PHP desde cero

Alfredo Cádiz R.

30 de abril de 2005

1. ¿Que es PHP?

PHP, acrónimo de "PHP: Hypertext Preprocessor", es un lenguaje interpretado de alto nivel embebido en páginas HTML. La mayoría de su sintaxis es similar a C, Java y Perl, con solamente un par de caracteristicas PHP específicas. La meta de este lenguaje es permitir escribir a los creadores de páginas web, páginas dinámicas de una manera rápida y fácil. A diferencia de Javascript o VBscript, PHP corre por el lado del servidor, por ende, se desliga al navegador de tener que soportar éste lenguaje.

1.1. PHP vs HTML

Realmente PHP y HTML no compiten, sino que se complementan. HTML es el lenguaje utilizado para mostrar todo tipo de información a través de un navegador, mientras que PHP permite generar contenido dinámico en formato HTML. Esto da pié a poder crear aplicaciones de gran envergadura en línea.

1.2. Primer programa en PHP

Las aplicaciones escritas en PHP usualmente se corren a través del acceso de un sitio web conteniendo el archivo, el cual se ejecuta y se muestra en el navegador. Además es posible ejecutarlo como un script de unix, pero ésto no se tratará en este documento.

Para poder ejecutar cualquier aplicación PHP en un servidor web es necesario verificar lo siguiente:

- La aplicación debe localizarse en el directorio web personal, usualmente es */public_www* o */public_html*, en caso contrario, hay que contactarse con el administrador del servidor para que se indique el directorio a utilizar.
- Los archivos tanto PHP como HTML deben poseer permisos de lectura y ejecucion para cualquier usuario. En unix esto se logra con el comando *chmod*[1].
- Usualmente los programas en PHP solo son interpretados por el servidor como tales cuando son guardados con la extensión *.php*.
- El código en PHP dentro de un archivo solo será interpretado si esta entre los tags *<?php* y *? >*

`Ejemplo: Hola Mundo`

```
<?php
echo "Hola mundo";
?>
```

[1] Mas información en unix: man chmod

2. Variables

- En PHP las variables no son declaradas explicitamente. Es decir, se utilizan en cuando se las requiera.
- Toda variable en PHP comienza con el signo $ y luego viene su nombre. Ej: $i, $rut
- El tipo de la variable se da por el contexto en el que se esté usando.

  Ejemplo:
  $numero = "1"; //Aqui se asigna el string "1" a la variable $numero
  print($numero); //Se imprime la variable
  $veces = $numero+1; // Aqui se comporta como entero

  $veces tiene el valor 2

3. Arreglos

- Al igual que las variables normales, los arreglos no son necesario declararlos.
- Además pueden guardarse cualquier tipo de datos en un mismo arreglo. Ejemplo:

  $datos[0] = "Pedro";
  $datos[1] = 14375524;

4. Strings - Cadenas

- Las cadenas se definen como cualquier variable, cuidando de que estén entre comillas. Éstas son muy poderosas ya que su flexibilidad de uso y gran cantidad de funciones asociadas permiten manipular texto de forma facil y rápida.
  Ejemplo: $cadena = "Hola Mundo";

- Si la cadena comienza con un número y luego ésta es operada como número, se considerá solo aquel que esté al comienzo y el resto de la cadena será ignorada.
  Ejemplo:
  $cadena1 = "123Hola";
  $cadena2 = "123Hola345";
  $cadena3 = "Hola123";
  $entero1 = $cadena1 + 2; //$entero1 tiene el valor 125

$entero2 = $cadena1 + 2; //$entero2 tiene el valor 125
$entero3 = $cadena1 + 2; //$entero3 tiene el valor 2

- La concatenación de cadenas es a través del operador ".".
  Ejemplo:
  $rut = 134567345;
  $verificador = 0;
  $salida = $rut."−".$verificador;
  print($salida); // Imprime 134567345-0

- Dentro de una asignación de cadena es posible incluir variables, las cuales serán evaluadas, ésto puede ser util si se poseen asignaciones muy extensas de strings y asi se evita la confusión a causa de muchos operadores de concatenación ("."), aunque tambien puede ser un problema si es que se deseea mostrar nombres de variables y no su contenido. En tal caso hay 2 soluciones:

  - Forzando a no evaluar ninguna variable utilizando comillas simples en lugar de comillas dobles al asignar una cadena.
  - Si se desea mezclar la evaluación de variables con la impresión explítica de ellos, es posible escapar el simbolo $ con \$.

