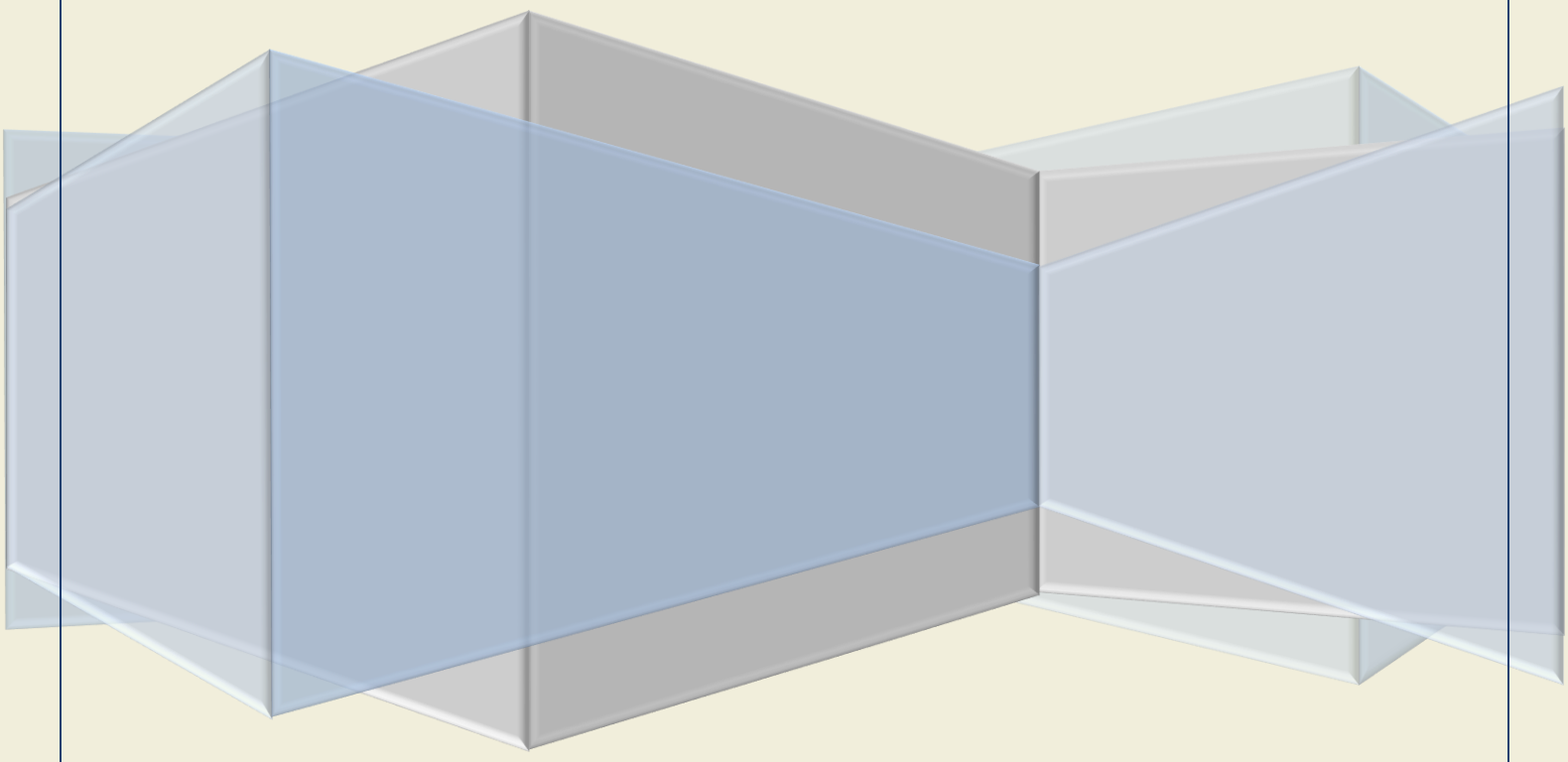
Capítulo 2
Variables y Operadores

Programación: Backend Developer

Potrero Digital



CONTENIDO

VARIABLES

¿QUÉ ES UNA VARIABLE?

En los lenguajes de programación, una variable es un elemento que permite almacenar información. Las variables se identifican por su nombre. Para facilitar la comprensión del programa, se aconseja que los nombres de las variables hagan referencia a la información que contienen. Por ejemplo, si se va a guardar en una variable la edad del usuario, un nombre adecuado de la variable sería por ejemplo $edad.

En PHP el programador puede dar el nombre que quiera a las variables, con algunas restricciones:

- Los nombres de las variables tienen que empezar por el carácter $.
- A continuación tiene que haber una letra (mayúscula o minúscula) o un guion bajo (_).
- El resto de caracteres del nombre pueden ser números, letras o guiones bajos.

```php
<?php
// Ejemplos de nombres de variables:
$edad   = 16;
$edad1  = 17;
$edad_1 = 18;
// Ejemplos de nombres de variables (estilos NO redomendados)
$años = 19;
$_edad  = 20;
$Edad   = 21;
$__edad = 22;
$EDAD   = 23;
$_EDAD  = 24;
?>
```

Los nombres de variables pueden contener caracteres no ingleses (vocales acentuadas, eñes (ñ) o cedillas (ç), etc.) pero se aconseja no hacerlo. Si los programas que escribimos los van a leer solamente programadores de países en los que también se utilizan esos caracteres, no habrá inconveniente, pero si colaboran con nosotros programadores de países en los que esos caracteres no se utilizan, les resultarán incómodos editar los programas. De hecho, algunos aconsejan incluso utilizar términos ingleses en los nombres de variables para facilitar la colaboración entre programadores de distintos países.

En los nombres de las variables, PHP distingue entre mayúsculas y minúsculas, es decir, si se cambia algún carácter de mayúscula a minúscula o viceversa, para PHP se tratará de variables distintas.

Si el nombre de la variable contiene varias palabras, se aconseja utilizar la notación "camel case", es decir, escribir la primera palabra en minúsculas y la primera letra de las siguientes palabras en mayúsculas.

```php
<?php
// Ejemplos de nombres de variables Camel Case:
$edadHijo          = 16;       // edad del hijo
$edadMadre         = 47;       // edad de la madre
$edadPadre         = 46;       // edad del padre
$edadAbueloPaterno = 73;       // edad del abuelo paterno
?>
```

PHP define automáticamente una serie de variables (son las llamadas variables predefinidas). Los nombres de estas variables siguen siempre el mismo patrón: empiezan por un guion bajo y se escriben en mayúsculas (por ejemplo, $_REQUEST, $_SERVER, $_SESSION, etc.). Para evitar conflictos con variables

predefinidas que se creen en el futuro, se recomienda no crear en los programas variables con nombres que sigan ese mismo patrón.

TIPOS DE VARIABLES

Los tipos de variables básicos son los siguientes:

- lógicas o booleanas (boolean)
- enteros (integer)
- decimales (float)
- cadenas (string)
- matrices (arrays)

Existen además los tipos:

- objetos (object)
- recursos (resource)
- nulo (**null**)

VARIABLES LÓGICAS (BOOLEAN)

Las variables de tipo lógico sólo pueden tener el valor **true** (verdadero) o **false** (falso). Se suelen utilizar en las estructuras de control.

Nota: El ejemplo siguiente utiliza la estructura de selección **if** que se explica en detalle en la lección de estructuras de control. Para entender este ejemplo, es suficiente saber que if significa si (si como condición, no sí como afirmación) y va seguida de una comparación de igualdad == (las comparaciones se explican en la lección de operaciones lógicas). En caso de que la comparación sea cierta, es decir si los dos términos a ambos lados de la comparación son iguales, se ejecuta la instrucción entre corchetes **{ }**.

```
<?php
$autorizado = true;

if ($autorizado == true) {
    print "<p>Usted está autorizado.</p>\n";
}

if ($autorizado == false) {
    print "<p>Usted no está autorizado.</p>\n";
}
?>
```

```
<p>Usted está autorizado.</p>
```

Estos valores se pueden escribir en mayúsculas o minúsculas o combinando ambas, aunque se recomienda utilizar minúsculas:

```
<?php
$autorizado = fAlSe;

if ($autorizado == TRUe) {
    print "<p>Usted está autorizado.</p>\n";
}
```

```php
if ($autorizado == FaLsE) {
    print "<p>Usted no está autorizado.</p>\n";
}
?>
```

```
<p>Usted no está autorizado.</p>
```

Nota: No es necesario comparar una variable lógica con **true** o **false**, podemos emplear variables lógicas directamente en la condición del **if**.

```php
<?php
$autorizado = true;

if ($autorizado) {            // equivale a $autorizado == true
    print "<p>Usted está autorizado.</p>\n";
}

if (!$autorizado) {           // equivale a $autorizado == false
    print "<p>Usted no está autorizado.</p>\n";
}
?>
```

```
<p>Usted está autorizado.</p>
```

```php
<?php
$autorizado = false;

if ($autorizado) {            // equivale a $autorizado == true
    print "<p>Usted está autorizado.</p>\n";
}

if (!$autorizado) {           // equivale a $autorizado == false
    print "<p>Usted no está autorizado.</p>\n";
}
?>
```

```
<p>Usted no está autorizado.</p>
```

VARIABLES ENTERAS (INTEGER)

Las variables de tipo entero pueden guardar números enteros (positivos o negativos).

```php
<?php
$lado = 14;
$area = $lado * $lado;

print "<p>Un cuadrado de lado $lado cm \ntiene un área de $area cm<sup>2</sup>.</p>\n";
?>
```

```
<p>Un cuadrado de lado 14 cm
tiene un área de 196 cm<sup>2</sup>.</p>
```

Como las variables enteras se guardan en la memoria del ordenador utilizando un número de bytes fijo (que depende del ordenador, pero se puede averiguar mediante la constante predefinida PHP_INT_SIZE), no se

pueden guardar números arbitrariamente grandes o pequeños. El valor más grande que se puede almacenar se puede averiguar mediante la constante predefinida PHP_INT_MAX. Los enteros se guardan en notación complemento a dos.

