria  
date_default_timezone_set("America/caracas");  
// se determina la/hora en segundos  
$fechahora = time();  
echo "el numero de segundos transcurridos es: $fechahora <br>";  
// se imprime la fecha y hora de hoy con formato entendible  
echo "Hoy es ".date("d/m/y h:n:s")."<br>";  
  
// se imprime la fecha y hora de hoy + 2 horas  
echo "Hoy + 2 horas es ".date("d/m/y h:n:s", $fechahora+(2*60*60)) . "<br>";  
  
// se imprime la fecha y hora de hoy + 3 dias  
echo "Dentro de 3 dias es ".date("d/m/y h:n:s", $fechahora+(3*24*60*60)) . "<br>";
```



Capítulo 3. LIBRERÍAS

3.1.- Creación de Librerías

PHP da la posibilidad de crear un archivo que contenga funciones generales definidas para poder usarlas en cualquier otro archivo, dándole así la posibilidad de reutilizar código. Para esto primero se debe crear un archivo de PHP donde se colocan todas las funciones y declaraciones. Por ejemplo:

```
<?php  
function mensajedeerror($mensaje)  
{  
    echo $mensaje;  
}  
?>
```

Puede definir todas las funciones, variables y/o constantes que necesiten en este archivo, sin importar el orden en el cual se coloquen. Se pueden tener tantos archivos php que sean usados como librerías como hagan falta.

3.2.- Incluir Librerías en Páginas

Para usar las funciones, variables o cualquier otra declaración desde otro archivo php, se debe incluir el archivo donde estan declaradas. Esto se hace con la función "require", "require_once", "include" o "include_once". El siguiente es un ejemplo:

```
<?php libreria.php  
  
function calcularSueldo($sueldobase, $antiguedad )  
{  
    return $sueldobase + ($antiguedad * 20 );  
}  
  
?>
```

```
<?php cualquierOtroArchivo.php  
  
// producira un error  
echo "El sueldo es " . calcularSueldo(750, 15);  
include("libreria.php");  
  
// ahora si funcionara  
echo "El sueldo es " . calcularSueldo(750, 15);  
  
?>
```

3.3.- Namespaces

A pesar que teóricamente no se pueden tener dos funciones y/o clases definidas con el mismo nombre, PHP lo permite hacerlo gracias a los Namespace. Los espacios de nombres son una manera de encapsular elementos. En el mundo de PHP, los espacios de nombres están diseñados para solucionar dos problemas con los que se encuentran los autores de bibliotecas y de aplicaciones al crear elementos de código reusable, tales como clases o funciones:

1. 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.
2. La capacidad de apodar (o abreviar) Nombres_Extra_Largos diseñada para aliviar el primer problema, mejorando la legibilidad del código fuente.

De esta manera se pueden definir funciones con el mismo nombre pero bajo espacios de nombres distintos.

Para definir un nombre de espacio a un archivo se usa la palabra reservada namespace. Su sintaxis es la siguiente:

```
<?php  
namespace nombreDeEspacio;  
  
?>
```

El nombre de espacio debe ser la primera instrucción de código PHP dentro del documento, si no es así, PHP mostrara un error.

Un ejemplo del uso de un namespace dentro un archivo php es el siguiente:

```
<?php  
namespace funciones;//Debe ser lo primero que aparezca en el documento  
  
function promediar_notas($arreglo_notas){  
    $suma=0;  
    foreach($arreglo_notas as $valor){  
        $suma+=$valor;  
    }  
    return $suma/count($arreglo_notas);  
}  
$notas=array(15,12,18);
```

Este ejemplo muestra como crear dos librerías distintas que contengan una función con el mismo nombre en ambas librerías, incluir las librerías en un tercer archivo PHP y usar las funciones especificando el espacio de nombres al que pertenecen.

```
funciones.php
<?php
namespace funciones;
function sumar($num1,$num2){
    return $num1+$num2;
}

funciones_extras.php
<?php
namespace funcionesExtras;
function sumar(){
    $args=func_get_args();//cargamos la variable $args con los parametros
    return array_sum($args);//array_sum devuelve la suma de los elementos
    //del arreglo
}

index.php
<?php
include("funciones.php");
include("funciones_extras.php");

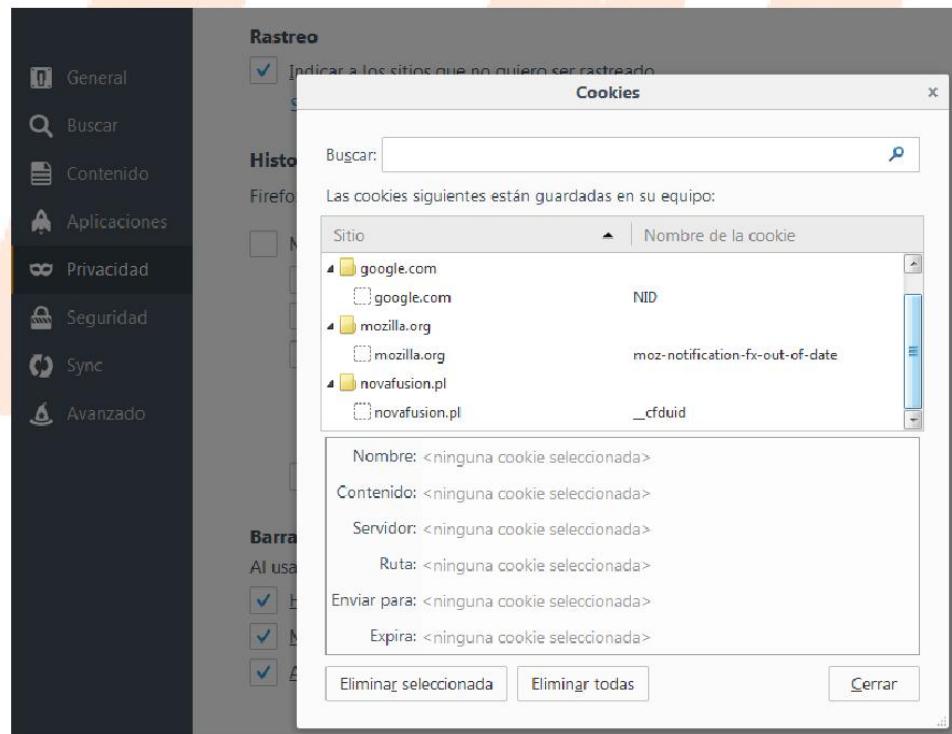
echo funciones\sumar(15,20)."  
";//funcion de la libreria "Libreria.php"
echo funcionesExtras\sumar(15,20);///funcion de la libreria "funciones_extras.php"
```

Capítulo 4. COOKIES

4.1.- Qué es Una Cookie

Las cookies son segmentos de texto de hasta 4 KB que se almacenan en la computadora del usuario. Esos segmentos de texto persistirán, incluso cuando la computadora del usuario este apagada, de modo que se almacena información acerca del usuario con facilidad. Las cookies se utilizan para todos los fines, desde los bien intencionados hasta los más siniestros, como registrar que anuncios ya ha visto un usuario y a los que ha respondido. PHP cuenta con soporte para establecer y leer cookies.

Los navegadores mediante sus opciones de configuración, le permiten al usuario ver las cookies que se han almacenado en su máquina, dándole la posibilidad de eliminarlas o simplemente ver su contenido. El siguiente ejemplo es del navegador Firefox, el cual en la pestaña de "Privacidad" de su menú de configuración muestra una lista de las cookies registradas hasta el momento:



4.2.- Guardar Una Cookie

Establecer y leer cookies en PHP no es difícil. Las cookies se alojan en la máquina del usuario con la función "setcookie" de PHP:

```
setcookie(name [, value[, expire [, path [,domain[,secure]]]]])
```

El significado de los parametros es el siguiente:

- name: el nombre de la cookie.
- value: el valor de la cookie.
- expire: el tiempo en que expira o caduca la cookie.
- path: ruta de acceso en el servidor al que estara disponible la cookie (opcional).
- domain: el dominio para el que esta disponible la cookie (opcional).
- secure: indica que la cookie solo debe transmitirse a través de una conexión HTTPS (opcional).

El siguiente ejemplo establece en una cookie llamada "message" el texto "Sin preocupaciones":

```
//nombre de la cookie: message  
//Valor de la cookie: Sin preocupaciones  
setcookie("message", "Sin preocupaciones");
```

4.3.- Recuperar Una Cookie

Por otra parte, para obtener las cookies que el navegador del usuario pueda tener almacenadas localmente, se utiliza el array asociativo `$_COOKIE`. En este array están todas las cookies que tiene disponible la página PHP en el dominio y el directorio donde está colocado.

De esta manera, si queremos recuperar una cookie del navegador del usuario, podemos hacerlo como sigue:

```
$_COOKIE["Nombre de la cookie"]
```

Donde "Nombre de la cookie" es precisamente el nombre que se estableció para esa cookie en la función setcookie().

4.4.- Modificar y Eliminar Una Cookie

El proceso de modificación de una Cookie es relativamente sencillo, ya que consiste en utilizar nuevamente la función setcookie() pero esta vez utilizando el nombre de la cookie que se desea modificar.

```
setcookie("nombre de la cookie","nuevo valor");
```

En el caso de la eliminación de la cookie, ldebemos hacerlo mediante la función unset() de PHP para eliminar del arreglo asociativo \$_COOKIE[] la clave de correspondiente:

```
unset($_COOKIE["nombre de la cookie"]);
```

Ejemplo completo de gestión de cookies:

```
Setting new cookie
=====
<?php
setcookie("name","value",time()+$int);
/*name is your cookie's name
value is cookie's value
$int is time of cookie expires*/
?>

Getting Cookie
=====
<?php
echo $_COOKIE["your cookie name"];
?>

Updating Cookie
=====
<?php
setcookie("color","red");
echo $_COOKIE["color"];
/*color is red*/
/* your codes and functions*/
setcookie("color","blue");
echo $_COOKIE["color"];
/*new color is blue*/
?>

Deleting Cookie
=====
<?php
unset($_COOKIE["yourcookie"]);
/*Or*/
setcookie("yourcookie","yourvalue",time()-1);
/*it expired so it's deleted*/
?>
```

Capítulo 5. MANEJO DE SESIONES. PARTE 1

5.1.- Concepto de una Sesión

Básicamente una sesión es la secuencia de páginas que un usuario visita en un sitio web. Desde que entra en nuestro sitio (colocando en la barra de direcciones del navegador el nombre de una página de nuestro sitio), hasta que lo abandona (cerrando la ventana del navegador).

El término sesión en PHP, se aplica a esta secuencia de navegación. Para ello se crea un identificador único que asignamos a cada una de estas sesiones de navegación. A este identificador de sesión se le denomina, comúnmente "la sesión".

El proceso en cualquier lenguaje de programación podría ser algo así: ¿Existe una sesión?, si existe lo retomamos, si no existe la creamos y le damos un identificador único. En resumen, una sesión va a ser una o varias variables, cuyo valor(es) no se perderá(n) cuando pasamos de una página a otra.

5.2.- Crear una Variable de Sesión

Antes de poder hacer referencia o crear una variable de sesión, se debe inicializar la sesión. Para lograrlo se utiliza la función "session_start" antes de intentar utilizar cualquier variable de sesión. Esta función inicializa el manejo interno de las sesiones en PHP, de no invocarlo nada que tenga que ver con las sesiones funcionará.

Existen dos maneras de referirse a las variables de sesión, una con la variable global `$_SESSION` y otra con la variable global `$HTTP_SESSION_VARS`. Ambas son arreglos asociativos. Se recomienda usar `$_SESSION` (o `$HTTP_SESSION_VARS` con PHP 4.0.6 o inferior) por seguridad y para hacer el código más legible.

Las opciones de configuración `track_vars` y `register_globals` influyen notablemente en la forma en que las variables de la sesión se almacenan y restauran.

Por ejemplo, si queremos contar la cantidad de páginas que visita un usuario en nuestro sitio debemos colocar al comienzo de cada una de las páginas lo siguiente:

```
<?php  
    session_start();  
    if (isset( $_SESSION["visitas"] ) ) $_SESSION["visitas"]++;  
    else $_SESSION["visitas"]=0;  
?>
```

Se utiliza la función `isset` para verificar si la variable existe, de no existir se crea inicializándola con un valor arbitrario (como se hace con cualquier otra variable de PHP). La siguiente vez que el usuario visite la página (u otra página), la variable va a existir y se va a incrementar el valor que ella trae. El nombre que se le asigne a la variable (en el ejemplo anterior "visitas") es arbitrario también.

5.3.- Destruir Una Variable de Sesión

El valor de una variable de sesión se pierde o desaparece cuando liberamos explícitamente la variable o cuando el usuario cierra la ventana del navegador donde inició la visita a nuestro sitio. Destruir una variable de sesión significa eliminar una variable que se haya creado anteriormente. Esto es útil en sistemas como los de correo en los cuales hay un botón donde el usuario decide salir del sistema sin necesidad de cerrar la ventana del navegador, para así evitar el acceso a las páginas privadas.

La destrucción de las variables de sesión se hace con el procedimiento "unset", al cual se le pasa por parámetro el nombre de la variable que se quiere destruir. Por ejemplo:

```
<?php  
    session_start();  
    unset($_SESSION["visitas"]);  
?>
```

Capítulo 6. MANEJO DE SESIONES. PARTE 2

6.1.- Formulario

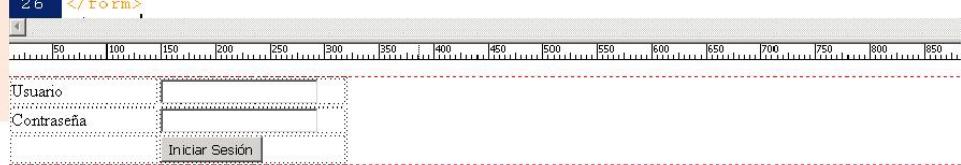
El uso más frecuente que se le da a las variables de sesión es en las páginas donde se solicitan nombre de usuario y contraseña a los usuarios para poder entrar en algunas páginas privadas (accesibles solo para usuario registrados).

Luego se crea una página con un formulario html que contenga 2 cuadros de texto, uno para que el usuario introduzca el "Nombre de Usuario" y otro para la "Contraseña". Al cuadro de texto para la contraseña se le debe colocar en las propiedades tipo "Contraseña" para que aparezcan asteriscos "*" mientras el usuario escribe su contraseña.

```

9 <form id="form1" name="form1" method="post" action="verificarlogin.php">
10 <table width="311" border="0">
11   <tr>
12     <td width="133">Usuario</td>
13     <td width="168"><label>
14       <input name="txtusuario" type="text" id="txtusuario" />
15     </label></td>
16   </tr>
17   <tr>
18     <td>Contrase&ntilde;a</td>
19     <td><input name="txtclave" type="password" id="txtclave" /></td>
20   </tr>
21   <tr>
22     <td>&nbsp;</td>
23     <td><input type="submit" name="Submit" value="Iniciar Sesi&acute;n" /></td>
24   </tr>
25 </table>
26 </form>

```



6.2.- Validación

Los datos solicitados en este formulario serán enviados a la página verificarlogin.php, en la cual debe estar el código para verificar en la tabla "usuarios" de la base de datos si existe un registro con el que coincida el nombre de usuario y la contraseña.

En resumen, es hacer una búsqueda en la tabla usuarios, pero buscando a través de dos campos. Para esto se debe usar la instrucción SQL "select" con la cláusula "where". El código debe ser algo como el siguiente:

```
<?php
$cexion=mysql_connect ("localhost", "root", "1234") or die(mysql_error());
mysql_select_db("bdprueba",$conexion) or die(mysql_error());

$usuario=$_POST['txtusuario'];
$clave=$_POST['txtclave'];
$sql="select * from usuarios where usuario='".$usuario."' and clave='".$clave."'";
$busqueda = mysql_query($sql,$conexion);
if (mysql_num_rows($busqueda)==0) echo "Usuario o contraseña incorrectos";
else
{
    session_start();
    $_SESSION['usuario']=$usuario;
    // se puede redireccionar a una página índice de las páginas privadas
}
?>
```

6.3.- Verificación

Para complementar esto, todas las páginas privadas deben tener una validación donde se verifique si el usuario ya ha iniciado sesión. Se trata de verificar si la variable de sesión que se crea cuando el visitante coloca "usuario" y "clave" correctamente existe, si no existe la variable de sesión, significa que no se ha logueado y por lo tanto no puede ver la página privada que está intentando accesar.

En ese caso, se debe redireccionar al visitante hacia la página de seguridad para que coloque "usuario" y "clave" correctamente o para que se registre como nuevo usuario (en caso de permitirlo). Esta validación debe colocarse al comienzo de la página.

El código de validación debe ser algo como el siguiente:

```
<?php
session_start();
if (isset($_SESSION['usuario'])==false) // si no existe la variable $_SESSION['usuario']
header("location:busqueda.php"); // se redirecciona a la página de login y password
?>
```

Capítulo 7. DECLARACIÓN DE CLASES EN PHP

7.1.- Definición de Una Clase

A partir de PHP5 se cubren las carencias de las versiones anteriores, proveyendo un soporte completo para la Programación Orientada a Objetos (POO). Este paradigma se logra gracias al nuevo motor (Zend Engine 2) que proporciona todos los mecanismos necesarios para este propósito, de esta manera PHP puede ser empleado desde dos perspectivas distintas, la Programación Procedural y Orientada a Objetos.

Las clases en php se definen usando la palabra reservada class, seguido del nombre de la clase y un par de llaves que indican el ámbito de la clase. Dentro de las llaves, se declaran los atributos y los métodos. Los atributos se declaran usando la palabra reservada "var". Estos atributos no tienen tipo hasta que se les asigna un valor. Un atributo puede ser un entero, un arreglo, un arreglo asociativo o incluso otro objeto.

El siguiente es un ejemplo de la declaración de una clase:

```
1  <?php
2      class Usuario
3  {
4      var $nombre;
5      var $contraseña;
6      var $correo;
7
8
9  }
10
11 ?>
```

7.2.- Creación y Uso Objetos

Para instanciar una clase (dicho de otra manera, para crear un objeto de una clase), se usa el operador "new". Luego, a través del objeto se hace referencia a cualquier miembro de la clase (propiedad o método), usando el operador "->". El siguiente es un ejemplo:

```
1 <?php
2     class Usuario
3     {
4         var $nombre;
5         var $contraseña;
6         var $correo;
7     }
8
9
10    // causa un error porque el objeto no ha sido instanciado
11    $usu_1->nombre="Jose luis";
12    echo "el nombre del usuario 1 es ".$usu_1->nombre;
13
14    // se instancia el objeto
15    $usu_2 = new Usuario();
16    // se accede a sus atributos
17    $usu_2->nombre="Manuel";
18    echo "el nombre del usuario 2 es ".$usu_2->nombre;
19 ?>
```

7.3.- Niveles de Acceso

Hasta ahora, todas las propiedades o atributos que se han usado son "públicos" (en php los atributos son públicos mientras que no se indique lo contrario). Un atributo público es aquel que puede ser accedido desde fuera del objeto, simplemente escribiendo algo como: \$objeto->atributo.

Tener atributos públicos, sin embargo, no es la práctica más recomendable, porque si desde fuera del objeto se pueden modificar sus propiedades, se puede llegar a alterar el propósito o la funcionalidad del mismo. Por esto, para tener el control del uso de las propiedades del objeto, es necesario de indicar que un determinado atributo es interno al objeto, de su uso exclusivo, o dicho de otra forma, privado del objeto. Por tanto, solo accesible y/o modificable desde los métodos internos de la clase.

```

1 <?php
2     class Usuario
3     {
4         public $nombre;
5         private $contraseña;
6         private $correo;
7     }
8
9 ?>

```

Para hacer que un atributo sea privado, se debe declarar usando la palabra reservada "private". Así, cualquier referencia del tipo \$objeto->atributo provocará un error del intérprete de PHP. Es decir, cualquier intento de acceder externamente a un atributo privado no estará permitido.