  Ejemplo:
  $hola ="chao";
  $frase1 = "$hola, ¿como estas?"
  print($frase1); // Imprime "chao, ¿como estas?"
  $frase2 = "\$hola, ¿como estas?"
  print($frase2); // Imprime "$hola, ¿como estas?"
  $frase3 = '$hola, ¿como estas?'
  print($frase3); // Imprime "$hola, ¿como estas?"

5. Operadores

5.1. Aritméticos

Operador	Nombre
+	suma
−	resta
*	multiplicación
/	división
%	módulo
++	suma 1
−−	resta 1

5.2. Comparación

Operador	Nombre
==	igual
! =	distinto
<	menor
>	mayor
<=	menor igual
>=	mayor igual

5.3. Lógicos

Operador	Nombre
&&	Y lógico
and	Y lógico
||	O lógico
or	O lógico
!	Negación

6. Control de flujo

PHP contiene las instrucciones usuales para ejecutar código sometido a condiciones y ciclos.

6.1. If-Else

```
if ($i<$j)
{
   print($i es menor que $j);
}
else
{
   print($i no es menor que $j);
}
```

6.2. While

```
$i=0;
while ($i<10)
{
   echo "El valor de i es ", $i,"<br>";
   $i++;
}
```

6.3. For

```
for($i=0 ; $i<10 ; $i++)
{
   echo "El valor de i es ", $i,"<br>";
}
```

7. Salida

Existen muchas maneras de imprimir usando PHP. Aqui se muestran las mas comunes.

7.1. print - echo

Las instrucciones print y echo permiten imprimir una cadena o una union de éstas. Se pueden usar indistintamente.

```
print($cadena);
echo($cadena1.$cadena2);
```

7.2. printf

- La función printf permite imprimir una cadena formateada al estilo printf de C.
- El formato es printf(string, variables);

  string: Es la cadena que se imprimirá, en lugar de colocar las variables para que sean evaluadas, se agregan símbolos especiales indicando el tipo como la variable se desea que sea evaluada.

Carácter	Tipo de variable
%s	Cadena de carácteres
%d	número sin decimales
%f	número con decimales
%c	caracter ASCII
%x	número entero mostrado en hexadecimal

  variables: A continuacion de la cadena, se colocan en orden las variables que corresponden a ser reemplazadas en *string*.

```
Ejemplo:
$rut = 14356876;
$dv = 9;
printf("El rut del alumno es %d-%d.", $rut, $dv);

Se imprime "El rut del alumno es 14356876-9"
```

8. Arreglos asociativos

- Los arreglos asociativos funcionan de manera similar que un arreglo normal, con la diferencia que su llave puede ser cualquier cadena.
- Dentro de arreglos asociativos es posible guardar otros arreglos asociativos.
- Pueden ser asignados asignando un valor a cada llave o a través de la funcion *array()*.

```
Ejemplo:

//Asignando los valores de a uno
$persona['nombres'] = "Juan Alejandro";
$persona['apellidos'] = "Gonzales Castro";
$persona['rut'] = 14777555;
$persona['dv'] = 8;

//Utilizando array();
$persona = array("nombres" => "Juan Alejandro",
                 "apellidos" => "Gonzales Castro".
```

```
"rut" => 14777555,
"dv" => 8);
```

- Una forma efectiva de mostrar el contenido de un arreglo asociativo, por ejemplo para debug, es utilizando la función *print_r($arreglo)*.

  Ejemplo:

  ```
  echo "<pre>"; //Este tag es para que se muestre el contenido tabulado
  print_r(\$persona);
  echo "</pre>";
  ```

8.1. Foreach

Como los arreglos asociativos no poseen un orden para sus llaves como es en el caso de los arreglos normales, si se desean recorrer linealmente se utiliza la instrucción *foreach()*.

```
$persona['Juan Perez'] = "Alumno";
$persona['Alejandro Flores'] = "Profesor";
$persona['Carmen Gomez'] = "Profesor";

foreach($persona as $key => $dato){
printf("%s es un %s<br>",$key, $dato);
}

El resultado sera:
Juan Perez es un Alumno
Alejandro Flores es un Profesor
Carmen Gomez es un Profesor
```