NOTACIÓN COMPLEMENTO A DOS

En un byte (8 bits) se pueden almacenar 256 (2^8) valores diferentes. Eso permite almacenar por ejemplo, desde el valor 0 hasta el 255. Para almacenar tanto números negativos como positivos, se utiliza la notación complemento a dos, en la que la primera mitad de los valores representan los valores positivos (desde 0 hasta 127) y la segunda mitad de los valores representan los valores negativos (desde -128 hasta -1), como indica la tabla siguiente

El valor ...	0	1	2	3	4	5	...	126	127	128	129	...	253	254	255
... representa el entero ...	0	1	2	3	4	5	...	126	127	-128	-127	..	-3	-2	-1

En cuatro bytes (32 bits) se pueden almacenar 4.294.967.296 (2^{32}) valores distintos. En notación complemento a dos se pueden representar desde el valor -2.147.483.648 hasta 2.147.483.647.

Si se intenta guardar un número demasiado grande positivo, el resultado será seguramente incorrecto. En el ejemplo siguiente se fuerza a PHP a guardar como entero el valor siguiente al máximo valor posible. El valor se guarda, pero cuando se utiliza, PHP lo interpreta como negativo.

```
<?php
$maximo = PHP_INT_MAX;
print "<p>El mayor entero que se puede guardar en una variable entera es
$maximo</p>\n";

$demasiado = (int)($maximo+1);
print "<p>Si se intenta guardar 1 más, el resultado es $demasiado</p>\n";
?>
```

```
<p>El mayor entero que se puede guardar en una variable entera es 2147483647</p>
<p>Si se intenta guardar 1 más, el resultado es -2147483648</p>
```

VARIABLES DECIMALES (FLOAT)

Las variables de tipo decimal (float) pueden guardar números decimales (positivos o negativos). Como en las calculadoras, el separador de la parte entera y la parte decimal es el punto (.), no la coma (,)..

```
<?php
$lado = 14.5;
$area = $lado * $lado;

print "<p>Un cuadrado de lado $lado cm \ntiene un área de $area cm<sup>2</sup>.</p>\n";
?>
```

```
<p>Un cuadrado de lado 14.5 cm
tiene un área de 210.25 cm<sup>2</sup>.</p>
```

Como las variables decimales se guardan en la memoria del ordenador utilizando un número de bytes fijo (que depende del ordenador, aunque no existen constantes predefinidas para conocer su tamaño), no se pueden guardar números arbitrariamente grandes o pequeños. Normalmente las variables decimales siguen la norma IEEE 758, concretamente el formato denominado "doble precisión", que emplea 8 bytes (64 bits) para almacenar un número.

De esos 64 bits, uno se utiliza para el signo, 53 para la parte fraccionaria y 11 para el exponente. Eso hace que el valor más grande que se pueda almacenar sea aproximadamente $1{,}8.10^{308}$.

Si se intenta guardar un número demasiado grande positivo (tanto si ese valor se obtiene en un cálculo o si se escribe directamente) PHP no puede manejarlo y lo que hace es sustituirlo por la constante predefinida INF.

```
<?php
$maximo = 10 **308;  // 10^308
print "<p>10<sup>308</sup> se puede guardar en una variable decimal: $maximo</p>\n";
print "\n";
$demasiado = 10 * $maximo;
print "<p>Si se intenta guardar 10<sup>309</sup>, el resultado es $demasiado</p>\n";
?>
```

```
<p>10<sup>308</sup> se puede guardar en una variable decimal: 1.0E+308</p>

<p>Si se intenta guardar 10<sup>309</sup>, el resultado es INF</p>
```

```
<?php
$prueba =
100000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000;
print "<p>Si se intenta guardar 10<sup>309</sup>, el resultado es $prueba\n";
?>
```

```
<p>Si se intenta guardar 10<sup>309</sup>, el resultado es INF</p>
```

VARIABLES DE CADENAS (STRING)

Las variables de tipo cadena pueden guardar caracteres.

PHP no impone ningún límite al tamaño de las cadenas. Las cadenas pueden ser todo lo largas que permita la memoria del servidor.

El juego de caracteres que utiliza PHP viene determinado en principio por el juego de caracteres que utiliza el fichero fuente del programa. Pero hay que tener en cuenta que las funciones de tratamiento de cadenas no están preparadas para tratar la diversidad de juegos de caracteres: muchas suponen que cada carácter ocupa solamente un byte, otras suponen un juego de caracteres determinado (UTF-8, por ejemplo), otras utilizan el juego de caracteres definido localmente, etc.

Se puede acceder a caracteres individuales indicando la posición del carácter, como si se tratara de una matriz de una dimensión en la que el primer carácter ocupa la posición 0.

```
<?php
$saludo = "Hola, Don Pepito";
print "<p>$saludo</p>\n";

$saludo[0] = "M";
print "<p>$saludo</p>\n";

$saludo[14] = "n";
print "<p>$saludo</p>\n";
```

```
?>
<p>Hola, Don Pepito</p>
<p>Mola, Don Pepito</p>
<p>Mola, Don Pepino</p>
```

Si se indica una posición mayor que la longitud de la cadena, la cadena se alarga con espacios hasta llegar a ese valor:

```
<?php
$saludo = "Hola, Don Pepito";
print "<p>$saludo</p>\n";

$saludo[16] = "n";
print "<p>$saludo</p>\n";

$saludo[25] = "!";
print "<pre>$saludo</pre>\n";
?>
<p>Hola, Don Pepito</p>
<p>Hola, Don Pepiton</p>
<pre>Hola, Don Pepiton         !</pre>
```

Si en una posición se guarda una cadena vacía, la cadena se acorta eliminando el carácter de esa posición

```
<?php
$saludo = "Hola, Don Pepito";
print "<p>$saludo</p>\n";

$saludo[4] = "";
print "<p>$saludo</p>\n";
?>
<p>Hola, Don Pepito</p>
<p>Hola Don Pepito</p>
```

En PHP, las cadenas de texto se delimitan por comillas (dobles o simples).

```
<?php
print "<p>Esto es una cadena.</p>\n";
?>
<p>Esto es una cadena.</p>
<?php
print '<p>Esto es otra cadena.</p>';
?>
<p>Esto es otra cadena.</p>
```

Las cadenas contienen lógicamente caracteres del juego de caracteres en el que se guarde el archivo (UTF-8, CP-1252, ISO-8859-1, etc.), pero algunos caracteres necesitan un tratamiento especial, como se explica más adelante.

Si las comillas de apertura y cierre no son del mismo tipo (una de ellas doble y otra simple), se produce un error de sintaxis.

```php
<?php
print "<p>Esto es una cadena.</p>\n';
?>
```

Parse error: syntax error, unexpected end of file, expecting
variable (T_VARIABLE) or ${ (T_DOLLAR_OPEN_CURLY_BRACES)
or {$ (T_CURLY_OPEN) in **ejemplo.php** on line **3**

COMILLAS DENTRO DE CADENAS

Si una cadena está delimitada por comillas dobles, en su interior puede haber cualquier número de comillas simples, y viceversa.

```php
<?php
print "<p>Esto es una comilla simple: '</p>";
?>
```

```
<p>Esto es una comilla simple: '</p>
```

```php
<?php
print '<p>Esto es una comilla doble: "</p>';
?>
```

```
<p>Esto es una comilla doble: "</p>
```

Lo que no puede haber en una cadena es una comilla del mismo tipo que las que delimitan la cadena.

```php
<?php
print "<p>Esto es una comilla doble: "</p>";
?>
```

Parse error: syntax error, unexpected '/' in **ejemplo.php** on line **2**

```php
<?php
print '<p>Esto es una comilla simple: '</p>';
?>
```

Parse error: syntax error, unexpected '/' in **ejemplo.php** on line **2**

Para poder escribir en una cadena una comilla del mismo tipo que las que delimitan la cadena, se debe utilizar los caracteres especiales \' o \".

```php
<?php
print "<p>Esto es una comilla simple: ' y esto una comilla doble: \"</p>";
?>
```

```
<p>Esto es una comilla simple: ' y esto una comilla doble: "</p>
```

```php
<?php
print '<p>Esto es una comilla simple: \' y esto una comilla doble: "</p>';
?>
```

```
<p>Esto es una comilla simple: ' y esto una comilla doble: "</p>
```

Y por supuesto se pueden utilizar los caracteres especiales \' y \" con cualquier delimitador:

```php
<?php
print "<p>Esto es una comilla simple: \' y esto una comilla doble: \"</p>";
?>
```

```
<p>Esto es una comilla simple: ' y esto una comilla doble: "</p>
```

```php
<?php
print '<p>Esto es una comilla simple: \' y esto una comilla doble: \"</p>';
?>
```

```
<p>Esto es una comilla simple: ' y esto una comilla doble: "</p>
```

Pero los caracteres especiales \" y \' no se pueden utilizar para delimitar cadenas.

```php
<?php
print \"<p>Esto es una cadena.</p>\n\";
?>
```

Parse error: syntax error, unexpected '"', expecting identifier (T_STRING)
in **ejemplo.php** on line **2**

```php
<?php
print \'<p>Esto es una cadena.</p>\n\';
?>
```

Parse error: syntax error, unexpected " Esto es una cadena. \n\'
(T_CONSTANT_ENCAPSED_STRING), expecting identifier (T_STRING)
in **ejemplo.php** on line **2**