```

1 <?php
2     class Usuario
3     {
4         public $nombre;
5         private $contraseña;
6         private $correo;
7     }
8
9
10    $usu_1 = new Usuario();
11    $usu_1->nombre="Jose luis";           // es valido
12    $usu_1->correo="joselrojas@cadif1.com"; // es invalido
13    $usu_1->contraseña="45678";          // es invalido
14
15 ?>

```

Como se puede ver en el ejemplo, los intentos de actualización de las propiedades "usuario" y "contraseña" son inválidos. La única forma de modificar y/o acceder a estos atributos es a través de métodos públicos. La práctica más recomendable es que todas las propiedades se declaran privadas y se accede a ellas de manera controlada mediante métodos denominados 'setters' y 'getters', los cuales, respectivamente, modifican y devuelven sus contenidos. Esto se tratará en el próximo capítulo.

Capítulo 8. MÉTODOS. PARTE 1

8.1.- Definición de Métodos

Los métodos se definen dentro de las llaves de la clase usando la palabra reservada `function`. El siguiente es un ejemplo sencillo:

```
1  <?php
2      class Usuario
3  {
4      public $nombre;
5      private $contraseña;
6      private $correo;
7
8      function mostrar_bienvenida()
9      {
10         echo "Bienvenido al sistema";
11     }
12 }
13 ?>
```

Los métodos pueden tener tantos parámetros como haga falta. Las reglas para usar parámetros en los métodos son las mismas que aplican a funciones tradicionales:

```
1  <?php
2      class Usuario
3  {
4          public $nombre;
5          private $contraseña;
6          private $correo;
7
8          function mostrar_bienvenida()
9          {
10             echo "Bienvenido al sistema";
11         }
12         function agregar_lista($n)
13         {
14             echo "<ul>";
15             for ($i=1;$i<$n;$i++)
16                 echo "<li> $i </li>";
17             echo "</ul>";
18         }
19     }
20 ?>
```

8.2.- Llamado de Los Métodos

El llamado de los métodos se hace de la misma forma que cómo se accede a los atributos públicos. Se instancia el objeto y se accede al nombre del método con el operador "->":

```
1  <?php
2      class Usuario
3  {
4          public $nombre;
5          private $contraseña;
6          private $correo;
7
8          function mostrar_bienvenida()
9          {
10             echo "Bienvenido al sistema";
11         }
12         function agregar_lista($n)
13         {
14             echo "<ul>";
15             for ($i=1;$i<$n;$i++)
16                 echo "<li> $i </li>";
17             echo "</ul>";
18         }
19     }
20     $usu_1 = new Usuario();
21     $usu_1->mostrar_bienvenida();
22     $usu_1->agregar_lista(5);
23 ?>
```

8.3.- Acceso a Los Atributos

Si desde un método se necesita hacer referencia a un atributo (propiedad) de la clase, debemos hacerlo con la variable `$this`, que hace referencia al objeto que está ejecutando el método en el momento, y con el operador "`->`" para acceder a la respectiva propiedad. Si se hace referencia al atributo sin la palabra `$this`, se asume que es una variable local al método. El siguiente es un ejemplo:

```
<?php
1      class Usuario
2      {
3          public $nombre;
4          private $contraseña;
5          private $correo;
6
7          function mostrar_bienvenida()
8          {
9              echo "Bienvenido al sistema " . $this->nombre;
10         }
11        function agregar_lista($n)
12        {
13            echo "<ul>";
14            for ($i=1;$i<$n;$i++)
15                echo "<li> $i </li>";
16            echo "</ul>";
17        }
18    }
19
20    $usu_1 = new Usuario();
21    $usu_1->mostrar_bienvenida();
22    $usu_1->agregar_lista(5);
23 ?>
```

Capítulo 9. MÉTODOS. PARTE 2

9.1.- Getter y Setter

Cuando los atributos de una clase son privados (que es la práctica más recomendable), debe existir una forma de conocer sus valores y de modificarlos. A estos métodos particulares se les denomina getters y setters. Sus nombres provienen de las palabras "get" (obtener) y "set" (establecer). Estos métodos deben ser públicos. El nombre particular generalmente está compuesto del verbo (get o set) seguido del nombre del atributo. Por ejemplo:

```
1 <?php
2     class Usuario
3     {
4         public $nombre;
5         private $contraseña;
6         private $correo;
7
8         function getCorreo()
9         {
10             return $this->correo;
11         }
12         function setCorreo($correo)
13         {
14             $this->correo=$correo;
15         }
16     }
```

La forma de usarlos es la siguiente:

```
1 <?php
2     class Usuario
3     {
4         public $nombre;
5         private $contraseña;
6         private $correo;
7
8         function getCorreo()
9         {
10            return $this->correo;
11        }
12        function setCorreo($correo)
13        {
14            $this->correo=$correo;
15        }
16    }
17 $usu_1 = new Usuario();
18 $usu_1->setCorreo("joselrojas@cadif1.com");
19 echo "El correo es ".$usu_1->getCorreo();
20 ?>
```

9.2.- Constructores

Al definir los atributos de las clases, es posible que existan algunos de ellos que deban ser inicializados en ciertos valores específicos cuando se instancia un objeto. Un constructor es un método que se ejecuta automáticamente al instanciar una clase (crear un objeto), y su utilidad es la de inicializar los atributos que necesiten estar inicializados antes de usar el objeto. Hay dos formas de indicar que un método es el constructor de la clase: colocandole el nombre "__construct" o colocandole el mismo nombre de la clase.

```
1.  <?php
2.      class Usuario
3.  {
4.      public $nombre;
5.      private $contraseña;
6.      private $correo;
7.      private $fecha;
8.
9.      function Usuario()
10.     {
11.         $this->fecha=date("d-m-y");
12.         $this->nombre="sin nombre";
13.     }
14.     function imprimir_Datos()
15.     {
16.         echo "El usuario ".$this->nombre.
17.             " Tiene fecha ".$this->fecha;
18.     }
19. }
20. $usu_1 = new Usuario();
21. $usu_1->imprimir_Datos();
22. ?>
```

9.3.- Métodos Estáticos

Existen un tipo de métodos que se pueden crear dentro de una clase, con la particularidad de que pueden ser llamados sin necesidad de instanciar un objeto de la clase. A estos métodos se les llama métodos estáticos.

El siguiente es un ejemplo:

```
<?php

class Prueba
{
    static function saludar()
    {
        echo "Hola que tal !";
    }
}

// se puede llamar directo con la clas
Prueba::saludar();

// se puede llamar a traves de un objeto
$objeto=new Prueba();
$objeto->saludar();

