PHP básico

Realizado por A.Garay (dpto. informática)

Resumen de contenidos

- Conceptos genéricos
- Variables, constantes y palabras reservadas
- Operadores
- Sentencias de control
- Entrada y salida estándar
- Funciones
- Arrays
- Manejo de cadenas
- Manejo de fechas
- Niveles de mensajes de error
- Inclusión de ficheros

Conceptos e instalación

- PHP es un lenguaje interpretado a diferencia de otros lenguajes compilados como C o JAVA.
- Eso significa que los programas en PHP se ejecutan a partir del código fuente directamente.
- Para ejecutar un programa en PHP se necesita un intérprete. Normalmente éste se instalará en un servidor web, pero podría instalarse uno localmente en un entorno Linux (debian) ejecutando “apt-get install php5”

PHP como lenguaje embebido

- PHP es un lenguaje pensado para ser “incrustado” en un fichero entre código HTML para ser posteriormente interpretado en un servidor web y no en un intérprete de comandos como JAVA, C ó C++.
- No obstante para estos primeros ejemplos y comprender su sintaxis, lo haremos de esta última forma.

Historia

- PHP inicialmente se llamaba PHP/FI (**P**ersonal **H**ome **P**age Form Interpreter)
- Actualmente es el acrónimo recursivo de **P**HP **H**ypertext **P**reprocessor)
- Desarrollado inicialmente en 1995 por Rasmus Lerdorf
- Actualmente en manos de “The PHP group” y licenciado como software libre (PHPv3_01)

Código fuente en PHP

- El código fuente en PHP debe guardarse en archivos con extensión “.php”
- El contenido del código debe comenzar con “**<?php**” y terminar con “**?>**”
  - Alternativamente se podrían utilizar las marcas:
    - <? … ?>
    - <script language="php"> …… </script>
    - <% …… %> (a partir de PHP 3.0.4)

Hola mundo en PHP

```
<?php
    echo “Hola mundo”;
?>
```

- Guardaremos este código en un archivo “holaMundo.php”
- Para ejecutar bastaría ejecutar en un terminal “php holaMundo.php” siempre y cuando tengamos un intérprete de PHP local instalado.
- Como se puede observar, “echo” es la instrucción básica de salida estándar.
- Como en otros lenguajes de programación, todas las instrucciones terminan con punticoma.

Variables

- Los nombres de variables deben comenzar por el signo dólar “$”
- Son sensibles a mayúsculas y después del “$” tendrán una letra o guión bajo, seguido de cualquier cantidad de letras, números y guiones bajos.
- PHP es un lenguaje débilmente tipado
- No hace falta declararlas, ni indicar el tipo de datos que van a contener. La primera vez que se les asigna un valor el intérprete deduce su tipo, y si se quiere cambiar el tipo de la variable después, también se puede hacer 😮.
- Si se intenta mostrar u operar con una variable a la que no se le ha asignado un valor previamente, el intérprete muestra un error.

Variables (ejemplos)

```
$nombre1 = ”Pepe”;   // variable tipo cadena
$nombre2 = ’Juan’;   // variable tipo cadena
$numero = 1;         // variable tipo entero
$_otroNumero = 1.3;  // variable tipo decimal
$terminado = true;   // variable tipo boolean
```

Constantes

- Se definen con “const” (PHP>=5.3) o la función define.
  - const pi = 3.141592;
  - define (“pi” , 3.141592);
- Su identificador no puede comenzar por el signo dólar

Palabras reservadas

Las palabras reservadas o construcciones del lenguaje PHP, no deben confundirse con funciones y tienen las siguientes características:

- No se pueden usar como constantes, nombres de clase, nombres de funciones o de métodos
- Se pueden usar como nombres de variables, pero no se recomienda
- Con las construcciones del lenguaje, en general, no se requiere el uso de paréntesis
- Las funciones se simplifican hasta obtener construcciones del lenguaje