DIFERENCIAS ENTRE COMILLAS SIMPLES Y DOBLES

Aunque en los ejemplos anteriores las comillas simples o dobles son equivalentes, en otras situaciones no lo son. Por ejemplo, PHP no sustituye las variables que se encuentran dentro de cadenas delimitadas con comillas simples, mientras que sí que lo hace (pero no siempre) si se utilizan comillas dobles, como se ve en el siguiente ejemplo:

```php
<?php
$cadena = "Hola";
print "<p>La variable contiene el valor: $cadena</p>";
?>
```

```
<p>La variable contiene el valor: Hola</p>
```

```php
<?php
$cadena = "Hola";
print '<p>La variable contiene el valor: $cadena</p>';
?>
```

```
<p>La variable contiene el valor: $cadena</p>
```

PHP tampoco sustituye algunos caracteres especiales (por ejemplo, el salto de línea \n) dentro de las comillas simples, como se ve en el siguiente ejemplo:

```php
<?php
print "<pre>Esto está en\ndos líneas.</pre>";
?>
```

```
<pre>Esto está en
dos líneas</pre>
```

```php
<?php
print '<pre>Esto está en\ndos líneas.</pre>';
?>
```

```
<pre>Esto está en\ndos líneas</pre>
```

COMILLAS EN CÓDIGO HTML / CSS

En el código HTML generado con PHP también se pueden escribir comillas simples o dobles. Los dos ejemplos siguientes producen código html válido:

```
<?php
print "<p><strong style='color: red;'>Hola</strong></p>";
?>
```

```
<p><strong style='color: red;'>Hola</strong></p>
```

```
<?php
print '<p><strong style="color: red;">Hola</strong></p>';
?>
```

```
<p><strong style="color: red;">Hola</strong></p>
```

En estos apuntes se delimitan las cadenas con comillas dobles. Si el código html / css generado contiene comillas, en estos apuntes se generan también comillas dobles, por lo que dentro de las cadenas se utiliza el carácter especial \". Únicamente en las consultas SQL (que se verán más adelante) se utilizan comillas simples dentro de las cadenas.

```
<?php
print "<p><strong style=\"color: red;\">Hola</strong></p>";
?>
```

```
<p><strong style="color: red;">Hola</strong></p>
```

CARACTERES ESPECIALES

Por distintos motivos, algunos caracteres necesitan escribirse dentro de las cadenas de una forma especial. Por ejemplo:

- En PHP los nombres de variables empiezan por el carácter dólar ($). Cuando en una cadena aparece una palabra que empieza por el carácter $, PHP entiende que esa palabra hace referencia a una variable. Pero si a esa variable no se le ha dado valor anteriormente, se producirá un aviso. Además PHP escribe el valor de la variable (es decir, nada si la variable no está definida):

```
<?php
print "<p>Los nombres de variables empiezan por $. Por ejemplo $edad.</p>";
?>
```

Notice: Undefined variable: edad in **ejemplo.php** on line **2**

Los nombres de variables empiezan por $. Por ejemplo.

- Si en una cadena se utilizan comillas del mismo tipo (simple o doble) que las que la delimitan, se se produce un error de sintaxis y el programa no puede ejecutarse. El problema es que cuando PHP encuentra la comilla dentro de la cadena (simple o doble, pero del mismo tipo que delimita la cadena), PHP entiende que la cadena se termina precisamente ahí. Como PHP es incapaz de entender lo que viene a continuación, se produce un error de sintaxis.

```
<?php
print "<p>Las cadenas se delimitan con comillas (" o ').</p>";
?>
```

Parse error: syntax error, unexpected 'o' (T_STRING) in **ejemplo.php** on line **2**

Algunos caracteres (saltos de línea, tabuladores, etc) no se pueden escribir con el teclado.

Para poder escribir estos caracteres en una cadena, en PHP existen los llamados caracteres especiales. Los caracteres especiales empiezan por el carácter contrabarra (\). Los caracteres especiales más utilizados son los siguientes:

Carácter	Significado
****	Carácter contrabarra
\$	Carácter dólar
\"	Carácter comilla doble (en una cadena delimitada por comillas dobles)
\'	Carácter comilla simple (en una cadena delimitada por comillas simples)
\n	Salto de línea
\t	Tabulador horizontal

Además, existen también estos caracteres especiales, menos utilizados que los anteriores:

Carácter	Significado
\XXX	Carácter de código XXX en octal
\xXX	Carácter de código XX en hexadecimal
\r	Retorno de carro
\v	Tabulador vertical
\e	Escape
\f	Salto de página (en impresoras antiguas)

Los ejemplos anteriores se podrían escribir así:

```
<?php
print "<p>Los nombres de variables empiezan por $. Por ejemplo \$edad.</p>";
?>
```

```
<p>Los nombres de variables empiezan por $. Por ejemplo $edad.</p>
```

```
<?php
print "<p>Las cadenas se delimitan con comillas (\" o ').</p>";
?>
```

```
<p>Las cadenas se delimitan con comillas (" o ').</p>
```

SALTOS DE LÍNEA EN EL CÓDIGO FUENTE

Se pueden insertar saltos de línea en el código HTML de la página insertándolos directamente en el interior de las cadenas o mediante el carácter especial \n. Este salto de línea es ignorado por los navegadores, pero aumenta la legibilidad del código fuente.

Los ejemplos siguientes producen el mismo resultado en los navegadores (una lista de dos elementos), pero el código fuente es más legible en los tres últimos casos. En las soluciones de los ejercicios de este curso se utiliza preferiblemente el último estilo, ya que facilita la reutilización de fragmentos de código.

```php
<?php
print "<ul><li>Uno</li><li>Dos</li></ul>\n";
?>
```
```
<ul><li>Uno</li><li>Dos</li></ul>
```

```php
<?php
print "<ul>";
print "<li>Uno</li>";
print "<li>Dos</li>";
print "</ul>\n";
?>
```
```
<ul><li>Uno</li><li>Dos</li></ul>
```

```php
<?php
print "<ul>\n  <li>Uno</li>\n  <li>Dos</li>\n</ul>\n";
?>
```
```
<ul>
  <li>Uno</li>
  <li>Dos</li>
</ul>
```

```php
<?php
print "<ul>
  <li>Uno</li>
  <li>Dos</li>
</ul>\n";
?>
```
```
<ul>
  <li>Uno</li>
  <li>Dos</li>
</ul>
```

```php
<?php
print "<ul>\n";
print "  <li>Uno</li>\n";
print "  <li>Dos</li>\n";
print "</ul>\n";
?>
```
```
<ul>
  <li>Uno</li>
  <li>Dos</li>
</ul>
```

CONCATENAR CADENAS

El operador . (punto) permite concatenar dos o más cadenas.

```php
<?php
print "<p>Pasa" . "tiempos</p>\n";
?>
```
```
<p>Pasatiempos</p>
```

El mismo resultado se puede conseguir sin utilizar el operador . (punto):

```
<?php
print "<p>Pasatiempos</p>\n";
?>
```

```
<p>Pasatiempos</p>
```

Trabajando únicamente con cadenas, el operador . (punto) no parece demasiado útil, pero en la lección dedicada a Variables se explica el uso del operador utilizando variables y cadenas, que es el uso más habitual.

El operador de asignación con concatenación (.=) permite concatenar una cadena a otra y asignarla a esta:

```
<?php
$cadena = "Pasa";
print "<p>$cadena</p>\n";
$cadena .= "tiempos";
print "<p>$cadena</p>\n";
?>
```

```
<p>Pasa</p>
<p>Pasatiempos</p>
```

MATRICES (ARRAYS)

Una matriz es un tipo de variable que permite almacenar simultáneamente varios datos diferentes, a los que se accede mediante un índice, numérico o de texto. En inglés, las matrices se llaman *arrays*. A veces el término inglés *array* se traduce como *arreglo*.

En PHP, una matriz es un tipo de variable muy flexible, ya que podemos añadir, modificar, eliminar o reordenar los elementos de forma individual. Además los elementos pueden ser de tipos de datos diferentes.

Si los elementos de una matriz son datos de tipos simples (booleanos, enteros, decimales o cadenas), sólo se necesita un índice para identificar los datos. Se dice entonces que las matrices son unidimensionales.

A las matrices de una dimensión también se les llama vectores.

CREAR UNA MATRIZ

En la notación compacta, las matrices se crean empleando corchetes ([]).

```
<?php
// Notación compacta
$nombres = ["Ana", "Bernardo", "Carmen"];
?>
```

Los elementos de la matriz deben separarse con comas. Tras el último elemento se puede escribir o no una coma, pero la coma final no crea un nuevo elemento (la matriz obtenida es la misma, independientemente de que se escriba la coma final o no).

```
<?php
// Notación compacta
$nombres = ["Ana", "Bernardo", "Carmen", ];
?>
```

Para hacer referencia a los valores individuales de la matriz, se deben utilizar índices, que se escriben entre corchetes ([ ]). Si al crear la matriz no se han indicado otros valores de índices, el primer término tiene el índice [0], el segundo tiene el índice [1], etc.:

PHP sustituye las referencias a valores de matrices de una dimensión dentro de las cadenas, por lo que no es necesario concatenar cadenas y referencias a matrices.

```
<?php
$nombres = ["Ana", "Bernardo", "Carmen"];

print "<p>$nombres[1]</p>\n";
print "<p>$nombres[0]</p>\n";
?>
```

Bernardo

Ana

Una vez creada la matriz, los elementos individuales se pueden utilizar como si fueran variables independientes. Pero para referirse a un elemento individual, hay que indicar siempre el índice correspondiente.

```
<?php
$nombres = ["Ana", "Bernardo", "Carmen"];

print "<p>$nombres[1]</p>\n";

$nombres[1] = "David";

print "<p>$nombres[1]</p>\n";
?>
```

Bernardo

David

Si se solicita un valor no definido de una matriz, se produce un aviso (undefined offset). Los avisos no interrumpen la ejecución del programa, pero se deben corregir porque el programa seguramente no tendrá el comportamiento esperado:

```
<?php
$nombres = ["Ana", "Bernardo", "Carmen"];

print "<p>$nombres[3]<p>\n";
?>
```

Notice: Undefined offset: 3 in **ejemplo.php** on line **4**

Se puede crear una matriz vacía (para añadirle posteriormente elementos). Se suele hacer para asegurarse de que una matriz ya utilizada en el programa no contiene elementos o para evitar errores si las operaciones que se van a realizar posteriormente requieren la existencia de la matriz (por ejemplo, si se van a utilizar uniones de matrices).

```php
<?php
$nombres = [];
?>
```

También se pueden crear matrices asignando directamente un elemento de la matriz.

```php
<?php
$apellidos[1] = "García";

print "<p>$apellidos[1]</p>\n";
?>
```

García

MATRICES ASOCIATIVAS

Las matrices de PHP son matrices *asociativas*, es decir, que los índices no tienen por qué ser correlativos, ni siquiera tienen por qué ser números.