9. Funciones

- En PHP es posible definir funciones para agrupar instrucciones de uso común sin necesidad de escribirlas cada vez que se utilizen.
- Pueden ser declaradas en cualquier parte del código, aunque es recomendable hacerlo en el inicio del archivo o al final, por orden.
- El formato para declarar funciones es el siguiente:

  ```
  function nombre(parametro1, parametro2, ...){
  instruccion1;
  instruccion2;
  instruccion3;
  ```

```
...
instruccion-N;

return valor_de_retorno;
}
```

- Los parámetros son pasados como cópias de éstos, si se desean pasar referencias se debe utilizar &$variable.
- Se puede retornar un valor, una variable o no retornar (equivalente a una función void).

```
function area_cuadrado($alto, $ancho){
$resultado = $alto*$ancho;
return $resultado;
}
$i = 6;
$j = 7;

print(area_cuadrado(4,5));
print(area_cuadrado($i, $j));
```

10. Incluir archivos externos

- Dada una cantidad grande de funciones, es posible agruparlas en un archivo distinto de donde se esta procesando la información esto permite compartir funciones entre distintas aplicaciones PHP sin necesidad de copiarlas en cada una de ellas.
- Además de contener funciones, pueden contener variables asignadas, de este modo de pueden crear archivos de configuración para una aplicación que utilize varias páginas PHP distintas y se deseen centralizar los parámetros globales.
- Las funciones usuales para llamar a archivos externos son *include*, *require*, *include_once* y *require_once*.
  La diferencia entre *include* y *require* es que el primero, al fallar el include solo dará una advertencia, por otro lado, si *require* falla, se detendrá la ejecución del código con un error fatal.
- Las funciones *include_once* y *require_once* funcionan de manera similar a sus equivalentes en nombre, solo que si un archivo ya fue invocado una vez, no volverá a hacerse.
- Usualmente estas funciones se colocan al inicio de archivo PHP, pero pueden posicionarse en cualquier parte del código.

11. Manejo de Archivos

- PHP provee de funciones para poder manipular archivos locales. Para abrirlos se utiliza *fopen()*.
- A continuacion se presentan las formas mas usuales de uso de esta función. Para mas funcionalidades revisar el manual en línea de PHP.

11.1. Leer archivos

```
$fp = fopen("archivo.dat", "r");
while(!feof($fp)){
   $line=fgets($fp, 512);
   print($line);
}
fclose($fp);
```

11.2. Escribir archivos

```
$fp = fopen("archivo.dat", "w");
fputs($fp, "$nombre $apellido\n");
fclose($fp);
```

12. Manejo de formularios

- Una de las formas de enviar información a través de una página web es a través de los formularios. PHP es capaz de recibir los datos enviados por este medio para manipularlos como sea conveniente.
- Los formularios HTML deben ser declarados cuidando de especificar el atributo *name* para que pueda ser recibido por el servidor.
- Existen 2 métodos para enviar información: GET y POST.

  GET: Envia la información al servidor a través de una codificación de los dátos en la URL. Este método permite que el usuario vea todo lo que se esta enviando con solo inpeccionar la barra de dirección.

  POST: Envia la información al servidor de forma interna, de tal manera que no es posible revisar la información que se ha enviado al servidor.

```
Ejemplo de un formulario típico:

<html>
<body>
<form action="#" method ="post" name="form">
Nombre: <input type="text" name="nombre"> <br>
Apellido: <input type="text" name="apellido"> <br>
```

```
Rut: <input type="text" name="rut"><br>
</form>
</body>
</html>
```

- En PHP los datos enviados a través de un formulario son guardados en los arreglos asociativos $_GET y $_POST, para cada método de los mencionados anteriormente.

```
Ejemplo utilizando el formulario anterior:

<?php
print("Su nombre es $_POST['nombre'] $_POST['apellido']. <br>");
print("Su rut es: $_POST['rut']");
?>
%$
```

13. Buenas costumbres de programación en PHP

PHP es un lenguaje extremadamente flexible con respecto donde se puede agregar código PHP, declarar funciones y asignar variables. Esto da pie a que al implementar aplicaciones ya medianamente grandes, el código resulte inmantenible dada la mezcla de líneas en PHP con HTML y la redundancia de funciones y parámetros entre cada página. Con el fín de lograr aplicaciones mantenibles es necesario fijar algunas normas de programación.

13.1. Separación por carpetas

Es muy recomendable separar los archivos por carpetas, usualmente se separan los archivos por funciones, imagenes, plantillas y la aplicación en si, con la siguiente estructura:

include :es donde se deben alojar los archivos que contengan funciones encapsuladas.

imagenes :todas las imagenes utilizadas en el servidor.

templates :aqui corresponden los archivos de plantillas.