Listado de palabras reservadas:

__halt_compiler(), abstract, and, array(), ask, break, callable, case, catch, class, clone, const, continue, declare, default, die(), do, echo, else, elseif, empty(), enddeclare, endfor, endforeach, endif, endswitch, endwhile, eval(), exit(), extends, final, finally, for, foreach, function, global, goto, if, implements, include, include_once, instanceof, insteadof, interface, isset(), list(), namespace, new, or, print, private, protected, public, require, require_once, return, static, switch, throw, trait, try, unset(), use, var, while, xor, yield

Operadores básicos

- Aritméticos: + - * / % ** (PHP >= 5.6)
- Incremento: ++ -- (con pre y post incremento)
- Concatenación de cadenas: .
- Asignación: =
- Acumuladores: += *= /= %= .= **=
- Comparación: == != < > <= >=
- Lógicos: && ó and || ó or ! ó not xor
  - Funcionan en cortocircuito
- Ternario: <cond> ? <valorSiCierto> : <valorSiFalso>

La división entera

- En PHP no existe el operador “/” para la división entera
  - echo 99/10; // Muestra 9.9
- La forma de obtener la parte entera es haciendo un “casting” a “int” o “integer” o utilizar las funciones “round”, “ceil” o “floor”
  - echo (int)(99/10); // Muestra 9
  - echo round(99/10); // Muestra 10

Operadores (particularidades)

- En PHP se pueden comparar directamente cadenas (con criterio lexicográfico)
  - “a” < “b” devolvería true
  - “p” < “pepe” devolvería true
  - “p” < “Q” devuelve false porque el código ASCII de cualquier mayúscula es menor que el de cualquier minúscula.
- Aunque es muy desaconsejable, en PHP los números y cadenas se pueden involucrar en expresiones lógicas, teniendo en cuenta que 0 es false y cualquier otro número true, y que la cadena vacía es false, y cualquier otra cadena true.
  - `$a = 3; while ($a) {echo $a--;}` //Mostraría 3 2 1
- Existen los operadores **===** y **!==** que evalúan respectivamente la IGUALDAD en valor y tipo o la DIFERENCIA en valor o en tipo

Precedencia de operadores

- Como en otros lenguajes de programación, el operador con más preferencia es el paréntesis y su contenido es lo primero que se evalúa.
- Si hay varios paréntesis anidados se evalúan de dentro hacia afuera y de izquierda a derecha.
- Para el resto de las operaciones la asociatividad está indicada en la columna correspondiente (D=derecha, I=izquierda)
  - Ej: 8 ** 7 ** 2 = 8 ** (7 ** 2)
  - Ej: 8 + 7 + 2 = (8 +7) + 2

Asociat.	*OPERACIÓN*
D	**
D	++ --
D	!
I	* / %
I	+ - .
	= += -= *= **= /= .= %=
	== !=
I	&&
I	||
I	? :
D	= += -= *= **= /= .= %=
I	*and*
I	*xor*
I	*or*

Variables por referencia

```
$a = 1;
$b = 3;
$aCopiaValor = $a
$bCopiaReferencia = &$b;
$aCopiaValor ++;
$bCopiaReferencia ++;
echo $a.” ”.$aCopiaValor.” ”.$b.” ”.$bCopiaReferencia;

// Mostraría 1 2 4 4
```

Sentencias de control

- if (<cond>) {…} elseif (<cond2>) {...} ... else {…}
  - elseif y else son optativas
- switch (variable) {case val1:insts1;...;break; case val2:insts2;...;break; …..;break; default:instDefault }
- while (<cond>) {...}
- do {...} while (<cond>)
- for (<inst_ini>; <cond>; <inst_finBucle>) {...}