Al crear matrices asociativas, debemos indicar el valor de los índices, utilizando la notación **$indice => $valor**:

```php
<?php
$cuadrados = [3 => 9, 5 => 25, 10 => 100];

print "<p>El cuadrado de 3 es $cuadrados[3]</p>\n";
?>
```

El cuadrado de 3 es 9

PHP sustituye las referencias a valores de matrices de una dimensión dentro de las cadenas, por lo que no es necesario concatenar cadenas y referencias a matrices, pero los índices deben escribirse sin comillas, aunque sean cadenas.

```php
<?php
$edades = ["Andrés" => 20, "Bárbara" => 19, "Camilo" => 17];

print "<p>Bárbara tiene $edades["Bárbara"] años</p>\n";
?>
```

Parse error: syntax error, unexpected '"', expecting identifier (T_STRING) or variable (T_VARIABLE) or number (T_NUM_STRING) in **ejemplo.php** on line **4**

```php
<?php
$edades = ["Andrés" => 20, "Bárbara" => 19, "Camilo" => 17];

print "<p>Bárbara tiene $edades[Bárbara] años</p>\n";
?>
```

Bárbara tiene 19 años

Si la referencia a un valor de una matriz está fuera de una cadena o entre llaves, los índices que son cadenas deben escribirse con comillas.

```php
<?php
$edades = ["Andrés" => 20, "Bárbara" => 19, "Camilo" => 17];

print "<p>Bárbara tiene " . $edades[Bárbara] . " años</p>\n";
print "\n";
print "<p>Camilo tiene {$edades[Camilo]} años</p>\n";
?>
```

Notice: Use of undefined constant Bárbara - assumed 'Bárbara' in **ejemplo.php** on line **4**

Bárbara tiene 19 años

Notice: Use of undefined constant Camilo - assumed 'Camilo' in **ejemplo.php** on line **6**

Camilo tiene 17 años

```php
<?php
$edades = ["Andrés" => 20, "Bárbara" => 19, "Camilo" => 17];

print "<p>Bárbara tiene " . $edades["Bárbara"] . " años</p>\n";
print "\n";
print "<p>Camilo tiene {$edades["Camilo"]} años</p>\n";
?>
```

Bárbara tiene 19 años

Camilo tiene 17 años

CONSTANTES

Las constantes son elementos de PHP que guardan un valor fijo que no se puede modificar a lo largo del programa. Las constantes pueden ser definidas por el programa o estar predefinidas por el propio PHP o por algún módulo. Los nombres de las constantes siguen las mismas reglas que los nombres de las variables, pero sin el dólar ($) inicial. La costumbre es escribir los nombres de las constantes en mayúsculas.

En principio, se puede no utilizar constantes nunca, puesto que las constantes definidas por el programa podrían reemplazarse por variables. La ventaja de usar constantes y variables es que se puede distinguir a simple vista si a lo largo de un programa algo va a permanecer constante (si es una constante) o puede cambiar (si es una variable). El inconveniente de usar constantes es que las constantes no se sustituyen dentro de las cadenas y es necesario sacarlas fuera de las cadenas, haciendo el código un poco más incómodo de escribir y leer. Desde el punto de vista del rendimiento, la diferencia es inapreciable.

DEFINIR CONSTANTES

La función **define(nombre_constante, valor_constante)** permite definir constantes.

```php
<?php
define("PI", 3.14);
print "<p>El valor de pi es " . PI . "</p>\n";
?>
```

```
<p>El valor de pi es 3.14</p>
```

```php
<?php
define("AUTOR", "Bartolomé Sintes Marco");
```

```
print "<p>Autor: " . AUTOR . "</p>\n";
?>
```
```
<p>Autor: Bartolomé Sintes Marco</p>
```

La costumbre es escribir los nombres de las constantes en mayúsculas. Los nombres de las constantes deben empezar por una letra y tener sólo letras (acentos, etc), números y guiones bajos.

Las constantes no se sustituyen dentro de una cadena (ni siquiera escribiéndolas entre llaves), por lo que es necesario concatenarlas para mostrar su valor.

```
<?php
define("PI", 3.14);
print "<p>El valor de pi es PI</p>\n";          // El valor NO se sustituye
print "<p>El valor de pi es {PI}</p>\n";        // El valor NO se sustituye
print "<p>El valor de pi es " . PI . "</p>\n"; // El valor SÍ se sustituye
?>
```
```
<p>El valor de pi es PI</p>
<p>El valor de pi es {PI}</p>
<p>El valor de pi es 3.14</p>
```

En las constantes, PHP distingue entre minúsculas y mayúsculas:

```
<?php
define("PI", 3.14);
define("pi", 3.141592);
print "<p>El valor de pi es " . PI . "</p>";
print "<p>El valor de pi es " . pi . "</p>";
?>
```
```
<p>El valor de pi es 3.14</p>
<p>El valor de pi es 3.141592</p>
```

CONSTANTES PREDEFINIDAS

Tanto PHP como los módulos cargados definen automáticamente una serie de constantes predefinidas, de las que se comentan algunas en esta lección:

- INF
- PHP_INT_MAX
- PHP_INT_SIZE

LISTA COMPLETA DE CONSTANTES PREDEFINIDAS

El número de constantes predefinidas depende de los módulos cargados en php.ini. La función get_defined_constants() devuelve las constantes predefinidas en el servidor que estemos utilizando:

```
<?php
print "<pre>";
print_r(get_defined_constants());
print "</pre>\n";
?>
```
```
<pre>Array
(
```

```
    [E_ERROR] => 1
    [E_RECOVERABLE_ERROR] => 4096
    [E_WARNING] => 2
    [E_PARSE] => 4
    [E_NOTICE] => 8
    [E_STRICT] => 2048
    ...
)
</pre>
```

USAR VARIABLES

Para guardar un valor en una variable se utiliza el operador de asignación (=) escribiendo a la izquierda únicamente el nombre de la variable y a la derecha el valor que queremos guardar. Si queremos guardar un número, no hace falta poner comillas, pero si queremos guardar una cadena de texto hay que poner comillas (dobles o simples).

```php
<?php
$edad = 15;                // La variable $edad tiene ahora el valor 15
$nombre = "Pepito Conejo";  // La variable $nombre tiene ahora el valor Pepito Conejo
?>
```

El programa anterior es correcto, aunque no produciría ningún resultado visible para el usuario.

En el lado izquierdo de la asignación (=) no se puede escribir más que el nombre de una variable. El programa siguiente no es sintácticamente correcto por lo que no se ejecutará y PHP mostrará un mensaje de error.

```php
<?php
$edad + 1 = 16;
print $edad;
?>
```

Parse error: syntax error, unexpected '=' in **ejemplo.php** on line **2**

Una vez se ha guardado un valor en una variable, se puede utilizar en el resto del programa.

```php
<?php
$edad = 15;
print $edad;
?>
```

```
15
```

A lo largo de un programa una variable puede guardar diferentes valores.

```php
<?php
$edad = 15;
print "<p>$edad</p>\n";
$edad = 20;
print "<p>$edad</p>\n";
?>
```

```
<p>15</p>
<p>20</p>
```

En la parte derecha de la asignación se pueden escribir expresiones matemáticas:

```
<?php
$radio = 15;
$perimetro = 2 * 3.14 * 15;
print $perimetro;
?>
```

```
94.2
```

Esas expresiones pueden contener variables:

```
<?php
$radio = 15;
$perimetro = 2 * 3.14 * $radio;
print $perimetro;
?>
```

```
94.2
```

En PHP una igualdad no es una ecuación matemática, sino una asignación (el resultado de la derecha se almacena en la variable de la izquierda). Por ello, una asignación pueden utilizar en el lado derecho la variable en la que se va a guardar el resultado:

```
<?php
$edad = 15;
$edad = 2 * $edad;
print $edad;
?>
```

```
30
```

Lo que hace el ordenador es calcular el resultado de la expresión de la derecha utilizando el valor actual de la variable y finalmente guardar el resultado en la variable.

