Unidad 3

Centro Don Bosco
Villamuriel de Cerrato

PHP (II). arrays, strings y fechas

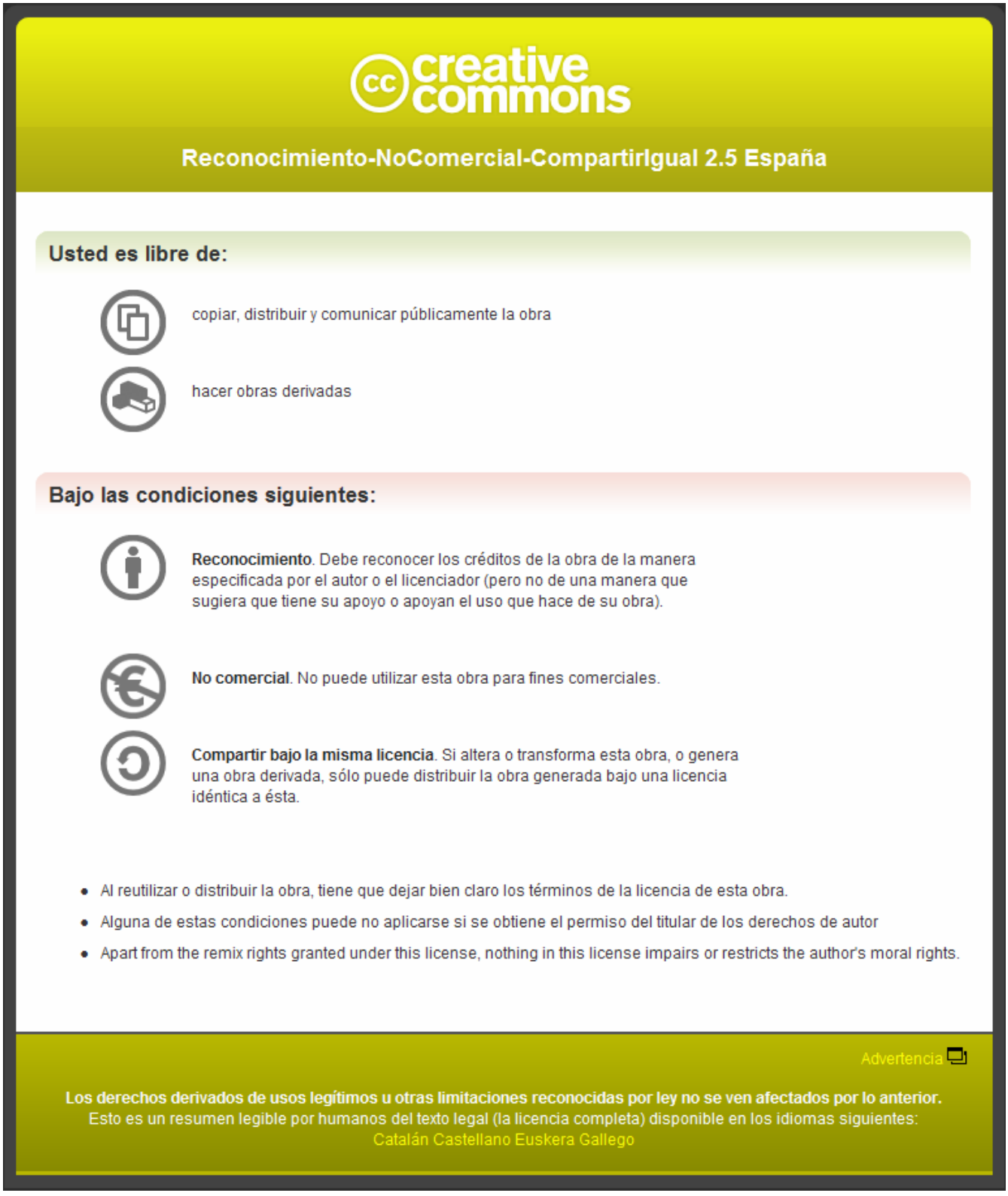
Apuntes realizados para la asignatura de FP Grado Superior:

Implantación de Aplicaciones Web

del ciclo Administración de Sistemas Informáticos en Red

Autor: Jorge Sánchez Asenjo (www.jorgesanchez.net)

Versión del documento: 2.03, Año 2013



Para ver una copia completa de la licencia, acudir a la dirección
http://creativecommons.org/licenses/by-nc-sa/2.5/es/legalcode.es

índice

(3.1) arrays ____ 5

(3.1.1) introducción a los arrays ... 5
(3.1.2) arrays escalares ... 6
(3.1.3) arrays asociativos ... 9
(3.1.4) bucle **foreach** ... 10
(3.1.5) arrays multidimensionales ... 11
(3.1.6) inspección de arrays mediante funciones de recorrido ... 13
(3.1.7) funciones y arrays ... 14
(3.1.8) funciones estándar de uso con arrays ... 14
(3.1.9) uso de arrays en formularios ... 22

(3.2) strings ____ 24

(3.2.1) introducción ... 24
(3.2.2) asignación de valores ... 24
(3.2.3) concatenación de textos ... 24
(3.2.4) uso de variables en strings. uso de llaves ... 25
(3.2.5) manejo de strings como arrays de caracteres ... 25
(3.2.6) cadenas **heredoc** ... 25
(3.2.7) cadenas **nowdoc** ... 26
(3.2.8) expresiones regulares ... 26
(3.2.9) funciones estándar de uso con strings ... 31

(3.3) funciones de fecha ____ 38

(3) PHP (II). estructuras de datos y sesiones

(3.1) arrays

(3.1.1) introducción a los arrays

Los tipos de datos que conocemos hasta ahora no permiten solucionar problemas que requieren gestionar muchos datos a la vez. Por ejemplo, imaginemos que deseamos almacenar las notas de una clase de 25 alumnos, no habrá más remedio que declarar 25 variables.

Eso es tremendamente pesado de programar. Manejar esos datos significaría estar continuamente manejando 25 variables. Por supuesto si necesitamos 2000 notas, el problema se hace inmanejable.

Por ello en casi todos los lenguajes se pueden agrupar una serie de variables del mismo tipo en una misma estructura que comúnmente se conoce como array[1]. Esa estructura permite referirnos a todos los elementos, pero también nos permite acceder individualmente a cada elemento.

Los arrays son una colección de datos del mismo tipo al que se le pone un nombre (por ejemplo ***nota***). Para acceder a un dato individual de la colección hay que utilizar su posición. La posición es un número entero que se llama **índice**, así para acceder a(por ejemplo ***nota[4]*** es el nombre que recibe el cuarto elemento de la sucesión de notas).

Hay que tener en cuenta que en los arrays el primer elemento tiene como índice el número cero. El segundo el uno y así sucesivamente; es decir ***nota[4]*** en realidad es el quinto elemento del array.

---

[1] Otros nombres habituales además de ***arrays*** son: listas, matrices, arreglos,… Estimo que **array** es el más aceptado en la actualidad, los otros términos son más ambiguos.

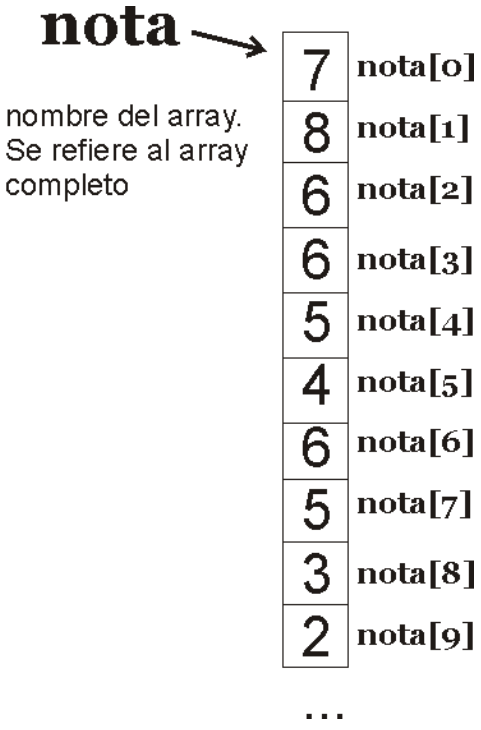


Ilustración 3-1, Ejemplo de array de notas

Esta definición de arrays es la común en todos los lenguajes clásicos. En el caso de PHP, los arrays son elementos más complejos, puesto que admiten datos de distinto tipo. La realidad es que los arrays de PHP son en realidad elementos que asocian un valor y una clave (arrays asociativos) como se explica más adelante.