Salida estándar

Código	Salida
echo “Hola”;	Hola
echo ‘Adiós’;	Adiós
echo ‘Hola’,’ y’,’ adiós’;	Hola y adiós
$edad=18; echo “Tengo $edad años”;	Tengo 18 años
$edad=18; echo ‘Tengo $edad años’;	Tengo $edad años
echo “Hola ‘amigo’, qué tal”;	Hola ‘amigo’, qué tal
echo ‘Hola “amigo”, qué tal’;	Hola “amigo”, qué tal
echo “Hola \“amigo\”, qué tal”;	Hola “amigo”, qué tal
echo ‘Hola \’amigo\’, qué tal’;	Hola ‘amigo’, qué tal

Entrada estándar

```
fscanf(STDIN, "%d\n", $numero);
```

%d	número entero
%f	número con decimales
%s	cadena

- No es la forma más habitual de obtener información del usuario en un programa PHP, ya que como suelen ejecutarse en un servidor web, éstos se reciben a través de la query de entrada (método GET), o a través de las cabeceras HTTP (método POST)
- Para leer cadenas que contengan espacios, utilizar mejor la función **fgets()** o **readline**()
  - $cadena = **fgets**(STDIN);
  - $cadena = **readline**();

Funciones (1/6)

- Son el equivalente a los métodos en POO, pero no están asociadas a ninguna clase sino al propio script PHP.
- Sintaxis

```
function nombreFuncion($par1, $par2, …. $parN) {
        … cuerpo función ….
}
```

- Pueden devolver un valor con “return”, aunque podrían no devolver nada.
- Se invocan utilizando su nombre y proporcionando valores para sus parámetros que se copiarán en cada uno de ellos (paso por valor)

Funciones (2/6) ejemplo

```php
<?php
function maximo($n1, $n2) {
        $sol = $n2;
        if ($n1 > $n2) {
                $sol = $n1;
        }
        return $sol;
}
echo maximo(4,5);
?>
```

Saldría 5

```php
<?php
echo uno(); echo dos();
function uno() {return 1;}
function dos() {return 2;}
?>
```

Sería más correcto definir las funciones al principio pero así también vale.

Funciones (3/6) ámbito variables

```php
<?php
$x = 10;
function f() {
        echo $x;
}
f();
?>
```

Daría un error, ya que las variables son LOCALES a cada función y $x no tiene valor en f()

```php
<?php
$x = 10;
function f() {
        $x = 1;
        echo $x.” --- ”;
}
f();
echo $x;
?>
```

Saldría 1 --- 10

```php
<?php
$x = 10;
function f() {
        $x = 1;
        echo $x . ” --- ”;
        global $x;
        echo $x;
}
f();
?>
```

Saldría 1 --- 10

Funciones (4/6) paso por valor y referencia

```php
<?php
function f($xVal) {
        $xVal ++;
        echo $xVal . ”---” ;
}
$x = 1;
f($x);
echo $x
?>
```

Saldría 2 --- 1

```php
<?php
function f(&$xRef) {
        $xRef ++;
        echo $xRef . ”---” ;
}
$x = 1;
f($x);
echo $x
?>
```

Saldría 2 --- 2

Funciones (5/6) Valores por defecto

```php
<?php
function soy($asi = “listo”) {
        echo “Soy $asi \n”;
}
soy();
soy(“guapo”);
?>
```

Saldría:

Soy listo

Soy guapo

```php
<?php
function p($a = 4, $b = 2) {
        return $a / $b;
}
echo p () . "--";
echo p ( 8 ) . "--";
echo p ( 16, 2 ) ;

?>
```

Saldría: 2--4--8

Funciones (6/6) Parámetros variables

```php
<?php
function p() {
  for($i = 0; $i < func_num_args (); $i ++) {
    echo func_get_arg ( $i ), " ";
  }
  echo " // ";
}
p ( 'Pepe' );
p ( 'Rosa', 'Mari' );
?>
```

Saldría: Pepe // Rosa Mari //

```php
<?php      //sólo PHP >= 5.6
function sum(...$numeros) {
     $acc = 0;
     foreach ($numeros as $n) {
          $acc += $n;
     }
     return $acc;
}
echo sum(1, 2, 3, 4);
?>
```