En los nombres de las variables, PHP distingue entre mayúsculas y minúsculas, es decir, si se cambia algún carácter de mayúscula a minúscula o viceversa, para PHP se tratará de variables distintas.

```
<?php
$Edad = 15;
$edad = 20;
print $Edad;
?>
```

```
15
```

Una variable pueda tomar valores de tipos distintos a lo largo de un programa.

```
<?php
$edad = 15;
print "<p>$edad</p>\n";
$edad = "quince";
print "<p>$edad</p>\n";
```

```
?>
<p>15</p>
<p>quince</p>
```

CONCATENAR VARIABLES Y CADENAS

El operador . (Punto) permite concatenar dos o más cadenas o variables.

```
<?php
$cadena1 = "Pasa";
$cadena2 = "tiempos";
$cadena3 = $cadena1 . $cadena2;
print "<p>$cadena3</p>\n";
?>
<p>Pasatiempos</p>
```

```
<?php
$cadena1 = "Corre";
$cadena2 = "ve";
$cadena3 = "idile";
$cadena4 = $cadena1 . $cadena2 . $cadena3;
print "<p>$cadena4</p>\n";
?>
<p>Correveidile</p>
```

El operador . (punto) se puede utilizar en la instrucción print. En el ejemplo siguiente se concatenan una cadena, una variable y una cadena.

```
<?php
$nombre = "Don Pepito";
print "<p>¡Hola, " . $nombre . "! ¿Cómo está usted?</p>\n";
?>
<p>¡Hola, Don Pepito! ¿Cómo está usted?</p>
```

Nota: En el ejemplo anterior, se puede obtener el mismo resultado sin utilizar el operador . (punto), escribiendo la variable en la cadena:

```
<?php
$nombre = "Don Pepito";
print "<p>¡Hola, $nombre! ¿Cómo está usted?</p>\n";
?>
<p>¡Hola, Don Pepito! ¿Cómo está usted?</p>
```

En el ejemplo siguiente no queda más remedio que utilizar el operador . (punto) porque uno de los datos que se intercala es el resultado de una operación.

```
<?php
$semanas = 5;
print "<p>$semanas semanas son " . (7 * $semanas) . " días.</p>\n";
?>
<p>5 semanas son 35 días.</p>
```

En el interior de las cadenas no se realizan operaciones aritméticas.

```
<?php
$x = 3;
$y = 4;
print "<p>Suma: $x + $y = $x + $y</p>\n";
print "<p>Multiplicación: $x x $y = $x * $y</p>\n";
?>
```

```
<p>Suma: 3 + 4 = 3 + 4</p>
<p>Multiplicación: 3 x 4 = 3 x 4</p>
```

Si se quiere mostrar el resultado de operaciones matemáticas, es necesario efectuar las operaciones fuera de las cadenas. En algunos casos no es necesario escribir las operaciones entre paréntesis, pero se recomienda escribirlas siempre entre paréntesis para no tener que preocuparse por cuándo hacen falta y cuándo no.

```
<?php
$x = 3;
$y = 4;
print "<p>Suma: $x + $y = " . ($x + $y) . "</p>\n";
print "<p>Multiplicación: $x x $y = " . ($x * $y) . "</p>\n";
?>
```

```
<p>Suma: 3 + 4 = 7</p>
<p>Multiplicación: 3 x 4 = 12</p>
```

Si no se escriben paréntesis en las operaciones, pueden no producirse errores, pero el resultado puede ser distinto al esperado en algunos casos:

```
<?php
$x = 3;
$y = 4;

print "<p>Suma: $x + $y = " . $x + $y . "</p>\n";
print "<p>Multiplicación: $x x $y = " . $x * $y . "</p>\n";
?>
```

```
4</p>
<p>Multiplicación: 3 x 4 = 12</p>
```

Por supuesto, siempre se pueden utilizar definir auxiliares que guarden los resultados y utilizar las variables auxiliares en la cadena.

```
<?php
$x = 3;
$y = 4;
$suma = $x + $y;
$producto = $x * $y;
print "<p>Suma: $x + $y = $suma</p>\n";
print "<p>Multiplicación: $x x $y = $producto</p>\n";
?>
```

```
<p>Suma: 3 + 4 = 7</p>
<p>Multiplicación: 3 x 4 = 12</p>
```

UNIR VARIABLES Y TEXTO

En html/css a veces es necesario juntar números y caracteres, como en el ejemplo siguiente en el que se establece el tamaño del párrafo en 30px:

```php
<?php
print "<p style=\"font-size: 30px\">Texto grande</p>\n";
?>
```

```
<p style="font-size: 30px">Texto grande</p>
```

Si el tamaño está almacenado en una variable, no se puede juntar la variable con los caracteres ya que se interpretaría como una variable que no está definida y toma el valor vacío:

```php
<?php
$x = 30;

print "<p style=\"font-size: $xpx\">Texto grande</p>\n";
?>
```

```
<p style="font-size: ">Texto grande</p>
```

... pero se pueden utilizar llaves o sacar la variable de la cadena:

```php
<?php
$x = 30;
print "<p style=\"font-size: {$x}px\">Texto grande</p>\n";
?>
```

```
<p style="font-size: 30px">Texto grande</p>
```

```php
<?php
$x = 30;
print "<p style=\"font-size: " . $x . "px\">Texto grande</p>\n";
?>
```

```
<p style="font-size: 30px">Texto grande</p>
```

```php
<?php
$x = 30;
print "<p style=\"font-size: $x" . "px\">Texto grande</p>\n";
?>
```

```
<p style="font-size: 30px">Texto grande</p>
```

Dentro de las llaves no se pueden realizar operaciones. Si se realizan operaciones, se puede producir un error de sintaxis u obtener resultados no esperados:

```php
<?php
$x = 7;

print "<p>{$x} por {$x} es {$x * $x}</p>\n";
?>
```

Parse error: syntax error, unexpected '*', expecting :: (T_PAAMAYIM_NEKUDOTAYIM)
in **ejemplo.php** on line **4**

```php
<?php
$x = 7;

print "<p>2 por {$x} es {2 * $x}</p>\n";
?>
```

2 por 7 es {2 * 7}

Nota: T_PAAMAYIM_NEKUDOTAYIM es el nombre en hebreo del signo :: (dos puntos duplicados).

FUNCIONES DE MATRICES

GENERAR VALORES NUMÉRICOS EN SUCESIÓN ARITMÉTICA

La función **range($inicio, $final, $paso)** genera una matriz de valores en sucesión aritmética, es decir los valores desde $inicio hasta $final contando de $paso en $paso (sin superar el valor $final).

Nota: El valor $final de la sucesión se incluye en su caso en el resultado.

```php
<?php
$valores = range(1, 10, 3);

print "<pre>\n"; print_r($valores); print "</pre>\n";
?>
```

```
Array
(
    [0] => 1
    [1] => 4
    [2] => 7
    [3] => 10
)
```

```php
<?php
$valores = range(0.5, 1.5, 0.4);

print "<pre>\n"; print_r($valores); print "</pre>\n";
?>
```

```
Array
(
    [0] => 0.5
    [1] => 0.9
    [2] => 1.3
)
```

MEZCLAR MATRICES

La función **array_merge($matriz_1, $matriz_2, ...)** mezcla dos o más matrices en una única matriz.

Si los índices son numéricos, al mezclar las matrices se conservan todos los valores y se renumeran los índices.

```php
<?php
$uno = [1, 2, 4];
$dos = [1, 3, 9];

$final = array_merge($uno, $dos);
```

```php
print "<pre>\n"; print_r($final); print "</pre>\n";
?>
```

```
Array
(
    [0] => 1
    [1] => 2
    [2] => 4
    [3] => 1
    [4] => 3
    [5] => 9
)
```

```php
<?php
$uno = [1 => 1, 2 => 2, 3 => 4];
$dos = [1 => 1, 2 => 3, 3 => 9];

$final = array_merge($uno, $dos);

print "<pre>\n"; print_r($final); print "</pre>\n";
?>
```

```
Array
(
    [0] => 1
    [1] => 2
    [2] => 4
    [3] => 1
    [4] => 3
    [5] => 9
)
```

Pero si los índices no son numéricos, al mezclar las matrices los valores se sobrescriben.

```php
<?php
$uno = ["a" => 1, "b" => 2, "c" => 4];
$dos = ["a" => 2, "b" => 4, "c" => 6];

$final = array_merge($uno, $dos);

print "<pre>\n"; print_r($final); print "</pre>\n";
?>
```

```
Array
(
    [a] => 2
    [b] => 4
```

```
    [c] => 6
)
```

Si hay índices numéricos e índices no numéricos, al mezclar las matrices los valores de índices numéricos se conservan, pero los no numéricos se sobrescriben.

```php
<?php
$uno = [1 => 1, "b" => 2, "c" => 4];
$dos = [1 => 100, "b" => 4, "c" => 6];

$final = array_merge($uno, $dos);

print "<pre>\n"; print_r($final); print "</pre>\n";
?>
```

```
Array
(
    [0] => 1
    [b] => 4
    [c] => 6
    [1] => 100
)
```

CONTAR ELEMENTOS DE UNA MATRIZ

La función **count($matriz)** permite contar los elementos de una matriz.

```php
<?php
$nombres[1] = "Ana";
$nombres[10] = "Bernardo";
$nombres[25] = "Carmen";

$elementos = count($nombres);

print "<p>La matriz tiene $elementos elementos.</p>\n";
print "\n";
print "<pre>\n"; print_r($nombres); print "</pre>\n";
?>
```

La matriz tiene 3 elementos.

```
Array
(
    [1] => Ana
    [10] => Bernardo
    [25] => Carmen
)
```

En una matriz multidimensional, la función **count($matriz)** considera la matriz como un vector de vectores y devuelve simplemente el número de elementos del primer índice:

```php
<?php
$datos["pepe"]["edad"] = 25;
$datos["pepe"]["peso"] = 80;
$datos["juan"]["edad"] = 22;
$datos["juan"]["peso"] = 75;
$datos["ana"]["edad"]  = 30;

$elementos = count($datos);

print "<p>La matriz tiene $elementos elementos.</p>\n";
print "\n";
print "<pre>\n"; print_r($datos); print "</pre>\n";
?>
```