(3.1.2) arrays escalares

Los arrays escalares cumplen la definición indicada anteriormente de array: es un conjunto de valores a los que se accede a través de un índice que es un número entero. Así, por ejemplo las notas se podrían almacenar de esta forma:

```
$nota[0]=5;
$nota[1]=9;
$nota[2]=8;
$nota[3]=5;
$nota[4]=6;
$nota[5]=7;
```

Acceder a una nota cualquiera, requiere conocer su índice.

No hace falta declarar el array (siguiendo el estilo habitual de PHP), simplemente se utiliza. Es posible incluso este código:

```
$valor[0] =18;
$valor[1]="Hola";
$valor [2]=true;
$valor[3]=3.4;
```

Es decir, los datos pueden ser heterogéneos. Sin indicar índice podemos:

```
$valor[] =18;
$valor[]="Hola";
$valor []=true;
$valor[]=3.4;
```

Código equivalente al anterior, ya que al no indicar índice se usa el siguiente índice libre en el array.

recorrer arrays

La gracia de los arrays en todos los lenguajes es la facilidad de rellenar o utilizar su contenido mediante bucles **for**. Imaginemos que necesitamos almacenar 1000 números aleatorios del 1 al 10; evidentemente sería terrible tener que escribir 1000 líneas de código para realizar esta labor:

```
$valor[0]=rand(1,10);
$valor[1]=rand(1,10);
$valor[2]=rand(1,10);
...//¡¡¡¡mucho trabajo!!!!
```

Lo lógico es utilizar este código:

```
for($i=0;$i<1000;$i++)
  $valor[$i]=rand(1,10);
```

La variable **$i** sirve para recorrer cada elemento del array. Va cambiando de 0 a 999 y así en dos líneas se rellena el array.

función count

La función **count** que posee PHP permite saber el tamaño de un array. Es útil para utilizar un array del que previamente no sabemos su tamaño.

Por ejemplo si deseamos recorrer un array que ya tiene valores para, por ejemplo, mostrar cada elemento en pantalla, el código sería:

```
for($i=0;$i<count($valor);$i++){
  echo $valor[$i]."<br />";
}
```

elementos vacíos

En este código:

```
$nota[0]=5;
$nota[1]=9;
$nota[3]=5;
$nota[4]=6;
$nota[5]=7;
```

Deliberadamente se ha dejado sin rellenar el elemento ***nota[2]***. De modo que la cuestión es ¿qué sacaría por pantalla la instrucción `echo $nota[2]`?

Normalmente un servidor Apache en el que se ejecute este código mostraría un mensaje parecido a este:

```
Notice: Undefined offset: 2 in E:\Mis
documentos\xampp\htdocs\arrays\arrayPrb1.php on line 18
```

Undefined offset se puede traducir como *desplazamiento indefinido* y además indica que dicho desplazamiento es el número *2*. Es decir, lo que Apache indica es que al moverse al elemento *2* del array se encuentra con que es una posición indefinida: dicho de otra forma, que es un índice de array que no existe.

La función **count** devuelve el tamaño del array sin tener en cuenta los elementos vacíos; es decir, sólo cuenta los elementos definidos. Por todo ello el código anterior:

```
for($i=0;$i<count($valor);$i++){
   echo $valor[$i]."<br />";
}
```

Puede fallar si el array *$valor* tiene elementos vacíos. Además los últimos elementos no saldrían por que el contador no llega a ellos al contar elementos y no los índices más altos.

Para evitar eso podemos utilizar la función **isset** que devuelve verdadero cuando a la variable que se le pasa como parámetro se le asignado ya un valor. El bucle quedaría entonces:

```
$tope=count($nota);
for($i=0;$i<$tope;$i++){
   if(isset($nota[$i]))
            echo $nota[$i]."<br />";
   else
            $tope++;
}
```

El código es más enrevesado. Ahora sólo se escribe cada elemento del array si la función **isset** nos asegura que dicho elemento tiene valor; la variable **$tope** va incrementando su valor a fin de alcanzar al último elemento real del array

uso del constructor array()

Los arrays pueden definirse de forma cómoda mediante la función **array()**. Dentro de esa función se colocan los valores que tendrá el array; y así en una sola línea podemos asignar todos los valores del array. Ejemplo:

```
$numero=array(17, 34, 45, 2, 9, -5, 7);
```

Esa línea asigna valores a cada elemento del array empezando por el cero. Es decir, es equivalente a:

```
$numero[0]=17;
$numero[1]=34;
$numero[2]=45;
$numero[3]=2;
```

```
$numero[4]=9;
$numero[5]=-5;
$numero[6]=7;
```

Esta función tiene posibilidad de elegir índices concretos. Eso se hace colocando el índice y después el símbolo => antes del valor. Ejemplo:

```
$numero=array(17,  34, 3=>45, 2, 7=>9, 10=>-5, 7);
```

Sería equivalente a:

```
$numero[0]=17;
$numero[1]=34;
$numero[3]=45;
$numero[4]=2;
$numero[7]=9;
$numero[10]=-5;
$numero[11]=7;
```

Los índices que no se indican siguen la serie natural

(3.1.3) arrays asociativos

En este caso el índice es un texto que se comporta como una especie de clave que permite acceder al dato. Ejemplo:

```
$nota["Antonio"]=5;
$nota["Luis"]=9;
$nota["Ana"]=8;
$nota["Eloy"]=5;
$nota["Gabriela"]=6;
$nota["Berta"]=7;
```

La ventaja de estos arrays es que son más legibles, la desventaja es que no permiten su manejo mediante los bucles clásicos.

Esos arrays también permiten el uso de la función **array()**:

```
$nota=array("Antonio"=>5,"Luis"=>9,"Ana"=>8,"Eloy"=>5,
   "Gabriela"=>6,"Berta"=>7);
```

Esta función incluso permite mezclar valores escalares y asociativos (aunque no es muy recomendable), por ejemplo:

```
$nota=array("Antonio"=>5,  4,  7, 5=>6, "Luis"=>9, 2,"Berta"=>7);
```

El ejemplo anterior sería equivalente a:

```
$nota["Antonio"]=5;
$nota[0]=4;
$nota[1]=7;
$nota[5]=6;
$nota["Luis"]=9;
$nota[6]=2;
$nota["Berta"]=7;
```

Otra cosa es la utilidad de esa sintaxis, que al final complica en demasía el código.

(3.1.4) bucle foreach

Recorrer arrays asociativos o incluso escalar con elementos si inicializar resulta muy complejo con los bucles **while** o **for**. Por ello PHP dispone de un bucle muy potente que permite recorrer todos los elementos de un array sin importar si hay saltos en la numeración o si se trata de un array asociativo. Se trata del bucle **foreach** que tiene esta sintaxis:

```
foreach(array as índice=>valor ){
    sentencias
}
```

array es el nombre del array que se va a recorrer; índice es la variable que recogerá el índice de cada elemento a medida que se recorra el array; y *valor* es la variable que irá recogiendo el valor de cada elemento del array. Ejemplo:

```
$nota=array("Antonio"=>5,  4,  7, 5=>6, "Luis"=>9, 2,"Berta"=>7);
foreach ($nota as $i=>$v){
    echo "La nota $i vale $v\n";
}
/* Escribe:
     La nota Antonio vale 5
     La nota 0 vale 4
     La nota 5 vale 6
     La nota Luis vale 9
     La nota 6 vale 2
     La nota Berta vale 7
*/
```

(3.1.5) arrays multidimensionales

Un array multidimensional sirve para representar datos agrupados en una estructura que utiliza dos índices para representar cada elemento del array. Ejemplos:

```
$nota[0][0]=9;
$nota[0][1]=7;
$nota[1][0]=9;
...
```

Por supuesto es posible utilizar cualquier tipo de valor tanto para los índices como para el valor:

```
$poblacion["Alemania"]["Berlin"]=4000000;
$poblacion["Alemania"]["Hanover"]=1200000;
$poblacion["Francia"]["Paris"]=7400000;
$poblacion["Francia"]["Lyon"]=2300000;
$poblacion["España"]["Palencia"]=80000;
```

Y se pueden utilizar más de dos dimensiones:

```
$poblacion["España"]["Castilla y León"]["Palencia"]=80000;
$poblacion["España"]["Castilla y León"]["Valladolid"]=370000;
$poblacion["España"]["Asturias"]["Oviedo"]=115000;
$poblacion["Alemania"]["Brandemburgo"]["Berlin"]=4000000;
```

De esa forma podemos considerar a los arrays como una especie de almacenes de datos que se parecen mucho a las tablas de las bases de datos relacionales; ese detalle facilita mucho la comunicación entre PHP y las bases de datos.