Saldría: 10

Arrays (1/2)

- No hace falta declararlos. Basta con asignar algún valor a alguna de sus casillas para empezar a utilizarlos.
- Los valores asignados pueden ser de cualquier tipo y estar mezclados.
- Se pueden inicializar varios valores a la vez, utilizando el constructor array(v1,v2,...,vn), o encerrando los valores entre corchetes (PHP >= 5.4)

Arrays (2/2) ejemplos

```php
<?php
$a[1] = 10;
$a[4] = “Pepe”;
echo $a[1] + 10;
echo “ le gusta a $a[4]”;
?>
```

Saldría:

20 le gusta a Pepe

```php
<?php
$a[1] = 10;
echo $a[0];
?>
```

Daría un error

```php
<?php
$a = array(“SI”, true, ”NO”);
$b = [“a”, 2, true, 4.1];
if ($a[1]) {
        echo $a[0];
        }
else {
        echo $a[2];
        }
echo “---” . $b[1];
?>
```

Saldría “SI --- 2”

```php
<?php
$a=[1,2,3];
foreach ($a as $n) {
        echo $n+1,'/';
}

?>
```

Saldría “2/3/4/”

Arrays asociativos (1/2)

- Son arrays formados por pares clave => valor, como los Map de JAVA
- En realidad los arrays “normales” también son arrays asociativos.
- Los valores pueden ser de cualquier tipo, las claves sólo pueden ser numéricas o cadenas. Cualquier otro tipo (de las claves) se convertirá implícitamente a alguno de estos dos.
- Si se asignan varios valores a la misma clave sólo prevalecerá el último.
- Las librerías estándar PHP contienen un gran número de funciones útiles de arrays

Arrays asociativos (2/2) ejemplos

```php
<?php
$colorFavorito[‘pepe’]=”azul”;
$colorFavorito[“maria”]=’rosa’;

echo $colorFavorito[“pepe”];
echo “ --- “;

$nombre = “maria”;
echo $colorFavorito[$nombre];
?>
```

Saldría “azul --- rosa”

```php
<?php
$colorFavorito[“pepe”]=”azul”;
echo $colorFavorito[“Pepe”];
?>
```

Daría un error, porque los índices son sensibles a mayúsculas

```php
<?php
$colorFavorito = [
    “pepe” => “azul”,
    “maria” => “rosa”
];

echo $colorFavorito[“pepe”];
?>
```

Saldría “azul”;

Arrays anidados y/o mixtos

```php
<?php
$a = [
        [1,2,3],
        ['a','b','c']
    ];
echo $a[0][1] , “----”;
echo $a[1][0];

?>
```

Saldría: 2----a

```php
<?php
$a = [
        [1,2,3],
        [   "pepe"=>"rojo",
            "marta"=>"azul"],
        [[10,20],[30,40],[50,60]]
    ];
echo $a[0][1] , "---"; echo $a[1]["marta"] , "---";
echo $a[2][1][0];
?>
```

Saldría 2---azul---30

Funciones útiles de arrays

- Manejo de **PILAS (stack)**
  - Meter: **array_push**($array, $elem1 `[,más elems]`)
    - Alternativamente: $array[] = $elem1 `(si sólo 1 elem.)`
  - Extraer: **array_pop**($array)
- Manejo de **COLAS (queue)**
  - Meter: con **array_push(...)** (igual que en una pila)
  - Colar (por delante): **array_unshift**($array, $elem1 `[,más elems]`)
  - Extraer: **array_shift**($array)
- Verificar existencia de un valor
  - **in_array**($elem, $array)
- Obtener lista de claves
  - **array_keys**($array)