La matriz tiene 3 elementos.

```
Array
(
    [pepe] => Array
        (
            [edad] => 25
            [peso] => 80
        )

    [juan] => Array
        (
            [edad] => 22
            [peso] => 75
        )

    [ana] => Array
        (
            [edad] => 30
        )


)
```

Para contar todos los elementos de una matriz multidimensional, habría que utilizar la función **count($matriz, COUNT_RECURSIVE)**.

```php
<?php
$datos["pepe"]["edad"] = 25;
$datos["pepe"]["peso"] = 80;
$datos["juan"]["edad"] = 22;
$datos["juan"]["peso"] = 75;
$datos["ana"]["edad"]  = 30;
```

```php
$elementos = count($datos, COUNT_RECURSIVE);

print "<p>La matriz tiene $elementos elementos.</p>\n";
print "\n";
print "<pre>\n"; print_r($datos); print "</pre>\n";
?>
```

La matriz tiene 8 elementos.

```
Array
(
    [pepe] => Array
        (
            [edad] => 25
            [peso] => 80
        )

    [juan] => Array
        (
            [edad] => 22
            [peso] => 75
        )

    [ana] => Array
        (
            [edad] => 30
        )

)
```

En el ejemplo anterior, la respuesta 8 se debe a que la función **count()**recursiva considera la matriz como un vector de vectores y cuenta los elementos que hay en cada nivel. Desde ese punto de vista, la matriz contiene tres elementos que a su vez contienen dos, dos y un elementos, lo que da un total de ocho elementos.

Si quisiéramos contar únicamente los elementos de una matriz bidimensional habría que restar los dos resultados anteriores (5 = 8 - 3):

```php
<?php
$datos["pepe"]["edad"] = 25;
$datos["pepe"]["peso"] = 80;
$datos["juan"]["edad"] = 22;
$datos["juan"]["peso"] = 75;
$datos["ana"]["edad"]  = 30;

$elementos = count($datos, COUNT_RECURSIVE) - count($datos);
```

```
print "<p>La matriz tiene $elementos elementos.</p>\n";
print "\n";
print "<pre>\n"; print_r($datos); print "</pre>\n";
?>
```

La matriz tiene 5 elementos.

```
Array
(
    [pepe] => Array
        (
            [edad] => 25
            [peso] => 80
        )

    [juan] => Array
        (
            [edad] => 22
            [peso] => 75
        )

    [ana] => Array
        (
            [edad] => 30
        )

)
```

CONTAR CUÁNTAS VECES APARECE CADA VALOR EN UNA MATRIZ

La función **array_count_values($matriz)** cuenta cuántos valores hay de cada valor de una matriz.

La matriz devuelta tiene como índices los valores de la matriz original y como valores la cantidad de veces que aparece el valor.

```
<?php
$nombres = ["Ana", "Bernardo", "Bernardo", "Ana", "Carmen", "Ana"];

$cuenta = array_count_values($nombres);

print "<pre>\n"; print_r($cuenta); print "</pre>\n";
?>
```

La matriz tiene 3 elementos.

```
Array
(
```

```
    [Ana] => 3
    [Bernardo] => 2
    [Carmen] => 1
)
```

MÁXIMO Y MÍNIMO

La función **max($matriz, ...)** devuelve el valor máximo de una matriz (o varias). La función **min($matriz, ...)** devuelve el valor mínimo de una matriz (o varias).

```php
<?php
$numeros = [10, 40, 15, -1];

$maximo = max($numeros);
$minimo = min($numeros);

print "<pre>"; print_r($numeros); print "</pre>\n";
print "\n";
print "<p>El máximo de la matriz es $maximo.</p>\n";
print "\n";
print "<p>El mínimo de la matriz es $minimo.</p>\n";
?>
```

```
Array
(
    [0] => 10
    [1] => 40
    [2] => 15
    [3] => -1
)
```

El máximo de la matriz es 40.

El mínimo de la matriz es -1.

Los valores no numéricos se tratan como 0, pero si 0 es el mínimo o el máximo, la función devuelve la cadena.

```php
<?php
$valores = [10, 40, 15, "abc"];

$maximo = max($valores);
$minimo = min($valores);

print "<pre>\n"; print_r($valores); print "</pre>\n";
print "\n";
print "<p>El máximo de la matriz es $maximo.</p>\n";
print "\n";
print "<p>El mínimo de la matriz es $minimo.</p>\n";
?>
```

```
Array
(
    [0] => 10
    [1] => 40
    [2] => 15
    [3] => abc
)
```

El máximo de la matriz es 40.

El mínimo de la matriz es abc.

ORDENAR UNA MATRIZ

Existen varias funciones para ordenar matrices. Las más simples son las siguientes:

Cuando los índices de la matriz que vamos a ordenar no son importantes y se pueden modificar, podemos utilizar las funciones **sort($matriz, $opciones)** y **rsort($matriz, $opciones)**, que ordenan atendiendo únicamente a los valores de la matriz (no a sus índices), en orden creciente o decreciente, y re indexan la matriz:

- **sort($matriz, $opciones)**: ordena por orden alfabético / numérico de los valores y genera nuevos índices numéricos consecutivos a partir de cero:

```php
<?php
$valores = [5 => "cinco", 1 => "uno", 9 => "nueve"];

print "<p>Matriz inicial:</p>\n";
print "\n";
print "<pre>\n"; print_r($valores); print "</pre>\n";
print "\n";

sort($valores);

print "<p>Matriz ordenada con sort():</p>\n";
print "\n";
print "<pre>\n"; print_r($valores); print "</pre>\n";
print "\n";
?>
```

Matriz inicial:

```
Array
(
    [5] => cinco
    [1] => uno
    [9] => nueve
)
```

Matriz ordenada con sort():

```
Array
```

```
(
    [0] => cinco
    [1] => nueve
    [2] => uno
)
```

- **rsort($matriz, $opciones)**: ordena por orden alfabético / numérico inverso de los valores y genera nuevos índices numéricos consecutivos a partir de cero:

```php
<?php
$valores = [5 => "cinco", 1 => "uno", 9 => "nueve"];

print "<p>Matriz inicial:</p>\n";
print "\n";
print "<pre>\n"; print_r($valores); print "</pre>\n";
print "\n";

rsort($valores);

print "<p>Matriz ordenada con rsort():</p>\n";
print "\n";
print "<pre>\n"; print_r($valores); print "</pre>\n";
print "\n";
?>
```

Matriz inicial:

```
Array
(
    [5] => cinco
    [1] => uno
    [9] => nueve
)
```

Matriz ordenada con rsort():

```
Array
(
    [0] => uno
    [1] => nueve
    [2] => cinco
)
```

Cuando los índices de la matriz que vamos a ordenar son importantes y no se deben modificar, podemos ordenar atendiendo únicamente a los valores de la matriz u ordenar atendiendo únicamente a los índices de la matriz y ordenar en orden creciente o decreciente, por lo que PHP dispone de cuatro funciones:

- **asort($matriz, $opciones)**: ordena por orden alfabético / numérico de los valores

```php
<?php
$valores = [5 => "cinco", 1 => "uno", 9 => "nueve"];
```

```php
print "<p>Matriz inicial:</p>\n";
print "\n";
print "<pre>\n"; print_r($valores); print "</pre>\n";
print "\n";

asort($valores);

print "<p>Matriz ordenada con asort():</p>\n";
print "\n";
print "<pre>\n"; print_r($valores); print "</pre>\n";
print "\n";
?>
```

Matriz inicial:

```
Array
(
    [5] => cinco
    [1] => uno
    [9] => nueve
)
```

Matriz ordenada con asort():

```
Array
(
    [5] => cinco
    [9] => nueve
    [1] => uno
)
```

- **arsort($matriz, $opciones)**: ordena por orden alfabético / numérico inverso de los valores

```php
<?php
$valores = [5 => "cinco", 1 => "uno", 9 => "nueve"];

print "<p>Matriz inicial:</p>\n";
print "\n";
print "<pre>\n"; print_r($valores); print "</pre>\n";
print "\n";

arsort($valores);

print "<p>Matriz ordenada con arsort():</p>\n";
print "\n";
print "<pre>\n"; print_r($valores); print "</pre>\n";
print "\n";
?>
```

Matriz inicial:

```
Array
(
    [5] => cinco
    [1] => uno
    [9] => nueve
)
```

Matriz ordenada con arsort():

```
Array
(
    [1] => uno
    [9] => nueve
    [5] => cinco
)
```

- **ksort($matriz, $opciones)**: ordena por orden alfabético / numérico de los índices

```php
<?php
$valores = [5 => "cinco", 1 => "uno", 9 => "nueve"];

print "<p>Matriz inicial:</p>\n";
print "\n";
print "<pre>\n"; print_r($valores); print "</pre>\n";
print "\n";

ksort($valores);

print "<p>Matriz ordenada con ksort():</p>\n";
print "\n";
print "<pre>\n"; print_r($valores); print "</pre>\n";
print "\n";
?>
```

Matriz inicial:

```
Array
(
    [5] => cinco
    [1] => uno
    [9] => nueve
)
```

Matriz ordenada con ksort():

```
Array
(
    [1] => uno
```

```
    [5] => cinco
    [9] => nueve
)
```

- **krsort($matriz, $opciones)**: ordena por orden alfabético / numérico de los índices

```php
<?php
$valores = [5 => "cinco", 1 => "uno", 9 => "nueve"];

print "<p>Matriz inicial:</p>\n";
print "\n";
print "<pre>\n"; print_r($valores); print "</pre>\n";
print "\n";

krsort($valores);

print "<p>Matriz ordenada con krsort():</p>\n";
print "\n";
print "<pre>\n"; print_r($valores); print "</pre>\n";
print "\n";
?>
```