Otra forma posible:

```
$alemania=array("Berlin", "Hannover");
$francia=array("París", "Lyon");
$espania=array("Palencia");
    $poblacion=array("alemania"=>$alemania,"francia"=>$francia,
    "espania"=>$espania);
```

constructor array() en arrays multidimensionales

La función array() está pensada para arrays de una dimensión. Pero es posible utilizarla en arrays multidimensionales considerando que un array de dos dimensiones es un array de arrays. De esa forma:

```
$poblacion=array("España"=>array("Palencia"=>80000,"Valladolid"=>350000,
                                 "Oviedo"=>120000),
          "Francia"=>array("Paris"=>7000000,"Lyon"=>2100000)
   );
```

Es equivalente a:

```
$poblacion["España"]["Palencia"]=80000;
$poblacion["España"]["Valladolid"]=350000;
$poblacion["España"]["Oviedo"]=120000;
$poblacion["Francia"]["Paris"]=7000000;
$poblacion["Francia"]["Lyon"]=2100000;
```

recorrido mediante foreach de arrays multidimensionales

Nuevamente se basa la idea en considerar que un array multidimensional es un array de arrays. Así en el que caso de que con el array anterior hiciéramos este recorrido:

```
$poblacion=array("España"=>array("Palencia"=>80000,"Valladolid"=>350000,
                                 "Oviedo"=>120000),
          "Francia"=>array("Paris"=>7000000,"Lyon"=>2100000)
   );
foreach($poblacion as $i=$valor){
   echo "El índice $i vale $valor<br />";
}
```

El resultado es:

```
El índice España vale Array
El índice Francia vale Array
```

Lo que nos indica el resultado es que el valor de cada elemento es un array, con lo cual el código que realmente nos permite examinar cada valor sería:

```
$poblacion=array("España"=>array("Palencia"=>80000,"Valladolid"=>350000,
                                 "Oviedo"=>120000),
   "Francia"=>array("Paris"=>7000000,"Lyon"=>2100000)
      );
foreach ($poblacion as $pais => $ciudades) {
  foreach ($ciudades as $ciudad => $valor) {
    echo "<p>País: $pais, Ciudad: $ciudad, población = $valor</p>";
  }
}
```

Que obtiene el resultado:

```
País: España, Ciudad: Palencia, población = 80000
País: España, Ciudad: Valladolid, población = 350000
País: España, Ciudad: Oviedo, población = 120000
País: Francia, Ciudad: Paris, población = 7000000
País: Francia, Ciudad: Lyon, población = 2100000
```

(3.1.6) inspección de arrays mediante funciones de recorrido

Se trata de funciones basadas en el manejo de punteros de los lenguajes tradicionales. Utilizan un puntero virtual, que sería un objeto que señala a uno de los elementos del array y que al moverle permite acceder al resto de elementos.

Esta técnica se basa en el manejo de las siguientes funciones:

función	uso
current(array)	Devuelve el valor del elemento al que actualmente señala el puntero. Si no hay ningún elemento, devuelve false.
key(array)	Devuelve la clave a la que señala el puntero
next(array)	Mueve el puntero al siguiente elemento del array y devuelve su valor. Si el elemento actual es el último, next devuelve false.
prev(array)	Mueve el puntero al elemento anterior del array y devuelve su valor. Si el elemento actual es el primero, prev devuelve false.
reset(array)	Coloca el puntero en el primer elemento del array y devuelve su valor. Si no hay ningún elemento en el array, devuelve false.
end(array)	Coloca el puntero en el último elemento y devuelve su valor.

Ejemplo de recorrido de un array mediante estas funciones:

```
$capital=array(  "Castilla y León"=>"Valladolid","Asturias"=>"Oviedo",
                 "Aragón"=>"Zaragoza");
while(current($capital)){
   echo "<strong>".current($capital)."</strong><br />";
   next($capital);
}
```

Para recorrerle al revés:

```
$capital=array(  "Castilla y León"=>"Valladolid","Asturias"=>"Oviedo",
                 "Aragón"=>"Zaragoza");
end($capital);
while(current($capital)){
   echo "<strong>".current($capital)."</strong><br />";
   prev($capital);
}
```

(3.1.7) funciones y arrays

Las funciones pueden recibir y devolver arrays al igual que cualquier otro tipo de variables.

A diferencia de la mayoría de lenguajes estructurados, en PHP los arrays se pasan por valor a las funciones. Es decir cuando se pasa un array a una función, ésta recibe una copia del mismo. Así las modificaciones al array que se hagan dentro de la función, no afectan al array original. Ejemplo:

```
function prueba($a){
    $a[0]=18;
}

$array=array(1,2,3,4,5,6,7);
prueba($array);
echo $array[0];
```

La función *prueba* modifica el elemento con índice cero en el array para darle el valor 18. Sin embargo al ejecutar el código PHP anterior, veremos el número *1* en pantalla. La modificación de la función prueba se ha hecho con el array *$a* que es una copia en realidad del original *$array*.

Si deseáramos que realmente la función modifique el array, necesitamos pasar el array por referencia, como ya se ha visto en PHP basta con indicar el símbolo & delante del parámetro (o parámetros) que deseamos pasar por referencia:

```
function prueba(&$a){
    $a[0]=18;
}
$a=array(1,2,3,4,5,6,7);
prueba($a);
echo $a[0];
```

Ahora sí escribe *18*.

(3.1.8) funciones estándar de uso con arrays

funciones básicas

función	significado
count($array)	número de elementos del array
print_r($array)	Escribe el contenido del array (tanto valores como índices)
is_array(array)	Devuelve verdadero si el parámetro que recibe es un array

ordenación de arrays

función	significado
sort(array, [flags])	Ordena el array indicado. Si no se indica el segundo parámetro, ordena de forma normal sin convertir los tipos de datos. El segundo parámetro permite establecer la forma de ordenación pudiendo indicar estas posibilidades: • SORT_REGULAR. Ordenación normal (utilizada por defecto), se adapta según el valor sea numérico o String y no usa ordenación basado en alfabetos distintos del inglés. • SORT_NUMERIC. Ordena entendiendo que el array contiene valores numéricos • SORT_STRING. Ordena entendiendo que el array contiene valores String (texto) • SORT_LOCALE_STRING. Ordena basándose en la configuración regional establecida con la función setlocale. De modo que setlocale(LC_ALL,"es_ES") establecería como configuración regional, español de España. • SORT_NATURAL. Ordena el array considerando sus valores como lo haría un ser humano (es decir el texto *"texto2"* iría delante de *"texto10"* porque entendería el diez como un número superior al dos, y no ordenaría como si fueran textos alfabéticos. El funcionamiento es el mismo que la función natsort • SORT_FLAG_CASE Permite combinar (usando el operador OR (|) a nivel de bits) con SORT_STRING o SORT_NATURAL para ordenar cadenas de forma insensible a mayúsculas/minúsculas. Por ejemplo: `sort($array,SORT_STRING | SORT_NATURAL)` Al ordenar de esta forma, los índices desaparecen, el array final es un array escalar (el primer índice será el cero, luego el uno, etc.)
rsort(array, [flags])	Igual que la anterior, pero ordena en orden descendente.
asort(array, [flags])	Idéntica a sort, pero los índices no se desprecian y se respetan. Es decir se ordenan los valores y los índices se recolocan junto a la posición de su valor correspondiente
arsort(array, [flags])	Como la anterior pero el orden es descendente
ksort(array, [flags])	Idéntica a sort, pero se ordenan las claves. La relación de la clave junto con su valor correspondiente se respeta.
krsort(array, [flags])	Como la anterior pero el orden es descendente