Cadenas(1/4) echo, print, print_r, var_dump, {}

- **echo** y **print** son constructores del lenguaje para mostrar cadenas de caracteres (no son funciones, por tanto no hace falta paréntesis para utilizarlos). Su sintaxis es:
  - void **echo** $cadena1, $cadena2, …., $cadenaN
  - int **print** $cadena
- Como print se comporta como una función (devolviendo 1 siempre), y PHP acepta expresiones como sentencias, podemos hacer cosas de este estilo.
  - $edad >= 18 ? print “Mayor” : print “Menor”;
- **print_r($v)**, muestra el valor de la variable $v, incluso si ésta es estructurada como un array.
- **var_dump($v)**. Similar a print_r, pero también indica el tipo de la variable.
- Las **llaves** dentro de una cadena de doble comilla sirven para acotar expresiones complejas de variables (p.ej echo “Hola {$a[1][‘pepe’]}”)

Cadenas (2/4) Heredoc y Nowdoc

- Son formas alternativas (a las comillas simples y dobles) de acotar una cadena de caracteres, útiles para no mezclar código HTML y PHP.
- Marca de apertura:
  - <<<ETIQUETA heredoc
  - <<<'ETIQUETA' nowdoc (PHP>5.3)
- Marca de cierre
  - ETIQUETA; Ha de ser el primer carácter de la línea
- El texto entre etiquetas puede ocupar varias líneas
- La diferencia entre los dos es que heredoc expande nombres de variables y el contenido de llaves y nowdoc no expande (similar a la diferencia entre comillas dobles y simples).
  - Nowdoc es útil, por ejemplo, para acotar código PHP que queremos que salga tal cual.

Cadenas (3/4) funciones útiles

- **strlen($c)**: devuelve el número de caracteres de $c
- **substr($c,ini,[n]):** devuelve la subcadena de $c desde el carácter de posición “ini” hasta “n” caracteres a la derecha de él incluido, teniendo en cuenta que la primera posición es cero. Si se omite n, devuelve hasta el final de la cadena. Si “ini” es negativo devuelve los n últimos caracteres
- **ltrim($c), rtrim($c), trim($c)** quita de $c espacios y caracteres no visibles (saltos de línea, tabuladores, etc) por la izquierda, derecha o en toda la cadena.
- **explode($sep, $c), implode($sep, $a)** Pasa de cadena a array o al revés partiendo o uniendo por el separador $sep

Cadenas (4/4) El infierno espa?ol

- Para que el manejo de tildes, eñes y diéresis no se vuelva un infierno, tanto en el script PHP como en el HTML generado recomiendo lo siguiente:
  - **mb_internal_encoding ("UTF-8");**
    - Al principio del script
  - **header('Content-Type: text/html; charset=UTF-8');**
    - Antes del primer echo del programa.
  - **echo '<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />';**
    - en el <head> del HTML
  - Utilizar las versiones multibyte "mb_" de las funciones habituales de string
    - p.ej: mb_strlen("uña") devuelve 3 // strlen("uña") devuelve 4
    - p.ej: mb_substr("uña",1,1) devuelve ñ
  - Asegurarse de que nuestro IDE o editor funciona con codificación UTF-8

Manejo de fechas (1/3)

- En PHP no existe el tipo de datos fecha, se trabaja con cadenas de caracteres y funciones que extraen la fecha y hora del sistema.
  - **time()** devuelve el timestamp actual (núm.de segundos transcurridos desde el 1/1/1970)
  - **date (formato, [timestamp])** devuelve una cadena con el formato especificado respecto al “timestamp” proporcionado. Si se omite será el actual.
    - Formato: **h**=horas, **H**=horas24, **i**=min., **s**=seg., **m**=mesNum, **M**=mes3L, **d**=día, **Y**=año
  - **strtotime**(string) devuelve el timestamp correspondiente a la fecha indicada en la cadena, en formatos varios
  - **mktime**(h, i, sa, m, d, Y) devuelve el timestamp que representa la fecha indicada por los argumentos. Se pueden dejar vacíos de derecha a izquierda.
  - **setlocale(LC_ALL,"es_ES")** configura formatos de fecha (y otros) en español
  - **strftime(formato,[timestamp])** Igual que date(...) pero para formatos locales.