Matriz inicial:

```
Array
(
    [5] => cinco
    [1] => uno
    [9] => nueve
)
```

Matriz ordenada con krsort():

```
Array
(
    [9] => nueve
    [5] => cinco
    [1] => uno
)
```

Para ver la diferencia entre estas funciones, la tabla siguiente resume los ejemplos anteriores:

Matriz inicial	sort()	rsort()	asort()	arsort()	ksort()	krsort()
Array ([5] => cinco [1] => uno [9] => nueve)	Array ([0] => cinco [1] => nueve [2] => uno)	Array ([0] => uno [1] => nueve [2] => cinco)	Array ([5] => cinco [9] => nueve [1] => uno)	Array ([1] => uno [9] => nueve [5] => cinco)	Array ([1] => uno [5] => cinco [9] => nueve)	Array ([9] => nueve [5] => cinco [1] => uno)

BUSCAR UN VALOR EN UNA MATRIZ

La función booleana **in_array($valor, $matriz[, $tipo])** devuelve **true** si el valor se encuentra en la matriz. Si el argumento booleano **$tipo** es **true**, **in_array()** comprueba además que los tipos coincidan.

```
<?php
$valores = [10, 40, 15, -1];

print "<pre>\n"; print_r($valores); print "</pre>\n";

if (in_array(15, $valores)) {
  print "<p>15 está en la matriz \$valores.</p>\n";
} else {
  print "<p>15 no está en la matriz \$valores.</p>\n";
}
print "\n";

if (in_array(25, $valores)) {
  print "<p>25 está en la matriz \$valores.</p>\n";
} else {
  print "<p>25 no está en la matriz \$valores.</p>\n";
}
print "\n";

if (in_array("15", $valores, true)) {
  print "<p>\"15\" está en la matriz \$valores.</p>\n";
} else {
  print "<p>\"15\" no está en la matriz \$valores.</p>\n";
}
?>
```

```
Array
(
    [0] => 10
    [1] => 40
    [2] => 15
    [3] => -1
)
```

15 está en la matriz $valores.

25 no está en la matriz $valores.

"15" no está en la matriz $valores.

La función **array_search($valor, $matriz[, $tipo])** busca el valor en la matriz y, si lo encuentra, devuelve el índice correspondiente, pero si hay varios valores coincidentes sólo devuelve el primero que encuentra.

La función **array_keys($matriz[, $valor[, $tipo])** busca el valor en la matriz y, si lo encuentra, devuelve una matriz cuyos valores son los índices de todos los elementos coincidentes.

```
<?php
$valores = [10, 40, 15, 30, 15, 40, 15];
```

```php
print "<pre>\n"; print_r($valores); print "</pre>\n";
print "\n";

$encontrado = array_search(15, $valores);
print "<p>$encontrado</p>\n";
print "\n";

$encontrados = array_keys($valores, 15);
print "<pre>\n"; print_r($encontrados); print "</pre>\n";
?>
```

```
Array
(
    [0] => 10
    [1] => 40
    [2] => 15
    [3] => 30
    [4] => 15
    [5] => 40
    [6] => 15
)
2
Array
(
    [0] => 2
    [1] => 4
    [2] => 6
)
```

REINDEXAR UNA MATRIZ

La función **array_values($matriz)** devuelve los valores de una matriz en el mismo orden que en la matriz original, pero renumerando los índices desde cero

```php
<?php
$nombres = ["a" => "Ana", "b" => "Bernardo", "c" => "Carmen", "d" => "David"];

print "<p>Matriz inicial:</p>\n";
print "\n";
print "<pre>\n"; print_r($nombres); print "</pre>\n";
print "\n";

$reindexada = array_values($nombres);

print "<p>Matriz reindexada con array_values():</p>\n";
```

```
print "\n";
print "<pre>\n"; print_r($reindexada); print "</pre>\n";
print "\n";
?>
```

Matriz inicial:

```
Array
(
    [a] => Ana
    [b] => Bernardo
    [c] => Carmen
    [d] => David
)
```

Matriz reindexada con array_values():

```
Array
(
    [0] => Ana
    [1] => Bernardo
    [2] => Carmen
    [3] => David
)
```

BARAJAR LOS ELEMENTOS DE UNA MATRIZ

La función **shuffle($matriz)** baraja los valores de una matriz. Los índices de la matriz original se pierden, ya que se renumeran desde cero.

```
<?php
$numeros = [0, 1, 2, 3, 4, 5];

print "<p>Matriz inicial:</p>\n";
print "\n";
print "<pre>\n"; print_r($numeros); print "</pre>\n";
print "\n";

shuffle($numeros);

print "<p>Matriz barajada con shuffle():</p>\n";
print "\n";
print "<pre>\n"; print_r($numeros); print "</pre>\n";
print "\n";
?>
```

Matriz inicial:

```
Array
(
```

```
    [0] => 0
    [1] => 1
    [2] => 2
    [3] => 3
    [4] => 4
    [5] => 5
)
```

Matriz barajada con shuffle():

```
Array
(
    [0] => 3
    [1] => 1
    [2] => 5
    [3] => 0
    [4] => 4
    [5] => 2
)
```

```php
<?php
$numeros = ["a" => 1, "b" => 2, "c" => 3, "d" => 4];

print "<p>Matriz inicial:</p>\n";
print "\n";
print "<pre>\n"; print_r($numeros); print "</pre>\n";
print "\n";

shuffle($numeros);

print "<p>Matriz barajada con shuffle():</p>\n";
print "\n";
print "<pre>\n"; print_r($numeros); print "</pre>\n";
print "\n";
?>
```

Matriz inicial:

```
Array
(
    [a] => 1
    [b] => 2
    [c] => 3
    [d] => 4
)
```

Matriz barajada con shuffle():

```
Array
(
    [0] => 2
    [1] => 1
    [2] => 4
    [3] => 3
)
```

EXTRAER AL AZAR UN ELEMENTO DE UNA MATRIZ

La función **array_rand($matriz)** extrae al azar uno de los índices de la matriz.

```php
<?php
$cuadrados = [2 => 4, 3 => 9, 7 => 49, 10 => 100];

$indice = array_rand($cuadrados);

print "<p>$indice</p>\n";
?>
```

```
<p>10</p>
```

Una vez obtenido el índice se puede obtener el valor correspondiente de la matriz.

```php
<?php
$cuadrados = [2 => 4, 3 => 9, 7 => 49, 10 => 100];

$indice = array_rand($cuadrados);

print "<p>$indice - $cuadrados[$indice]</p>\n";
?>
```

```
<p>10 - 100</p>
```

ELIMINAR VALORES REPETIDOS

La función **array_unique($matriz)** devuelve una matriz en la que se han eliminado los valores repetidos.

```php
<?php
$inicial = [0, 0, 1, 3, 1, 3, 2, 1];
$final = array_unique($inicial);
print "<pre>\n"; print_r($final); print "</pre>\n";
?>
```

```
Array
(
    [0] => 0
    [2] => 1
    [3] => 3
    [6] => 2
)
```

OPERADORES

NÚMEROS

Los números decimales se escriben con punto (.), no con coma (,).

Los operadores aritméticos básicos son los siguientes:

Ejemplo	Nombre	Resultado
-$a	Negación	El opuesto de $a.
$a + $b	Suma	Suma de $a y $b.
$a - $b	Resta	Diferencia entre $a y $b.
$a * $b	Multiplicación	Producto de $a y $b.
$a / $b	División	Cociente de $a y $b.
$a % $b	Módulo	Resto de $a dividido por $b. **Cuidado**: Los números se convierten a enteros antes de efectuar el cálculo. Por ejemplo, 5 % 2.5 da como resultado 1 y no 0 porque calcula el resto de 5 entre 2, no de 5 entre 2.5.

Hay que tener en cuenta que en PHP un entero no puede ser arbitrariamente grande. A partir de cierto valor, que depende del sistema operativo, PHP convierte automáticamente los enteros en float, perdiéndose precisión. En un sistema de 32 bits, el valor máximo es 2147483647 (2^{31}-1).

Si se necesita trabajar con enteros mayores, es necesario utilizar las funciones bcmath.

OPERADORES DE INCREMENTO Y DECREMENTO

Ejemplo	Nombre	Efecto
++$a	Pre-incremento	Incrementa $a en uno, y luego devuelve $a.
$a++	Post-incremento	Devuelve $a, y luego incrementa $a en uno.
--$a	Pre-decremento	Decrementa $a en uno, luego devuelve $a.
$a--	Post-decremento	Devuelve $a, luego decrementa $a en uno.

La diferencia entre el pre-incremento y el post-incremento es que en el primer caso primero se incrementa la variable y después se utiliza y en el segundo primero se utiliza y después se incrementa.

```php
<?php
$valor = 9;
print "<p>" . ++$valor . "</p>\n";
?>
```

```
<p>10</p>
```

```php
<?php
$valor = 9;
print "<p>" . $valor++ . "</p>\n";
?>
```

```
<p>9</p>
```

El operador de incremento funciona también con caracteres, teniendo en cuenta que al incrementar el carácter 'z' se obtiene la cadena 'aa'. El operador de decremento no funciona con caracteres.

```php
<?php
$valor = "b";
print "<p>" . ++$valor . "</p>\n";
?>
```

```
<p>c</p>
```

```php
<?php
$valor = "z";
print "<p>" . ++$valor . "</p>\n";
?>
```

```
<p>aa</p>
```

```php
<?php
$valor = "a9z";
print "<p>" . ++$valor . "</p>\n";
?>
```

```
<p>b0a</p>
```

RESTO DE UNA DIVISIÓN

El operador **%** calcula el resto de una división entera

```php
<?php
print "<p>El resto de 17 dividido entre 3 es " . 17 % 3 . "</p>\n";
?>
```

```
<p>El resto de 17 dividido entre 3 es 2</p>
```

La función **fmod** calcula el resto de una división con números decimales

```php
<?php
print "<p>El resto de 17 dividido entre 3.1 es " . fmod(17, 3.1) . "</p>\n";
?>
```

```
<p>El resto de 17 dividido entre 3.1 es 1.5</p>
```

Hay que tener en cuenta que en PHP un entero no puede ser arbitrariamente grande. A partir de cierto valor, que depende del sistema operativo, PHP convierte automáticamente los enteros en float, perdiéndose precisión. En un sistema de 32 bits, el valor máximo es 2147483647 (2^{31}-1).