?>
```

Desde un método estático solo se pueden usar otros métodos estáticos, al tratar de usar métodos de instancia se producirá un error de ejecución:

```
<?php

class Prueba
{
    static function saludar()
    {
        echo "Hola que tal !";
        $this->chao(); // prohibido
    }
    function chao()
    {
        echo "chao";
    }
}

// se puede llamar directo con la clas
Prueba::saludar();

// se puede llamar a traves de un objeto
$objeto=new Prueba();
$objeto->saludar();

?>
```

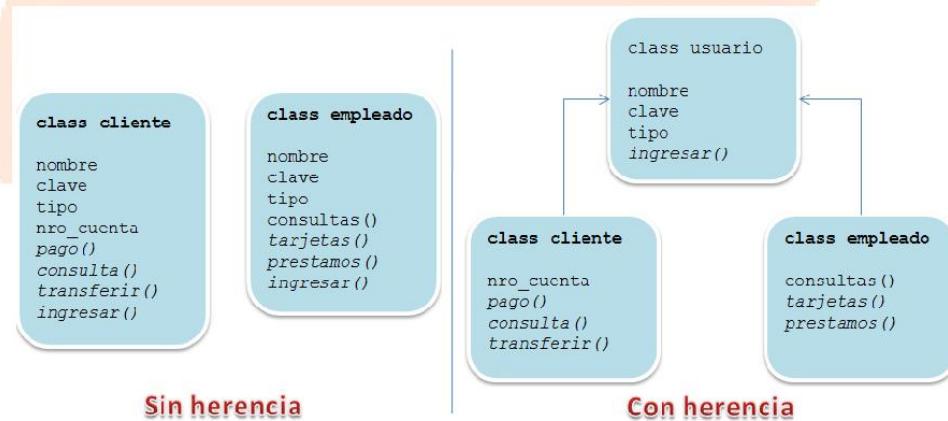
Capítulo 10. HERENCIA

10.1.- Herencia

La herencia en la POO es la relación que se da entre dos clases por la cual una de ellas (a la que llamaremos subclase o clase derivada), además de tener sus propiedades y métodos, tiene a su disposición (hereda) los miembros definidos en la clase padre o superclase.

El propósito final de la herencia es la reutilización de código, si se quiere implementar un nuevo tipo de archivo (por ejemplo un PDF) se podría aprovechar el código de la clase que implementa las características comunes y así ahorrar trabajo.

Por ejemplo, un sistema web bancario, una sitio web mediante el cual los clientes de esta institución pueden realizar diversas operaciones (pagos, consultas y transferencias). Adicionalmente, el personal que trabaja en la institución hace uso del sistema para otras operaciones que no son permitidas a los clientes (consultar cuentas, consulta datos personales de los afiliados, habilitar tarjetas de crédito, entre otras). Pero para ambos casos existe información común y de la misma naturaleza. En ambos ejemplos el ‘usuario’ tiene un nombre, una contraseña y por supuesto debe existir algún identificador que permita diferenciar un usuario de tipo cliente a un usuario de tipo empleado, porque es de esta manera como el sistema decidirá qué funcionalidades debe habilitar.



En PHP 5, para que una clase pueda heredar las características de otra (convirtiéndose en subclase suya) tiene que usar, en su declaración, la palabra reservada extends seguida del identificador de la clase que quiere obtener sus características.

```
<?
class Fichero_PNG extends Fichero {
    private $alto;
    private $ancho;
    private $bits_por_color;

    function bits_por_color() {
        return $this->bits_por_color;
    }

    function alto() {
        return $this->alto;
    }

    function ancho() {
        return $this->ancho;
    }
}

$obj_png = new Fichero_PNG();
echo "La cantidad de bits de la imagen es:" .
    $obj_png->bits_por_color() . "bits<br />";
echo "Las dimensiones del gráfico son: " .
    $obj_png->alto() . "x" . $obj_png->ancho() .
    "<br />";
```

10.2.- Herencia en Constructores

10.3.- Funciones Relacionadas

En este segmento se hará una breve descripción de las funciones que guardan relación con las clases y objetos:

- `class_exists(clase)`: Si clase está definida devuelve TRUE, en caso contrario devuelve FALSE.
- `get_declared_classes()`: Devuelve un arreglo numérico con las clases que están disponibles al momento de ejecutar la función, las nativas de PHP y las definidas por el usuario.
- `Get_class(obj)`: Para el objeto \$obj, devuelve el nombre de la clase de la que es una instancia.
- `Get_parent_class(obj_o_clase)`: Devuelve el nombre de la superclase de la que está derivado el objeto o la clase pasada como parámetro.
- `Get_class_vars(clase)` o `Get_object_vars(obj)`: Devuelve un arreglo asociativo con las propiedades (públicas) de que dispone la clase o el objeto respectivamente.
- `Get_class_methods(clase)`: Devuelve un arreglo asociativo numérico con el nombre de los métodos que dispone la clase.
- Operador `instanceof`: Si la variable \$obj es un ejemplar de la clase `nombre_clase`, devuelve TRUE, en caso contrario devuelve FALSE. Por ejemplo: `$obj instanceof nombre_clase`.
- Constante `__METHOD__`: Cuando se usa desde dentro de un método devuelve el nombre de la clase y el método desde el que se está ejecutando, si se usa desde una función devuelve su nombre.

Capítulo 11. CLASES ABSTRACTAS

11.1.- Clases Abstractas

En ocasiones, en un sistema de herencia como el de la programación orientada a objetos (POO), se tienen que declarar entidades aunque no se puede dar su definición todavía, simplemente se deben definir en general para empezar una jerarquía de clases.

Una clase abstracta define esencialmente el esqueleto básico de un tipo específico de entidad encapsulada; por ejemplo, puede usar una clase abstracta para definir el concepto básico de automóvil como si tuviera dos puertas, un candado y un método que bloquee o desbloquee las puertas. Las clases abstractas no se pueden usar directamente, se deben extender para que la clase descendiente proporcione un complemento completo de métodos.

Las clases abstractas no se pueden instanciar, es decir, no se pueden crear objetos a partir de ellas.

Por ejemplo:

```
abstract class nombre_clase{  
    //propiedades  
    public x;  
    private y;  
  
    //métodos  
    public function __construct(){  
    }  
}
```

11.2.- Métodos Abstractos

Un método abstracto es un método que se define en la clase pero no se implementa, es decir, tiene un encabezado pero no un cuerpo. La implementación del método debe hacerse en las subclases. Para que un método sea abstracto se coloca la palabra "abstract" antes de la definición de la misma. Por ejemplo:

```
abstract function prueba();
```

Si una clase tiene un método abstracto, la clase tiene que ser abstracta.

Por ejemplo:

```
abstract class nombre_clase{  
  
    //propiedades  
    public x;  
    private y;  
  
    //métodos  
    public function __construct(){  
  
    }  
    public abstract function nombre_método();  
}
```

11.3.- Interfaces

Se utilizan cuando se tienen muchos objetos diferentes y que no pertenecen a la misma jerarquía de herencia pero cuyas clases tienen en común un comportamiento. Un ejemplo sencillo sería todos los objetos con los que se comercia en grandes almacenes, los cuales deben contar con la funcionalidad de venderse; una mesa tiene poco en común con un calefactor o unas zapatillas, pero ambos deben implementar una función para poder venderse.

Al implementar una interface se garantiza que ciertos métodos estén disponibles en cualquiera de los objetos que se quieran crear. Son especialmente importantes para la arquitectura de aplicaciones complejas.

Básicamente una interface es una estructura de código que permiten especificar qué métodos deben ser implementados por una clase sin tener que definir cómo estos métodos son manipulados, es decir le dicen a la clase que métodos debe implementar. Contienen métodos vacíos que obligan a una clase a emplearlos, promoviendo así un estándar de desarrollo.

Las interfaces se definen de la misma manera que una clase, aunque reemplazando la palabra reservada `class` por la palabra reservada "interface" y sin que ninguno de sus métodos tenga su contenido definido. Todos los métodos deben ser públicos ya que ésta es la naturaleza de una interfaz.

```
interface encendible{  
    public function encender();  
    public function apagar();  
}
```

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

Luego, las clases que implementen la interfaz serán las encargadas de proporcionar un código a los métodos que contiene la interfaz. Si una clase implementa una interfaz, debe declarar todos los métodos de la interfaz, en caso de no tener código fuente para alguno de esos métodos, deben declararse como abstractos y, por tanto, la clase también tendrá que declararse como abstracta, porque tiene métodos abstractos.

Para implementar una interfaz, se utiliza el operador `implements`. La clase derivada debe tener obligatoriamente definidos todos los métodos que tiene la interface, el no cumplir con esta regla resultará en un error fatal. La clase que implementa una interface puede agregar métodos propios de la clase. Por ejemplo:

```
class Bombillo implements encendible{  
  
    public function encender(){  
        echo "encender el bombillo";  
    }  
    public function apagar(){  
        echo " apagar el bombillo";  
    }  
}
```

Capítulo 13. GENERACIÓN DE ARCHIVOS PDF. PARTE 1

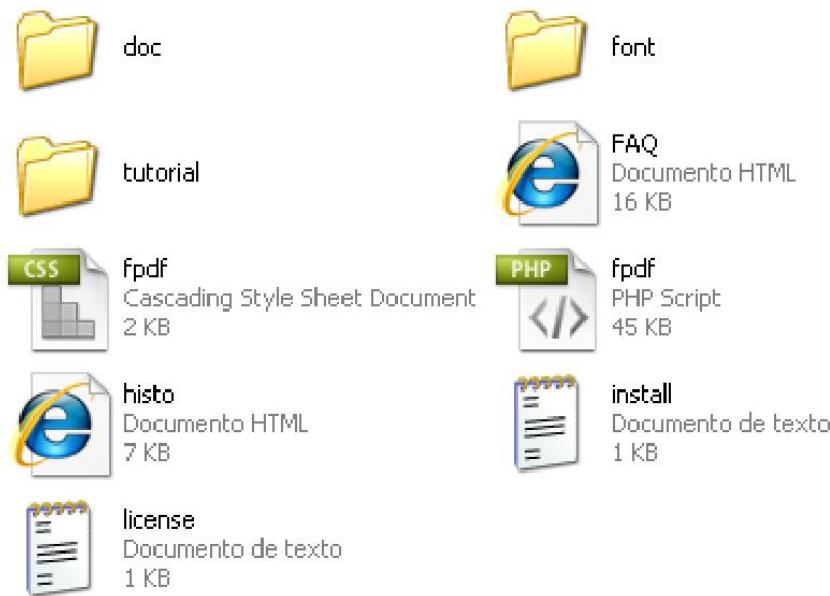
13.1.- ¿Qué es FPDF?

Es frecuente encontrar documentos en formato PDF (Portable Document Format) como resultado de alguna operación sobre un sitio web en particular. El objetivo de este tipo de recursos es generar información que no pueda ser editada por el usuario, que pueda ser trasladada con facilidad y que a su vez pueda tomar datos dinámicamente de distintas fuentes sin afectar lo primero.

Para PHP existen varias opciones a la hora de querer generar contenido en formato pdf, que luego pueda ser descargado por el usuario a su computador. Entre ellos están: HTML2PDF, fpdf, tcpdf, entre otros.



En el curso se usará la librería fpdf, que está desarrollada con el paradigma de la programación orientada a objetos y además es bastante simple de utilizar. Para usar esta librería se debe descargar un archivo comprimido de la página oficial y luego descomprimirlo en la carpeta raíz del proyecto. Al descomprimirlo, se encontrarán los archivos y directorios necesarios para utilizar la librería, entre ellos el archivo fpdf.php, la carpeta font (contiene las fuentes usadas) y la documentación.



13.2.- Creación de un archivo PDF.

Para crear un archivo pdf con la librería fpdf lo primero que se debe hacer es incluir el archivo fpdf.php. Luego se debe instanciar un objeto de la clase FPDF definida dentro de la librería, agregar una página nueva, establecer la fuente y empezar a ejecutar los métodos para agregarle información al documento pdf resultante. Por último se debe ejecutar el método output para cerrar el archivo.

Veamos un ejemplo:

```
<?php  
require('fpdf.php');  
  
$pdf=new FPDF();  
$pdf->AddPage();  
$pdf->SetFont('Arial','B',16);  
$pdf->Cell(40,10,'Hola, Mundo!');  
$pdf->Output();  
?>
```

Esto debe realizarse en un documento php exclusivamente hecho para crear el archivo pdf. Este documento php puede ser llamado con un hipervínculo desde cualquier página del sitio. A continuación, se explora más detalladamente las instrucciones colocadas en este ejemplo:

- La instanciación de la clase: El constructor FPDF() se usa en el ejemplo con sus valores por defecto: las páginas son de tamaño a4 alargado y la unidad de medida es el milímetro. Se podría haber declarado explícitamente con:

```
$pdf=new FPDF("P","mm","A4");
```

Es posible usar el formato apaisado(L), otros formatos de página (como Carta y Legal) y otras unidades de medida (pt, cm, in).

- Agregar una página en blanco: hasta el momento no se ha creado ninguna página, así que se debe añadir una con AddPage(). El origen de coordenadas está en la esquina superior izquierda y la posición actual está por defecto situada a 1 cm de los bordes; los márgenes pueden cambiarse con SetMargins().

- Establecer la fuente: antes de imprimir texto, es obligatorio escoger una fuente con SetFont(), si no, el documento no será válido. Se selecciona Arial en negrita de tamaño 16:

```
$pdf->SetFont("Arial","B",16);
```

Se pudo haber especificado itálica con I, subrayado con U o normal con una cadena vacía (o cualquier combinación de las anteriores). Observe que el tamaño de la fuente se determina en puntos, no en milímetros (ni en cualquier otra unidad).

establecida por el usuario); es la única excepción. Las otras fuentes incorporadas son Times, Courier, Symbol y ZapfDingbats.

Finalmente, el documento se cierra y se envía al navegador con Output(). También se puede guardar en un fichero pasando como parámetro el nombre del archivo.