Manejo de fechas (2/3)

```php
<?php
$hoy = time ();
$naci = mktime ( 0, 0, 0, 8, 19, 1980 );
$proxPilar = strtotime ( "10/12/14" );        //americano
$proxPilar = strtotime ( "2014/10/12" );      //ordenado
$proxPilar = strtotime ( "12-10-14" );        //español

echo "Fecha de hoy: ", date ( "d m Y", $hoy ), '<br>';
echo "Son las ", date ( "H:i", $hoy ), '<br>';
echo "Nací el ", date ( "d M Y", $naci ), '<br>';
echo "El Pilar guiri es en el mes de ", date ( "F", $proxPilar ),'<br>';
setlocale ( LC_ALL, "es_ES" );
echo "En España es en ", strftime( "%B", $proxPilar ),'<br>';
?>
```

Fecha de hoy: 23 09 2014
Son las 20:56
Nací el 19 Aug 1980
El Pilar guiri es en el mes de October
En España es en octubre

Fechas (3/3) Limitaciones

- Los timestamp se codifican utilizando un número entero cuyo rango oscila entre aprox. -2.000.000.000 y + 2.000.000.000
- Esto da un rango de manejo de fechas aproximado entre 1901 y 2038
- Para fechas fuera de estos rangos, o para evitar el “efecto 2038” utilizar objetos de la clase DateTime

Niveles de mensajes de error en PHP

- Se pueden cambiar definitivamente en el php.ini, línea “error_reporting”
  - E_ALL significa todos los mensajes.
- Se pueden cambiar momentáneamente en un programa utilizando la función
  - error_reporting(tipo)
  - Conectar los tipos a visualizar con “|”.
  - 0 para no mostrar ningún error
- Se pueden almacenar en un archivo indicándolo en la línea “log_errors” del php.ini, normalmente estarán en “/var/log/error_log” (el cual deberá tener permisos para que el usuario “apache” escriba
- Para cambiar el archivo de log, hacerlo en la directiva “error_log”
- Poner “display_errors” a 0 para no verlos por pantalla (sólo en producción)
- Todos los niveles, aquí

Nivel	Tipo	Ejemplo
Sintáctico	E_PARSE	x = 3 (No tiene sentido ocultarlo)
Lógico	E_NOTICE	echo de variable no inicializada
Semántico	E_WARNING	División por cero, “include” de fichero inexistente
Fatal	E_ERROR	Llamada a función inexistente (Rompe ejecución)
Conceptual	E_STRICT	Método no estático llamado estáticamente

Inclusión de ficheros (include, require)

- Sirven para incluir el contenido de un fichero php dentro de otro (como una especie de copia & pega), y sirven fundamentalmente para crear **librerías**.
- Las instrucciones de inclusión son: **include(**rutaFicheroIncluido.php**)**, **include_once(**rutaFicheroIncluido.php**)**, **require(**rutaFicheroIncluido.php**)** y **require_once(**rutaFicheroIncluido.php**)**
- **rutaFicheroIncluido.php** puede ser absoluta (respecto a los “include path” definidos en el php.ini) o relativa (respecto a la ruta en la que se encuentra el script donde se ha ejecutado el include.
- Para añadir otros “include path”, modificar el “php.ini” ó utilizar la función “set_include_path(path)”
- **include** y **require** difieren en que este último da un error fatal E_COMPILE_ERROR en lugar de un WARNING y se detiene la ejecución del script en el caso de que el fichero a incluir no exista.
- **_once** difiere en que sólo se incluyen los archivos una vez en la ejecución del script.

Referencias

- Referencia y tutoriales PHP W3schools
- Referencia y tutoriales PHP.net