. : los archivos de aplicación deben ir en la raiz del directorio

13.2. Modularización

Con el fín de evitar la redundancia de funciones y parámetros basta con separarlos de la aplicación, en este caso, de cada página PHP involucrada en la aplicación. Luego, para incluirlos en cada archivo PHP se utilizará el comando *require_once()*[2].

[2]También se puede utilizar include_once(), include() o require(), pero se recomienda el indicado ya que es más seguro.

13.2.1. Archivo de configuración (config.php)

Cualquier parámetro global debe guardarse aqui, tal como el título de la aplicación, los datos para ingresar a una base de datos, usuarios especiales, etc.

```
Ejemplo de config.php
<?php
$nombre_aplicacion = "Sistema de postulación";
$rut_administrador = "12456329";
?>
```

13.2.2. Archivo de funciones

Todas las funciones que se puedan utilizar en una o más páginas de la aplicación deben encontrarse en un archivo especializado. Para este caso, es posible que además de funciones normales, también se encuentren algunas que trabajen con bases de datos o que tengan otra clasificación especial, por lo que se recomienda además subdividir por el tipo de función que se trabaja, asi se simplifica el trabajo de aislar un grupo de funciones de un mismo tipo para su posible modificación. Usualmente se utilizan 2 grupos de funciones, las normales (funciones.php) y las relacionadas con bases de datos(funciones_bd.php). Los nombres indicados solo son recomendaciones, puede asignarse cualquier otro que quede más claro para indicar su tipo.

```
Ejemplo de funciones.php
<?php
function suma($n, $m){
return $n+$m;
}
?>

Ejemplo, incluir los archivos en la aplicación
<?php
require_once("config.php");
require_once("include/funciones.php");
require_once("include/funciones_bd.php");

... resto de la aplicación ...
?>
```

13.3. Uso de plantillas (Xipe)

El uso de plantillas es muy importante para que el software escrito en PHP sea mantenible, ya que separa la parte lógica (PHP) del diseño (HTML), permitiendo el trabajo conjunto de programadores y diseñadores sin interferir en las áreas que no les corresponden.

Existen muchas herramientas para procesar plantillas, en este caso se mostrarán las plantillas *Xipe*.

13.3.1. Instalación

Para efectos del curso, se utilizará una modificación a la distribución de Xipe oficial para simplificar la instalacion e importación, pero el modo de uso es exactamente el mismo.

- Para instalar Xipe, basta con descomprimir la carpeta *Xipe_Template* en el directorio raiz de la aplicación a desarrollar.
- Luego, se debe crear un directorio *tmp* dentro de *templates*, con permisos totales para cualquier usuario[3].

13.3.2. Uso

Para utilizar *Xipe* se deben crear 2 archivos por cada página que se muestre en el navegador:

archivo.php : donde se encuentran todas las instrucciones PHP. Su estructura para incluir las plantillas es la siguiente:

```
<?php
require_once('Xipe_Template/Xipe.php');
$options = array(
'templateDir' => "templates",
'compileDir' => 'tmp');
$t = new HTML_Template_Xipe($options);

... aplicación ...

$t->compile("index.html");
include($t->getCompiledTemplate());
?>
%$
```

archivo.html : aqui se encuentra el diseño de como se verá la página en el navegador, se trata como HTML normal a excepción que se agregan campos del tipo $variable en donde $variable, declarada en *archivo.php* sera reemplazada por el valor que tenía al momento de ejecutar *include($t->getCompiledTemplate())*.
Además, es posible incluir instrucciones de iteración para poder mostrar información que necesite de ellas para ser mostrada, como los arreglos.

[3] chmod 777 tmp

A. Ejemplo de uso de Xipe

Archivo: index.php

```
<?
require_once('Xipe_Template/Xipe.php');
$options = array(
// the root-directory where to find the templates
'templateDir' => "templates",
// the directory where to store the compiled template
'compileDir' => 'tmp');

$t = new HTML_Template_Xipe($options);

$mensaje = "Hola Mundo!";
$arr = array("Nombre" => "Juan", "Apellido" => "Perez");

$t->compile("index.html");
include($t->getCompiledTemplate());
?>
```

Archivo: templates/index.html

```
<html>
<head></head>
<body>
<h1>{$mensaje}</h1>

<hr></hr>

{foreach($arr as $llave => $dato):}
{$llave}: {$dato}
<hr>
{endforeach}

</body>
</html>
```