```
$pdf->output("prueba.pdf");
```

Cuidado: en caso de que el PDF se envíe al navegador, nada más debe enviarse, ni antes ni después (el más mínimo espacio en blanco o retorno de carro también cuenta). Si se envía algún dato antes, obtendrá el mensaje de error: "Some data has already been output, can't send PDF file". Si se envía después, su navegador puede que muestre únicamente una página en blanco.

13.3.- Agregar Contenido a un Documento

Para agregar texto al archivo pdf se debe usar el método cell. Una celda es una superficie rectangular, con borde si se quiere, que contiene texto, ésta se imprime en la posición actual. Al agregar una página nueva el origen de coordenadas está en la esquina superior izquierda y la posición actual está por defecto situada a 1 cm de los bordes. Al agregar la celda se debe especificar sus dimensiones, el texto (centrado o alineado), si se dibujan o no los bordes, y dónde se ubicará la posición actual después de imprimir la celda (a la derecha, debajo o al principio de la siguiente línea). El formato general del método cell es:

```
Cell(float w [,float h [,string txt[,mixed border[,int ln[], string align[,boolean fill[, mixed link]]]]]])
```

La explicación de cada uno de los parámetros es:

w

Ancho de Celda. Si es 0, la celda se extiende hasta la márgen derecha.

h

Alto de celda. Valor por defecto: 0.

txt

cadena a ser impresa. Valor por defecto: cadena vacía.

border

Indica si los bordes deben ser dibujados alrededor de la celda. El valor puede ser un número:

- 0: sin borde
- 1: marco

o una cadena conteniendo alguno o todos de los siguientes caracteres (en cualquier orden):

- L: izquierda
- T: superior
- R: derecha
- B: inferior

ln

Indica donde la posición actual debería ir antes de invocar. Los valores posibles son:

- 0: a la derecha
- 1: al comienzo de la siguiente linea
- 2: debajo

Poner 1 es equivalente a poner 0 y llamar justo después Ln(). Valor por defecto: 0.

align

Permite centrar o alinear el texto. Los posibles valores son:

- L o una cadena vacía: alineación izquierda (valor por defecto)
- C: centro
- R: alineación derecha

fill

Indica si el fondo de la celda debe ser dibujada (true) o transparente (false). Valor por defecto: false.

link

URL o identificador retornado por AddLink()

Luego de agregar la celda o en cualquier momento se puede agregar un salto de línea en el documento con el método ln(). También se puede ubicar la posición actual en el documento a un lugar arbitrario usando el método setxy().

Por ejemplo:

Otra forma de agregar texto al documento es utilizando el método multicell. Este método permite imprimir texto con saltos de línea. Estos pueden ser automáticos (tan pronto como el texto alcanza el borde derecho de la celda) o explícito (vía el carácter \n). Tantas celdas como sean necesarias son creadas, uno debajo de otra. El texto puede ser alineado, centrado o justificado. El bloque de celda puede ser enmarcado y el fondo impreso. Los parámetros de multicell son:

```
MultiCell(float w, float h, string txt [, mixed border [, string align [, boolean fill]]])
```



Capítulo 14. GENERACIÓN DE ARCHIVOS PDF. PARTE 2

14.1.- Agregar Imágenes

Para agregar imágenes a un archivo pdf se debe usar el método `image()`. Las dimensiones pueden establecerse de diferentes maneras:

- * Mediante la especificación explícita de ancho y alto (en unidades definidas por el usuario)
- * Mediante la indicación de una sola de las dimensiones, la otra se calculará automáticamente para mantener la proporción original
- * Sin indicar ninguna dimensión explícita. En este caso, la imagen se imprime a 72 puntos por pulgada

Los formatos admitidos son JPEG, PNG y GIF. La extensión GD es necesaria para GIF.

La sintaxis general de éste método es:

```
Image(string file [, float x [, float y [, float w [, float h [, string type [, mixed link]]]]]])
```

Los parámetros del método `image` son:

file

Nombre del fichero que contiene la imagen.

x

Abscisa de la esquina superior izquierda. Si no se especifica o es igual a null, se utilizará la abscisa actual.

y

Ordenada de la esquina superior izquierda. Si no se especifica o es igual a null, se utilizará la ordenada actual, además, un salto de página es invocado primero si es necesario (en caso de que esté habilitado el salto de página automático) y, después de la llamada, la ordenada actual se mueve a la parte inferior de la imagen.

w

Ancho de la imagen en la página. Si no se especifica o es cero, se calcula automáticamente.

h

Alto de la imagen en la página. Si no se especifica o es cero, se calcula automáticamente.

type

Formato de la imagen. Los posibles valores son (indiferente a mayúsculas): JPG, JPEG, PNG y GIF. Si no se especifica, el tipo se deduce de la extensión del fichero.

14.2.- Encabezado y Pie de Página

La clase FPDF tiene los métodos Header() y Footer() que agregan un encabezado a las páginas y un pie a las páginas automáticamente. Estos métodos en la clase por defecto no hacen nada, es por ello que se debe crear una clase que herede de la clase FPDF y sobreescibir estos 2 métodos para así poder agregar pies de páginas y/o encabezados de página a los documentos pdf. A continuación un ejemplo de como crear una clase que herede de la clase fpdf:

```
<?php
require('fpdf.php');

class PDF extends FPDF
{
//Cabecera de página
function Header()
{
    //Logo
    $this->Image('logo_pb.png',10,8,33);
    //Arial bold 15
    $this->SetFont('Arial','B',15);
    //Movernos a la derecha
    $this->Cell(80);
    //Título
    $this->Cell(30,10,'Title',1,0,'C');
    //Salto de línea
    $this->Ln(20);
}