función	significado
usort(array, funcionUsuario)	Ordena el array utilizando una función de usuario. Dicha función se debe crear de forma que reciba dos parámetros y devuelva un número mayor de cero cuando el primer parámetro sea mayor que cero, cero cuando sean iguales y un número menor que cero cuando el segundo parámetro sea mayor que el primero. **usort** recibe el nombre de la función entre comillas. Ejemplo: `usort($array,"funcion1")` Se supone que la ***funcion1*** ha sido creada previamente y cumple los requisitos indicados anteriormente Esto permite crear ordenaciones personales con los arrays. Las claves se pierden (al igual que ocurre con la función **sort**) y sólo se ordenan los valores, resultando un array escalar.
uasort(array, funcionUsuario)	Igual que la anterior, pero se respeta la relación de las claves con sus valores.
uksort(array, funcionUsuario)	Igual que la anterior, pero lo que ordena son las claves en lugar de los valores.
array_multisort(array1, arg1 ó array2,…)	Permite ordenar varios arrays al mismo tiempo, indicando cada array y (opcionalmente) la forma de ordenar. La forma de ordenar se indica mediante las palabras **SORT_ASC** (orden ascendente), **SORT_DESC** (orden descendente), **SORT_REGULAR** (orden normal), **SORT_NUMERIC** (ordenación numérica), **SORT_STRING** (ordenación para texto). Ejemplo: `array_multisort($a,SORT_DES, SORT_NUMERIC, $b,SORT_STRING)` En el ejemplo, el primer array (***$a***) se ordena en descendente y de forma numérica, mientras que el segundo (***$b***) se ordena de forma textual.
natsort(array)	Ordena el array de modo que se ordena en la forma en la que lo haría un ser humando. Es decir el texto ***imagen2*** iría antes que ***imagen10***. Se respeta la relación entre clave y valor, por lo que no se pierden las claves.
natcasesort(array)	Igual que la anterior, pero no distingue entre mayúsculas y minúsculas.

búsquedas y filtros en los arrays

función	significado
array_search(valorBusq,array [,estricto])	Busca el valor indicado en el array y devuelve la clave del valor en el array o **false** si no lo encuentra. El tercer parámetro (***stricto***) es opcional y en caso de valer verdadero (por defecto vale falso), sólo encuentra el valor si tiene el mismo valor en el array y además es del mismo tipo.
in_array(valorBusq,array [,estricto])	Busca el valor indicado en el array, de forma estricta o no (ver función anterior) y devuelve **true** si lo encuentra o **false** en caso contrario.
array_key_exists(array, clave)	Devuelve verdadero si en el array existe la clave indicada.
preg_prep(expresionRegular,array[,flag])	Devuelve un array que contiene los valores del array que cumplen la expresión regular. En el apartado (3.2.8), página 26, se explican las expresiones regulares. Si se usa el tercer parámetro con la constante **PREG_GREP_INVERT,** el array resultado contiene los elementos que no cumplen la expresión regular.
array_filter(array,función)	El segundo parámetro es el nombre de una función existente indicada entre comillas. Dicha función se creará de modo que tenga un solo parámetro y devuelva verdadero o falso en base a una condición que se valorará en el parámetro. **array_filter** aplica dicha función a cada elemento del array y devuelve un nuevo array que contiene los elementos del primer array a los que, pasándolos como parámetro a la función, ésta devuelva falso. Es decir elimina todos los elementos que no cumplan la condición establecida por la función.

funciones para manejo aleatorio

función	significado
shuffle(array)	Reordena de forma aleatoria el array
array_rand(**array**)	Devuelve un índice aleatorio del array

funciones de manipulación del contenido del array

función	significado
array_keys($array)	Devuelve un array escalar que contiene todas las claves del array
array_values($array)	Devuelve un array escalar en el que están los valores del array (los índices se retiran y se cambian por escalares)
array_flip($array)	Intercambia las claves por los valore en el array. Es decir: las claves pasan a ser valores y los valores las claves de dichos valores.

función	significado
array_combine(array1, array2)	Toma los valores de ambos arrays y devuelve un nuevo array donde las claves son los valores del primer array y los valores son los del segundo array. Ambos arrays deben de tener el mismo número de valores (de otro modo ocurrirá un error). Las claves de ambos arrays se ignoran en el resultado final.
array_fill(inicio,n,valor)	Devuelve un array que contiene ***n*** elementos, todos ellos con el valor indicado. El parámetro ***inicio*** indica cuál será el primer índice del array (es numérico); los siguientes índices serán los números consecutivos al inicial.
array_fill_keys(claves,valor)	Devuelve un array a partir de otro (parámetro ***claves***) array cuyos valores se considerarán las claves del nuevo. Todos los elementos del nuevo array valdrán el valor indicado.
array_unique(array, **[**flags**]**)	Elimina los valores duplicados en el array. El segundo parámetro sirve para indicar la forma de comparar los valores y utiliza las mismas posibilidades que las indicadas en la función **sort**.

array_pad(array,n,valor)	Devuelve un nuevo array copia del primero, pero en el que se rellena del valor indicado hasta alcanzar el tamaño indicado por ***n***. Si ***n*** es un número positivo el relleno se hace hacia la derecha y si no, hacia la izquierda. Ejemplo:
	
	`$a=array("a","b","c");`
	`$b=`**`array_pad`**`($a, 5, "x)`
	
	El array b será: ***a, b, c, x, x***

array_slice(array,posInicio[,posFin])	Devuelve una porción del array que se indica de modo que se toman los elementos desde la posición inicial indicada, hasta la posición final indicada. Si el parámetro ***posFin*** se omite, entonces se toma desde la posición inicial hasta el último elemento del array. Las posiciones indicadas se toman de forma escalar, es decir el primer elemento tiene la posición cero, el segundo uno,… El array original no cambia, pero en el resultante los índices nuevos será escalares; es decir, comienzan a numerar los índices desde el cero.

función	significado
array_splice(array,posInicio[,tamaño [,arraySubst]])	Elimina los elementos del array usado como primer parámetro de la función desde la posición marcada mediante el parámetro ***posInicio*** hasta el final. Este parámetro puede ser negativo y entonces la posición (***posInicio***) desde la que eliminar se cuenta desde el final. A la hora de indicar la ***posInicio***, se maneja el array como si fuera escalar. Es decir el primer elemento es el cero, el segundo es el uno,… Devuelve un array con los elementos eliminados. Si se usa el parámetro ***tamaño*** entonces se recorta el número indicado por ese parámetro (en lugar de llegar hasta el final). El parámetro ***arraySubst*** sirve para indicar un array que contiene los elementos con los que se sustituirán los eliminados.
array_count_values(array)	Genera un nuevo array en el que los índices son los valores anteriores y el valor es el número de veces que cada valor antiguo aparecía en el array original
array_sum(array)	Suma todos los valores del array y retorna el resultado
array_product(array)	Multiplica todos los valores del array y retorna el resultado
array_shift(array)	Retira el primer elemento del array, desplazando a todos los demás elementos en el mismo. Es decir, no deja el hueco del eliminado. Los índices se ponen de forma escalar, es decir comienzan a numerarse desde el número cero.
array_unshift(array,valor1,valor2,…)	Coloca al principio del array los valores indicados
array_pop(array)	Retira del array a su elemento y lo devuelve como resultado
array_push(array,valor1, valor2,…)	Coloca los valores indicados (al menos uno) al final del array original. Combinado con array_pop permite simular pilas
array_map(función, array)	Indica una función existente (pero entrecomillada) y devuelve un array resultado de aplicar la función indicada a cada elemento del array. La función se debe definir usando un solo parámetro, ese parámetro representa a cada elemento del array.

función	significado

array_walk(array, función)	Indica una función existente (pero entrecomillada) y la aplica a cada elemento del array. La función al definirla usará un solo parámetro que representa a cada elemento del array. Dicho elemento se define por referencia al crear la función (es decir utiliza el operador &) de ese modo la función realmente podrá modificar el valor de cada elemento..
	Ejemplo:
	
	`function doble(&$x){`
	`  $x*=2;`
	`}`
	`$array=array(1,2,3,4,5);`
	`array_walk($array,"doble");`
	`print_r($array);`
	