Si se necesita trabajar con enteros mayores, es necesario utilizar las funciones bcmath.

PARÉNTESIS

Los operadores aritméticos tienen el mismo orden de precedencia que en Matemáticas. Concretamente, el orden de precedencia de los operadores comentados anteriormente es, de mayor a menor, el siguiente:

- ++ (incremento) -- (decremento)
- (multiplicación) / (división) % (resto)
- + (suma) - (resta)

Los paréntesis permiten agrupar operaciones de manera que las operaciones entre paréntesis se realicen antes que las operaciones fuera de los paréntesis. Se aconseja no utilizar paréntesis cuando las operaciones den el mismo resultado con o sin paréntesis.

OPERADORES COMBINADOS

Los operadores combinados permiten simplificar algunas expresiones de asignación:

Ejemplo	Nombre	Equivale a
$a += $b	Suma	$a = $a + $b
$a -= $b	Resta	$a = $a - $b
$a *= $b	Multiplicación	$a = $a * $b
$a /= $b	División	$a = $a / $b
$a %= $b	Módulo	$a = $a % $b

REDONDEAR UN NÚMERO

La función **round(x)** redondea el número x al entero más próximo.

```
<?php
print "<p>2.6 se redondea con round() a " . round(2.6) . "</p>\n";
print "<p>2.3 se redondea con round() a " . round(2.3) . "</p>\n";
print "<p>-2.6 se redondea con round() a " . round(-2.6) . "</p>\n";
print "<p>-2.3 se redondea con round() a " . round(-2.3) . "</p>\n";
?>
```

```
<p>2.6 se redondea con round() a 3</p>
<p>2.3 se redondea con round() a 2</p>
<p>-2.6 se redondea con round() a -3</p>
<p>-2.3 se redondea con round() a -2</p>
```

La función **round(x,n)** redondea x con n decimales (si n es negativo redondea a decenas, centenas, etc.).

```
<?php
print "<p>2.6574 se redondea con round() con dos decimales a " . round(2.6574, 2) .
"</p>\n";
print "<p>3141592 redondeado con round() con centenas es " . round(3141592, -2) .
"</p>\n";
?>
```

```
<p>2.6574 se redondea con round() con dos decimales a 2.66</p>
<p>3141592 redondeado con round() con centenas es 3141600</p>
```

La función **floor(x)** redondea el número x al entero inferior (es decir, devuelve la parte entera).

```
<?php
print "<p>2.6 se redondea con floor() a " . floor(2.6) . "</p>\n";
print "<p>2.3 se redondea con floor() a " . floor(2.3) . "</p>\n";
print "<p>-2.6 se redondea con floor() a " . floor(-2.6) . "</p>\n";
print "<p>-2.3 se redondea con floor() a " . floor(-2.3) . "</p>\n";
?>
```

```
<p>2.6 se redondea con floor() a 2</p>
<p>2.3 se redondea con floor() a 2</p>
<p>-2.6 se redondea con floor() a -3</p>
<p>-2.3 se redondea con floor() a -3</p>
```

La función **ceil(x)** redondea el número x al entero superior.

```
<?php
print "<p>2.6 se redondea con ceil() a " . ceil(2.6) . "</p>\n";
print "<p>2.3 se redondea con ceil() a " . ceil(2.3) . "</p>\n";
print "<p>-2.6 se redondea con ceil() a " . ceil(-2.6) . "</p>\n";
print "<p>-2.3 se redondea con ceil() a " . ceil(-2.3) . "</p>\n";
?>
```

```
<p>2.6 se redondea con ceil() a 3</p>
<p>2.3 se redondea con ceil() a 3</p>
<p>-2.6 se redondea con ceil() a -2</p>
<p>-2.3 se redondea con ceil() a -2</p>
```

POTENCIAS

La función **pow(x, y)** calcula x elevado a y.

En PHP 5.6 se introdujo el operador **, equivalente a la función **pow()**.

```
<?php
print "<p>2<sup>3</sup> = " . pow(2, 3) . "</p>\n";
?>
```

```
<p>2 = 8</p>
```

```
<?php
print "<p>2<sup>3</sup> = " . 2 ** 3 . "</p>\n";
?>
```

```
<p>2 = 8</p>
```

MÁXIMO Y MÍNIMO

Las funciones **max()** y **min()** devuelven el máximo y el mínimo, respectivamente, de una lista o matriz de valores.

```
<?php
print "<p>El máximo es " . max(20, 40, 25.1, 14.7) . "</p>\n";
?>
```

```
<p>El máximo es 40</p>
```

```php
<?php
print "<p>El mínimo es " . min(20, 40, 25.1, 14.7) . "</p>\n";
?>
```

```
<p>El mínimo es 14.7</p>
```

```php
<?php
$datos = [20, 40, 25.1, 14.7];
print "<p>El mínimo es " . min($datos) . "</p>\n";
?>
```

```
<p>El mínimo es 14.7</p>
```

FORMATEAR UN NÚMERO

Para escribir un número con los símbolos de separación de decimales y de miles españoles, es decir, una coma (,) para separar la parte entera de la decimal y un punto (.) para separar las cifras de la parte entera en grupos de tres, se puede utilizar la función **number_format()**.

```php
<?php
print "<p>" . number_format(1300, 5) . "</p>\n";
?>
```

```
<p>1,300.00000</p>
```

```php
<?php
print "<p>" . number_format(123456.789, 2) . "</p>\n";
?>
```

```
<p>123,456.79</p>
```

```php
<?php
print "<p>" . number_format(123456789123, 0, ",", ".") . "</p>\n";
?>
```

```
<p>123.456.789.123</p>
```

```php
<?php
print "<p>" . number_format(123456789123456.789, 2, ",", ".") . "</p>\n";
?>
```

```
<p>123.456.789.123.456,78</p>
```

La función requiere dos o cuatro argumentos:

- el primer argumento es el número a formatear.
- el segundo argumento es el número de decimales a mostrar (el número se redondea o trunca dependiendo de la longitud del número, compárese el segundo y el cuarto ejemplo de los ejemplos anteriores).
- el tercer argumento es el carácter a utilizar como separador de la parte entera de la decimal.
- el cuarto argumento es el carácter a utilizar como separador de miles.

La función devuelve el número formateado. Si sólo se utilizan dos argumentos, se utiliza el punto como separador de parte entera y decimal y la coma como separador de miles (notación inglesa).

Para obtener un número entero aleatorio entre dos valores determinados, se pueden utilizar la función **rand()** o la función **mt_rand()** (más rápida, basada en el algoritmo Mersenne Twister). Ambas funciones requieren dos argumentos:

- El primer argumento es el valor mínimo que se quiere obtener
- El segundo argumento es el valor máximo que se quiere obtener.

```
<?php
print "<p>" . mt_rand(1, 6) . "</p>\n";
print "<p>" . mt_rand(1, 6) . "</p>\n";
print "<p>" . mt_rand(1, 6) . "</p>\n";
?>
```

```
<p>5</p>
<p>3</p>
<p>6</p>
```

```
<?php
print "<p>" . rand(-10, 10) . "</p>\n";
print "<p>" . rand(-10, 10) . "</p>\n";
print "<p>" . rand(-10, 10) . "</p>\n";
?>
```

```
<p>-8</p>
<p>-10</p>
<p>3</p>
```

Si se llama a las funciones sin argumentos, estas devuelven un número entero aleatorio entre 0 y un valor máximo que es el que devuelve la función **getrandmax()** o **mt_getrandmax()** (este valor depende del sistema operativo).

```
<?php
print "<p>" . mt_getrandmax() . "</p>\n";
print "<p>" . mt_rand() . "</p>\n";
print "<p>" . mt_rand() . "</p>\n";
?>
```

```
<p>2147483647</p>
<p>2023208892</p>
<p>800052801</p>
```

```
<?php
print "<p>" . getrandmax() . "</p>\n";
print "<p>" . rand() . "</p>\n";
print "<p>" . rand() . "</p>\n";
?>
```

```
<p>32767</p>
<p>12279</p>
<p>24164</p>
```

La función **rand()** puede devolver valores mayores que los que devuelve cuando se la llama sin argumentos, pero no se puede superar el mayor número entero que maneja PHP (PHP_INT_MAX).

```php
<?php
print "<p>" . PHP_INT_MAX . "</p>";
print "<p>" . rand(100000, 200000) . "</p>\n";
print "<p>" . mt_rand(100000000000, 200000000000) . "</p>\n";
?>
```

```
<p>2147483647</p>
<p>166031</p>
Warning: mt_rand(): max(-1863462912) is smaller than min(1215752192) in prueba.php on line 3
```

Las funciones **rand()** y **mt_rand()** generan números aleatorios que no se pueden considerar criptográficamente seguros. En PHP 7.0 se incorporó una nueva función, **random_int()**, que sí puede considerarse criptográficamente segura.