//Pie de página
function Footer()
{
    //Posición: a 1,5 cm del final
    $this->SetY(-15);
    //Arial italic 8
    $this->SetFont('Arial','I',8);
    //Número de página
    $this->Cell(0,10,'Page '.$this->PageNo().'/({nb})',0,0,'C');
}
}
```

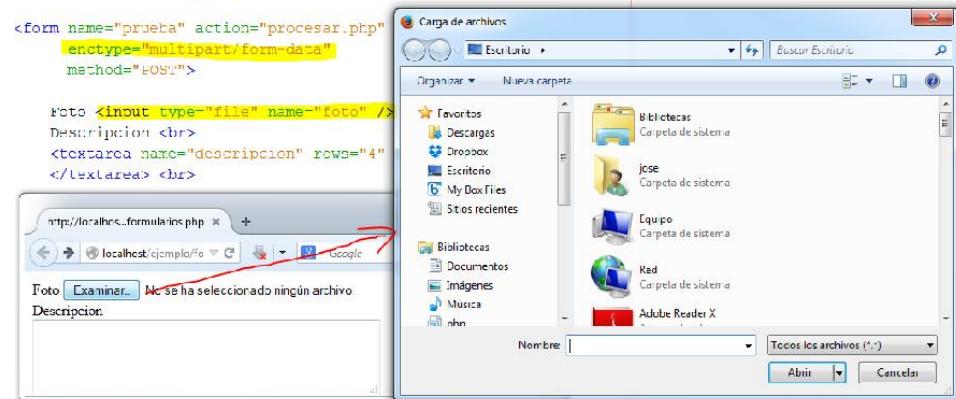
Luego debemos instanciar nuestra clase. Al agregar nuevas páginas automáticamente serán llamados los métodos header() y footer().

```
//Creación del objeto de la clase heredada
$pdf=new PDF();
$pdf->AliasNbPages();
$pdf->AddPage();
$pdf->SetFont('Times','',12);
for($i=1;$i<=40;$i++)
    $pdf->Cell(0,10,'Imprimiendo línea número '.$i,0,1);
$pdf->Output();
```

Capítulo 15. ENVÍO DE ARCHIVOS

15.1.- Campo de Archivos

Los campos de archivo se utilizan cuando necesitamos proveer al usuario una forma de que pueda seleccionar archivos para enviar, por ejemplo, archivos adjuntos en un correo, fotografías en un registro personal, foto de un producto, etc. La etiqueta es `input type="file"`. Para que los archivos sean subidos el "action" del formulario debe ser "post" y adicionalmente se le debe agregar una propiedad llamada "enctype" con el valor "multipart/form-data". Veamos el ejemplo:



15.2.- Procesar Archivos Subidos

Cuando un formulario posee etiquetas del tipo `input type="file"` se deben procesar los archivos subidos al servidor. Lo primero que se debe saber es que los archivos se suben al servidor web con un nombre temporal, y ubicados en una carpeta temporal. Luego de que el archivo esta subido, debemos moverlo a una carpeta dentro de nuestra carpeta raíz (la ubicación dependerá de lo que queremos hacer).

Los datos tipo archivo se manejan con el arreglo asociativo `_FILES` (anteriormente `HTTP_POST_FILES`), que contendrá todos los archivos que se hayan subido en el formulario (si es que hay varias etiquetas tipo file en el mismo formulario), usando el "name" asociado a la etiqueta input, por ejemplo, `$_FILES["mi_archivo"]`.

Luego se accede a las propiedades del archivo como si fuera una matriz, de la forma `$_FILES["mi_archivo"]["propiedad"]`. Las propiedades que posee el archivo son:

- `$_FILES["mi_archivo"]["name"]` : nombre original del archivo al seleccionarlo localmente.
- `$_FILES["mi_archivo"]["size"]`: tamaño en bytes del archivo subido.
- `$_FILES["mi_archivo"]["type"]` : tipo de archivo seleccionado.
- `$_FILES["mi_archivo"]["tmp_name"]`: nombre temporal que le fue asignado al subirlo.
- `$_FILES["mi_archivo"]["error"]`: código de error al subir archivo.

En el código php se debe verificar si el archivo fue subido exitosamente (`is_uploaded_file`) y luego moverlo con la función `copy`. Veamos un ejemplo:

```
<?php
    // se verifica si el archivo fue subido
    if (is_uploaded_file($_FILES['foto']['tmp_name']))
    {
        // se puede mantener el nombre original del archivo o se puede
        // usar otro
        $nombre=$_FILES['foto']['name'];
        // la ruta donde se van a almacenar los archivos subidos
        $ruta="ruta_deseada/";
        // se copia el archivo de la carpeta temporal a la ruta
        // establecida previamente y con el nombre que se decida usar
        copy($_FILES['foto']['tmp_name'],$ruta.$nombre );
    }
?>
<form name="prueba" action="procesar.php"
      enctype="multipart/form-data"
      method="POST">

    Foto <input type="file" name="foto" /> <br>
```

El código de error se puede encontrar en el segmento `error` del array del archivo que PHP crea durante la subida del archivo. Los valores que puede tomar esta propiedad son:

- `UPLOAD_ERR_OK`: Valor 0. No hay error, archivo subido con éxito.
- `UPLOAD_ERR_INI_SIZE`: Valor 1. El archivo subido excede la directiva `upload_max_filesize` en `php.ini`.
- `UPLOAD_ERR_FORM_SIZE`: Valor 2. El archivo subido excede la directiva `MAX_FILE_SIZE` que fue especificada en el formulario HTML.

- UPLOAD_ERR_PARTIAL: Valor 3. El archivo subido fue sólo parcialmente cargado.
- UPLOAD_ERR_NO_FILE: Valor 4. Ningún archivo fue subido.
- UPLOAD_ERR_NO_TMP_DIR: Valor 6. Falta la carpeta temporal.
- UPLOAD_ERR_CANT_WRITE: Valor 7. No se pudo escribir el archivo en el disco.
- UPLOAD_ERR_EXTENSION: Valor 8. Una extensión de PHP detuvo la carga de archivos.

15.3.- Configuración de Php

Los valores que se asignan en el php.ini inciden directamente en el comportamiento al subir archivos. Entre las propiedades de configuración que se establece en el php.ini están: si permite subir archivos (file_uploads), carpeta temporal donde se suben los archivos (upload_tmp_dir), el tamaño máximo de los archivos (upload_max_filesize), la cantidad máxima de archivos que se pueden subir (max_file_uploads). Veamos un ejemplo de una configuración:

```
;;;;;;;;;;;
; File Uploads ;
;;;;;;;;;;;

; Whether to allow HTTP file uploads.
; http://php.net/file-uploads
file_uploads = On

; Temporary directory for HTTP uploaded files (will use system default if not
; specified).
; http://php.net/upload-tmp-dir
upload_tmp_dir = "${path}\tmp\"

; Maximum allowed size for uploaded files.
; http://php.net/upload-max-filesize
upload_max_filesize = 2M

; Maximum number of files that can be uploaded via a single request
max_file_uploads = 20
```