	Escribiría: *2,4,6,8,10*

array_walk_recursive(array, función)	Indica una función existente (pero entrecomillada) y la aplica a cada elemento del array. La función al definirla usará un solo parámetro que representa a cada elemento del array. En el caso de que los valores del array sean otros arrays, se ejecutará la función para dichos arrays.
array_chunk(array,tamaño [,respetarClaves])	Devuelve un array multidimensional, resultado de dividir el array que se pasa como parámetro en trozos del tamaño indicado. El parámetro opcional **respetarClaves**, si vale **true** (por defecto es **false**) respeta las claves originales; de otro modo las claves se pierden y cada array se renumera de forma escalar.

combinación de arrays

función	significado

array_merge($array1,$array2,...)	Une los dos arrays que se le pasan los valores del segundo van a continuación de los del primero. Si el array es asociativo, en caso de repetir claves, sólo se queda con las últimas.
	`$a=array("uno"=>"Pedro","dos"=>"Antonio","tres"=>"Santiago");`
	`$b=array("dos"=>"Sara","tres"=>"Antonio","cuatro"=>"Santiago");`
	`$c=array_merge($a,$b);`
	`print_r($c);`
	`/* obtiene:`
	`Array`
	`(`
	`[uno] => Pedro`
	`[dos] => Sara`
	`[tres] => Antonio`
	`[cuatro] => Santiago`
	`)`
	Si el array es escalar se renumeran de nuevo todas las claves y sí aparecen las claves con índices repetidos.

array_merge_recursive($a1,$a2,...)	Igual que el anterior, sólo que ahora si hay índices repetidos, en el array resultante se convertirán en un array que combina todos los valores de ese índice.
array_intersect($array1,$array2,...)	Genera un nuevo array, intersección de los indicados. En el array resultante sólo aparecen los valores duplicados en todos los arrays. Se mantienen los índices, pero toma los primeros índices que parezcan en la lista de arrays.
array_intersect_key($array1,$array2,...)	Genera un nuevo array en el que aparecen las claves presentes en todos los arrays indicados como parámetros.
array_diff($array1,$array2,...)	Genera un nuevo array en el que aparecen los valores del primer array que no están en el segundo (array de diferencia). La relación entre clave y valor se mantiene.
array_diff_key($array1,$array2,...)	Genera un nuevo array en el que aparecen las claves del primer array que no están en el segundo. La relación entre clave y valor se mantiene.

conversión de matrices a partir de variables comunes y viceversa

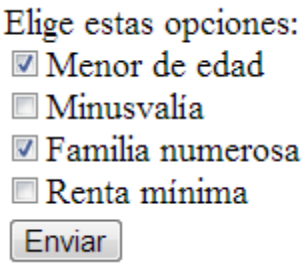
función	significado
compact(lista de variables)	Se le pasa una lista de variables. La lista es en realidad una lista de textos (strings) que contienen el nombre de las variables. La función devuelve un array con esos valores
list(listaDeVariables)=array	list no es una función, realmente permite asociar valores de un array a una lista de variables. Ejemplo: $a=array(979797979,"Manuel",1250.45); list($tlfno,$nombre,$salario)=$a; Las variables *$tlfno*, *$nombre* y *$salario* tomarán cada una el valor que les corresponda del array.

(3.1.9) uso de arrays en formularios

Una de las virtudes de un array, es su capacidad de recoger en una sola variable, valores procedentes de diferentes controles en un formulario. La forma es muy sencilla, basta con indicar un nombre de array como atributo **name** en los elementos del formulario; para indicar que ese nombre es de un array, se añaden la apertura y cierre de corchetes **[** y **]**. Ejemplo:

```
<!DOCTYPE html>
<html lang="es-ES">
  <head>
    <meta charset="UTF-8">
      <title></title>
  </head>
  <body>
    <form action="formArrayget.php" method="GET">
      Elige estas opciones:<br />
      <input type="checkbox" name="opciones[]" value="menor" />
          Menor de edad <br />
      <input type="checkbox" name="opciones[]" value="minusvalia" />
          Minusvalía<br />
      <input type="checkbox" name="opciones[]" value="numerosa" />
          Familia numerosa <br />
      <input type="checkbox" name="opciones[]" value="minima" />
          Renta mínima<br />
      <input type="submit" values="Enviar" />
    </form>
  </body>
</html>
```

La página resultante es (suponiendo que hemos marcado después algunas opciones):



En esta página, todos los checkbox están asociados a un array llamado ***opciones***. Dicho array contendrá un valor por cada elemento al que le hayamos hecho clic. Los índices se indican de forma escalar, es decir el primer valor (en el orden de escritura de la página web) será el cero, luego el uno,…

De modo que si el código de la página que recoge el array (***formArrayget.php***) es:

```
<!DOCTYPE html>
<html lang="es-ES">
   <head>
      <meta charset="UTF-8">
         <title></title>
   </head>
   <body>
      <?php
         print_r($_GET["opciones"]);
      ?>
   </body>
</html>
```

La salida de dicha página tras recoger los valores del formulario anteriormente comentados es:

```
Array ( [0] => menor [1] => numerosa )
```

Indicando que el array ***opciones*** (presente en el array de recogida del formulario **$_GET**) tiene dos elementos (con índices 0 y 1) con las valores ***menor*** y ***numerosa*** (correspondientes a los valores de los controles checkbox del formulario).

(3.2) strings

(3.2.1) introducción

La palabra String en inglés significa *cadena* y por eso es muy habitual en muchos libros de informática referirse así a este tipo de dato. Sin embargo en realidad por String se entiende a una serie de caracteres, lo que normalmente llamamos simplemente texto.

¿Por qué no llamarlas simplemente textos? Porque no todos los strings son textos. Es decir, hay series de caracteres que no representan necesariamente textos. Esto por ejemplo: *!"·$%&/()=?* es un String en cuanto a qué es una serie de caracteres, pero no diríamos que es un texto al no ser entendible.

En toda aplicación informática los strings son el tipo de dato fundamental. Tanto es así que en realidad ya hemos usado strings en los apartados anteriores. Sin embargo en este punto vamos a desglosar todo el manejo de strings que permite PHP.

(3.2.2) asignación de valores

Para asignar valores a un **string** basta con utilizar el operador **=**. A continuación se indica el texto entre comillas dobles o simples:

```
$string1="Soy un texto";
$string2='Yo también';
```

En el caso de los caracteres especiales, éstos sólo se pueden indicar dentro de comillas dobles.

(3.2.3) concatenación de textos

Como se ha comentado en apartados anteriores, el operador de concatenación de textos es el símbolo del punto (**.**).

Ejemplos:

```
$texto1="Hola";
$texto2="a todos y todas";
$texto3=$texto1." ".$texto2;
echo $texto3;
```

Por pantalla saldría *Hola a todos y todas*. Como se observa en el ejemplo se pueden concatenar tanto variables de tipo strings como textos literales (se encadena el contenido de la variable *$texto* se añade un espacio en blanco y se concatena con el contenido de la variable *texto2)*.

Podemos usar la concatenación y asignación:

```
$texto1="Hola ";
$texto1.="a todos y todas";
echo $texto1;
```

El resultado es como en el texto anterior ya que el símbolo **.=** permite añadir al final de la variable el String que se indique.

(3.2.4) uso de variables en strings. uso de llaves

Una de las grandes capacidades PHP es la de poder traducir contenidos de variables por su valor dentro de un texto. Ejemplo:

```
$libre="free";
echo "En inglés libre se dice $libre"; //muestra: En inglés libre se dice free
```

Sin embargo, este código daría error:

```
echo "<br />Me gusta patinar en $librestyle";
```

ya que no existe la variable *$librestyle* (ya que no entiende que haya que considerar *$libre* y luego *style*.

Sin embargo sí funciona mediante:

```
echo "<br />Me gusta patinar en {$libre}style";
//Sale: Me gusta patinar en freestyle
```

las llaves son además imprescindibles para utilizar con arrays multidimensionales; aunque, en cualquier caso el operador de concatenación de texto puede ayudarnos con expresiones más complejas.

```
echo "El doble de x es".x*2."<br />";
```

(3.2.5) manejo de strings como arrays de caracteres

Se puede….. Ejemplo:

```
$string1="Este es el texto de prueba";
$string1[6]="X";
echo $string1; //Sale: Este eX el texto de prueba
```

Aunque realmente no se considera un array; de hecho no se admite el uso de foreach en strings.

(3.2.6) cadenas heredoc

Permiten asignar valores de texto sin usar comillas. Ejemplo:

```
        $nombre="Jorge";
        $texto=<<<fin
        Mi querida amiga <br />
        escribo estas líneas esperando que me leas. <br />
        Firmado: $nombre <br />
fin;
        echo $texto;
```

En el ejemplo se ha coloreado de color rojo el texto que se almacena en la variable *$texto*. El texto a asignar es el que sigue al símbolo de inserción de documento (<<<) y

al **marcador de texto**, que es un grupo de caracteres concreto (en este caso se ha usado la palabra ***fin*** como marcador de texto). El marcador se indica inmediatamente después del símbolo <<< y vuelve a aparecer en la primera columna tras el último carácter que se almacenará en la variable. Es decir, escribe:

```
        Mi querida amiga
        escribo estas líneas esperando que me leas.
        Firmado: Jorge
```

(3.2.7) cadenas nowdoc

Funciona igual que el anterior, sólo que entiende que el texto está encerrado entre comillas, por lo que no se interpretan los nombres de variable ni los códigos de escape. Para diferenciar las dos notaciones, en ésta el marcador de línea va entre comillas simples (en la declaración, no en el cierre):

```
        $nombre="Jorge";
        $texto=<<<'fin'
        Mi querida amiga <br />
        escribo estas líneas esperando que me leas. <br />
        Firmado: $nombre <br />
fin;
        echo $texto;
```

Resultado:

```
Mi querida amiga
escribo estas líneas esperando que me leas.
Firmado: $nombre
```

No se ha interpretado la variable ***$nombre***, se ha entendido que es un texto normal.

(3.2.8) expresiones regulares

Uno de los usos más habituales en la mayoría de lenguajes de programación, tiene que ver con expresiones regulares.

Se utilizan, sobre todo, para establecer un patrón que sirva para conseguir que ciertos textos le cumplan. Ese patrón permite búsquedas avanzadas, criterios avanzados de verificación de claves o códigos (por ejemplo números de serie de productos que cumplen unas reglas muy concretas), etc.

PHP permite el uso de dos tipos de expresiones regulares:

- **POSIX (Portable Operating System Interface) 1003.2** correspondientes a un estándar aceptado por el organismo IEEE (***The Institute of Electrical and Electronics Engineers***) muy influyente en normas electrónicas y que se basa en la sintaxis del sistema **Unix**. Las funciones compatibles con este formato comienzan por la palabra **ereg**.
- **PCRE (Perl Compatible Regular Expressions)**. Parte de PHP desde la versión 4.2, procedente del lenguaje **Perl** (precisamente famoso por su uso de las expresiones regulares). Es el formato que más se usa, de hecho se recomienda no utilizar el anterior. Las funciones que utilizan este formato de expresión regulares comienzan por la palabra **preg**. Por otro lado este formato es

compatible con Unicode, por lo que también es más recomendable para lenguas que usan símbolos fuera del ASCII original como el castellano.

formato de las expresiones regulares PCRE

Las expresiones regulares utilizan símbolos especiales para indicar el patrón correspondiente. Las expresiones regulares de tipo Perl deben ir delimitadas por la barra de dividir /. Los símbolos y patrones que se pueden utilizar son:

símbolo	significado
c	Si *c* es un carácter cualquiera (por ejemplo *a*, *H*, *ñ*, *etc.*) indica, donde aparezca dentro de la expresión, que en esa posición debe aparecer dicho carácter para que la expresión sea válida.
cde	Siendo *c*, *d*, y *e* caracteres, indica que esos caracteres deben aparecer de esa manera en la expresión.
(*x*)	Permite indicar un subpatrón dentro del paréntesis. Ayuda a formar expresiones regulares complejas.
.	Cualquier carácter. El punto indica que en esa posición puede ir cualquier carácter
^*x*	Comenzar por. Indica el String debe empezar por la expresión *x*.
x$	Finalizar por. Indica que el String debe terminar con la expresión x.
x+	La expresión a la izquierda de este símbolo se puede repetir una o más veces
*x**	la expresión a la izquierda de este símbolo se puede repetir cero o más veces
x?	El carácter a la izquierda de este símbolo se puede repetir cero o una veces
x{*n*}	Significa que la expresión *x* aparecerá *n* veces, siendo *n* un número entero positivo.
x{*n*,}	Significa que la expresión *x* aparecerá *n* o más veces
x{*m*,*n*}	Significa que la expresión *x* aparecerá de *m* a *n* veces.
x|*y*	La barra indica que las expresiones *x* e *y* son opcionales, se puede cumplir una u otra.
c-*d*	Cumplen esa expresión los caracteres que, en orden ASCII, vayan del carácter *c* al carácter *d*. Por ejemplo *a*-*z* repsenta todas las letras minúsculas del alfabeto inglés.
[*cde*]	Opción, son válidos uno de estos caracteres: *c*, *d* ó *e*
[^*x*]	No es válido ninguno de los caracteres que cumplan la expresión *x*. Por ejemplo *[^dft]* indica que no son válidos las caracteres *d*, *f* ó *t*.
\d	Dígito, vale cualquier dígito numérico
\D	Todo menos dígito
\s	Espacio en blanco

<table>
<tr><th>símbolo</th><th>significado</th></tr>
<tr><td>\S</td><td>Cualquier carácter salvo el espacio en blanco</td></tr>
<tr><td>\w</td><td>Word, carácter válido dentro de los que PHP considera para identificar variables. Es decir, letras, números o el guion bajo.</td></tr>
<tr><td>\W</td><td>Todo lo que no sea un carácter de tipo Word.</td></tr>
<tr><td>\n</td><td>Nueva línea</td></tr>
<tr><td>\t</td><td>Tabulador</td></tr>
<tr><td>\c</td><td>Permite representar el carácter c cuando este sea un carácter que de otra manera no sea representable (como [,], /, \,…). Por ejemplo \\ es la forma de representar la propia barra invertida.</td></tr>
<tr><td>\xff</td><td>Permite indicar un carácter Unicode mediante su código hexadecimal.</td></tr>
<tr><td>\p{xx}</td><td>Indica un carácter que cumpla la propiedad Unicode indicada con los símbolos xx que pueden ser:
<table>
<tr><th>Código</th><th>Significado</th></tr>
<tr><td>Cc</td><td>Control</td></tr>
<tr><td>Cf</td><td>Formato</td></tr>
<tr><td>Cn</td><td>Sin asignar</td></tr>
<tr><td>Co</td><td>Uso privado</td></tr>
<tr><td>Cs</td><td>Sustituto</td></tr>
<tr><td>L</td><td>Letra</td></tr>
<tr><td>Ll</td><td>Letra minúscula</td></tr>
<tr><td>Lm</td><td>Letra modificadora</td></tr>
<tr><td>Lo</td><td>Otra letra</td></tr>
<tr><td>Lt</td><td>Letra de título</td></tr>
<tr><td>Lu</td><td>Letra mayúscula</td></tr>
<tr><td>M</td><td>Marca</td></tr>
<tr><td>Mc</td><td>Marca de espacio</td></tr>
<tr><td>Me</td><td>Marca de cierre</td></tr>
<tr><td>Mn</td><td>Marca de no-espacio</td></tr>
<tr><td>N</td><td>Número</td></tr>
<tr><td>Nd</td><td>Número decimal</td></tr>
<tr><td>Nl</td><td>Número letra</td></tr>
<tr><td>No</td><td>Otro número</td></tr>
<tr><td>P</td><td>Puntuación</td></tr>
<tr><td>Pc</td><td>Puntuación de conexión</td></tr>
<tr><td>Pd</td><td>Puntuación guión</td></tr>
<tr><td>Pe</td><td>Puntuación de cierre</td></tr>
<tr><td>Pf</td><td>Puntuación final</td></tr>
<tr><td>Pi</td><td>Puntuación inicial</td></tr>
<tr><td>Po</td><td>Otra puntuación</td></tr>
<tr><td>Ps</td><td>Puntuación de apertura</td></tr>
<tr><td>S</td><td>Símbolo</td></tr>
<tr><td>Sc</td><td>Símbolo de moneda</td></tr>
<tr><td>Sk</td><td>Símbolo modificador</td></tr>
</table></td></tr>
</table>

símbolo	significado
	<table><tr><th>Código</th><th>Significado</th></tr><tr><td>Sm</td><td>Símbolo matemático</td></tr><tr><td>So</td><td>Otro símbolo</td></tr><tr><td>Z</td><td>Separador</td></tr><tr><td>Zl</td><td>Separador de línea</td></tr><tr><td>Zp</td><td>Separador de párrafo</td></tr><tr><td>Zs</td><td>Separador de espacio</td></tr></table>
i	La letra i al final de los delimitadores permite ignorar mayúsculas y minúsculas.

funciones de expresiones regulares

función	significado
preg_match(patrón, string [, arrayResult] [, flag] [,despl])	Indica una expresión regular (apartado ***patrón***) y la comprueba en el String que se pasa a la función. Devuelve verdadero en el caso de que el texto cumpla la expresión regular. **arrayResult** es un array que se puede indicar para almacenar en él el texto dentro del String que cumple el patrón. De modo que el elemento cero del array será el texto completo que cumpla todo el patrón, el elemento 1 será el texto que cumpla el primer subpatrón que se haya definido (los subpatrones se indican entre paréntesis), etc. El parámetro **flags** permite utilizar la constante PREG_OFFSET_CAPTURE para que el array anterior almacene además de cada texto que cumpla la condición, la posición en la que se estaba ese texto dentro del array original (lo coloca en el array detrás de cada texto que cumpla la condición) **despl** permite indicar el índice dentro del String por el que se empezará a buscar el patrón (si no se indica este parámetro comenzamos a buscar por el principio.
preg_match_all(patrón, string [, arrayResult] [, flag] [,despl])	Igual que la anterior, sólo que cuando encuentra el patrón, sigue buscando en el resto del texto para buscar la siguiente aparición. >Esto la hace más útil para usar arrays de resultados.

<table>
<tr><th>función</th><th>significado</th></tr>
<tr><td>preg_replace(
patrón,
reemplazo,
string
[, límite]
[, cuenta]
)</td><td>Busca el patrón en el string indicado y cuando le encuentra, devuelve un nuevo string resultado de sustituir el patrón encontrado por el texto de reemplazo indicado. Ejemplo:

$s="Este es un documento Java, y no de la isla de Java";
echo preg_replace("/java/i", "PHP",$s);

Escribe Este es un documento PHP, y no de la isla de PHP

El patrón puede incluso ser un array con varios patrones: si es así, se sustituye cada patrón del array en el string por el texto de reemplazo indicado.

Es posible que incluso el reemplazo sea un array de textos de reemplazo. Si es así: cada elemento del array de patrones se reemplaza por el elemento del array de reemplazo correspondiente.

El parámetro opcional límite pone un tope al número de reemplazos efectuados. Por ejemplo si vale 1, sólo se reemplaza la primera aparición del patrón. Por defecto vale -1 (es decir, no hay límite).

La variable cuenta, si se utiliza, sirve para almacenar el número de reemplazos efectuados.</td></tr>
<tr><td>preg_split(
patrón,
string
[, límite
[, flags]]
)</td><td>Divide el string indicado según el patrón de expresión regular indicado y devuelve un array que contiene cada trozo obtenido del string. Ejemplo:

$s="texto a dividir, ole y ole";
print_r (preg_split("/[\s,]+/", $s));

Sale:
Array
(
[0] => texto
[1] => a
[2] => dividir
[3] => ole
[4] => y
[5] => ole
)

El parámetro opcional límite indica el máximo de trozos a obtener.</td></tr>
</table>

(3.2.9) funciones estándar de uso con strings

funciones básicas

función	significado
strlen(string)	Devuelve el tamaño del String

mayúsculas y minúsculas

función	significado
strtolower(texto)	Convierte el texto a minúsculas
strtoupper(texto)	Convierte el texto a mayúsculas
lcfirst(texto)	Retorna un string resultado de poner en minúsculas el primer carácter del texto (suponiendo que sea una letra).
ucfirst(texto)	Retorna un string resultado de poner en mayúsculas el primer carácter del texto (suponiendo que sea una letra).

funciones de comparación

función	significado
strcmp(texto1, texto2)	Compara los dos textos (strings) y devuelve cero si son iguales, uno si el primero es mayor en orden alfabético (usando el código ASCII) y -1 si es mayor el segundo string.
strccasemp(texto1, texto2)	Igual que la anterior pero no tiene en cuenta las mayúsculas (sólo en inglés)
strnatcmp(texto1,texto2)	Igual que las anteriores, pero la comparación que hace tiene en cuenta la forma natural humana de ordenar. Así por ejemplo el texto *imagen2* sería considerado menor que *imagen11* (en orden alfabético es mayor).
strcasenatcmp(texto1,texto2)	Igual que la anterior pero sin considerar mayúsculas ni minúsculas.
levenshtein(texto1, texto2)	Devuelve un número entero conocido como distancia Levenshtein que simboliza el número de modificaciones al primer texto que nos permitirían conseguir el segundo texto. De modo que si esa distancia es pequeña, los dos textos son parecidos.
metaphone(texto [,fonemas])	Devuelve un código que indica la forma de pronunciar una palabra (usando el inglés), de modo que dos palabras con pronunciación similar tendrían el mismo código. El parámetro *fonemas* indica el número máximo de fonemas a tener en cuenta (menos, más palabras parecidas habrá)
similar_text(	Devuelve el número de caracteres parecidos entre el *texto1*

función	significado
texto1, texto2 [,porcentaje])	y el ***texto2***. En realidad lo interesante es usar el tercer parámetro ***porcentaje***. Este parámetro se pasa por referencia por lo que tiene que ser una variable. Dicha variable recibe un porcentaje de similitud entre ambas cadenas de texto.

funciones de búsqueda y reemplazo

función	significado

strpos(texto, textoBusq)	Devuelve la posición del segundo texto dentro del primero (empieza a contar la posición desde el número cero). Si el texto buscado no se encuentra, devuelve ***false***. Ejemplo:
	
	`$a="Esta es la comunidad de Castilla y León";`
	`echo strpos($a," Castilla");`
	
	Escribiría ***24***, ya que ***Castilla*** empieza a aparecer en la posición 24 del String ***$a***.

stripos(texto, textoBusq)	Igual que la anterior pero no tiene en cuenta mayúsculas ni minúsculas. Sólo es válida con textos que no usen caracteres fuera del alfabeto inglés.

strpbrk(texto, listaCars)	***listaCars*** es un string que contiene todos los caracteres que deseamos buscar en el texto de modo que busca cualquiera de esos caracteres dentro del texto.
	
	La función si encuentra cualquiera de esos caracteres, devuelve el resto de caracteres desde la posición del primer carácter que encuentre en el texto. Ejemplo:
	
	`echo strpbrk("este año no voy a la montaña","ñyn");`
	
	Escribe (puesto que encuentra primero a la eñe):
	
	`ño no voy a la montaña`

str_replace(textoBuscado, textoReemplazo, texto [,veces])	Localiza todas las apariciones del ***textoBuscado*** en el texto que se pasa como tercer parámetro y las cambia por el texto indicado en ***textoReemplazo***. El cuarto parámetro (opcional), ***veces***, se envía por referencia y almacenará el número de reemplazos realizados
str_ireplace(textoBuscado, textoReemplazo, texto [,veces])	Idéntica a la anterior, pero no distingue entre mayúsculas y minúsculas.

función	significado
substr_replace(texto, textoReemplazo, posInicial [,posFinal])	Coloca en el texto original, el texto indicado como *textoReemplazo*, de modo que sustituya a todos los caracteres desde la posición indicada como *posInicial*, hasta el final o hasta el número indicado con el parámetro *posFinal*.

strtr(texto,arrayTraducción)	Hace reemplazos múltiples de caracteres en el texto y devuelve el texto resultante de realizar esos reemplazos.
	
	El array que se usa como segundo parámetro, contiene datos de forma que cada índice se buscará en el texto y se reemplazará por su valor. Ejemplo:
	
	`$array=array("a"=>"á","e"=>"é","i"=>"í","o"=>"ó");`
	`echo strtr("este texto tiene", $array)`
	
	Escribe:
	
	ésté téxtó tíéné

stripslash(texto)	Elimina en el texto todos los caracteres *backslash* (\). Es muy útil para texto procedente de lenguajes de programación.
strip_tags(texto [,noRetirables])	Retira del texto todas las etiquetas de tipo HTML o PHP que haya contenidas, excepto las que se indiquen en el parámetro *noRetirables*, que es un string que contendrá las etiquetas que no queremos retirar seguidas. Ejemplo: `strip_tags(texto,"<p><strong")` Elimina del texto todas las etiquetas que haya excepto *p* y *strong*.

funciones de extracción de texto

función	significado

strstr(texto, textoBusq)	Busca un string dentro de otro y devuelve los caracteres desde su primera aparición hasta el final. En la respuesta incluye el texto buscado. Ejemplo:
	
	`$a="Esta es la comunidad de Castilla y León";`
	`echo strstr($a," Castilla");`
	
	Escribiría *Castilla y León*

stristr(texto, textoBusq)	Igual que la anterior pero no tiene en cuenta mayúsculas ni minúsculas. Sólo es válida con textos que no usen caracteres fuera del alfabeto inglés.

función	significado

strtok(string [,token])	Permite extraer un texto en base a una serie de caracteres de modo que primer coge el primer texto dentro del string original (primer token) y en las siguientes llamadas irá devolviendo el resto de tokens (en las siguientes llamadas no se usa el primer parámetro) hasta que tras devolver el último retorna el valor falso.
	
	Ejemplo:
	
	`$s="este es el texto, le vamos a partir";`
	`$tok=strtok($s," ,");`
	`while($tok!=false){`
	`  echo $tok."\n";`
	`  $tok=strtok(" ,");`
	`}`
	
	Escribe:
	
	este
	es
	el
	texto
	le
	vamos
	a
	partir

explode(
delimitador,
texto
[,límite])
	Devuelve un array donde cada elemento del mismo es una subcadena que procede de separar el texto indicado en base a su delimitador.
	
	El parámetro límite indica el máximo número de cadenas a extraer.
	Ejemplo:
	
	`$s="soy una frase, con unas, cuantas, comas";`
	`$a=explode(",",$s);`
	`print_r($a);`
	
	Escribirá:
	
	`Array()`
	`  ([0]=>soy una frase`
	`  [1]=>con unas`
	`  [2]=>cuantas`
	`  [3]=>comas`
	`)`

función	significado
implode(array)	Une todos los elementos del array para formar un texto que contiene el valor de cada elemento.
str_repeat(texto, veces)	Devuelve un string que contiene el texto indicado repetido las veces que se indiquen en el segundo parámetro.
str_shuffle(texto)	Devuelve una copia del texto donde cada carácter se ha movido aleatoriamente. Es decir, forma un anagrama.
strrev(texto)	Devuelve un string que contiene el texto original al revés.
str_split(texto [,tamaño])	Devuelve un array en el que en cada elemento hay un trozo del texto original. El texto se trocea por tamaño de caracteres; si no se indica *tamaño* se divide carácter a carácter; si, por ejemplo, tamaño vale 3, se divide de tres en tres caracteres.
str_word_count(texto [,formato [,listaCaracteres]])	Sin indicar segundo ni tercer parámetro, devuelve el número de palabras encontradas en el texto. El parámetro *formato* puede tomar los valores: 1 La función devuelve el número de palabras encontradas 2 Devuelve un array escalar con todas las palabras encontradas 3 Devuelve un array asociativo donde cada valor es cada palabra encontrada y su índice la posición en el texto original. *listaCaracteres* es un string que contiene caracteres que en esta función se deben de considerar como parte de la palabra y no como separadores de caracteres. Es muy útil para texto escrito en cualquier lengua distinta del inglés y así considerar a la eñe como parte normal del texto.

funciones para extraer subcadenas de texto

función	significado
substr(texto, posInicial [,tamaño])	Extrae del string que se pasa como primer parámetro, el texto que va desde la posición indicada (empezando a contar por cero) hasta el final. O bien, extrae desde dicha posición el número de caracteres indicados por el parámetro *tamaño*.

funciones de limpieza de texto

función	significado
trim(texto, [caracteres])	Sin usar el segundo parámetro, elimina del texto los espacios en blanco del principio y el final. Elimina también saltos de línea, retornos de carro, tabuladores, nulos y caracteres de salto vertical. El segundo parámetro permite indicar una lista de caracteres entrecomillada que serán los que se eliminen si se encuentran al final y al principio del texto (en lugar de eliminar los espacios).
ltrim(texto, [caracteres])	Igual que el anterior, pero sólo elimina los caracteres al inicio del texto.
rtrim(texto, [caracteres])	Igual que el anterior, pero sólo elimina los caracteres al final del texto.

obtener códigos ASCII

función	significado
chr(códigoASCII)	Devuelve el carácter correspondiente al código ASCII indicado.
ord(carácter)	Inversa a la anterior. Devuelve el código ASCII del carácter indicado.

funciones de formato de datos

función	significado
number_format(número [,decimales [,caracterDecimal [,caracterDeMiles]]])	Devuelve un string que contiene al número que se indica formateado de modo que aparece el separador de miles. El segundo parámetro (opcional) permite indicar cuántos decimales vamos a utilizar para mostrar el número. Los otros dos parámetros se utilizan para indicar cuál es el carácter de separador de decimales y el de miles. Ejemplo: `echo number_format(1234567.892,2,",",".");` Muestra: 1.234.567,89

función	significado
money_format(formato, número)	Devuelve un string resultante de aplicar al número que se recibe como segundo parámetro el formato de moneda indicado mediante el primer parámetro. Sólo funciona en entornos compatibles con la función strfmon de lenguaje C (Windows no lo es). El parámetro formato es, a su vez, un string que contiene símbolos especiales para dar formato. Esa expresión está precedida por el símbolo % al que le pueden seguir uno o más de estos caracteres **i** Formatea el número usando los símbolos y separadores pertinentes según lo especificado por la función setlocale basándose en el sistema monetario internacional. Por ejemplo si antes hemos usado *setlocale("LC_ALL","es-ES")*, el formato del número será el correspondiente al formato monetario español. **n** Idéntica al anterior pero usa el sistema nacional de moneda, según la configuración regional especificada por setlocale. **=*f*** Mediante el símbolo =, se especifica un carácter de relleno para el número (por defecto se usa el espacio en blanco). **^** Deshabilita las opciones de agrupamiento (el separador de miles) **+** Indica signo para el número **(** Muestra los números negativos entre paréntesis **!** Elimina el símbolo de moneda **#*n*** Caracteres de anchura que se usarán para la parte entera del número. **.*p*** Número de decimales que se utilizarán **-** Alineación izquierda del número (normalmente la alineación es derecha)

cifrado

función	significado
md5(texto [,raw_input])	Devuelve hash correspondiente al texto indicado usando el algoritmo MD5. Si el parámetro *raw_input* se indica con valor true (por defecto vale false), codifica en binario crudo con tamaño 16.
sha1(texto [,raw_input])	Igual que el anterior pero utilizando el algoritmo SHA1

función	significado
hash(algoritmo, texto [,raw_input])	Genera el hash correspondiente a aplicar sobre el texto el algoritmo que se indique en el parámetro ***algoritmo*** (puede ser "sha256" , "md5". La lista completa de posibles algoritmos se puede consultar con la función ***hash_algos*** que devuelve un array con todos los nombres posibles a utilizar.
crc32(texto)	Genera el polinomio CRC32 de comprobación sobre el string indicado. Mediante ese polinomio podemos comprobar la validez de los datos.
crypt(texto, salt)	Función completa para cifrar el texto indicado. Su funcionamiento es complejo.
str_rot13(texto)	Codifica el texto utilizando la rotación **ROT13** de cifrado.

(3.3) funciones de fecha

Como en todos los lenguajes el manejo de fechas y horas en complejo. Pero PHP proporciona numerosas funcionas que facilitan su manejo.

PHP utiliza el formato de fecha y hora de Unix, por lo que muchas funciones requieren pasar las fechas en este formato. Por ello hay otras muchas funciones que nos ayudan a crear fechas en ese formato a partir de un texto que represente a una fecha.

función	significado
time()	Devuelve la fecha y hora actual en el formato nativo (Unix) de PHP.
strftime(formato[,fecha])	Devuelve un texto que representa la fecha actual (o la que se indique como segundo parámetro) en el formato regional establecido con **setlocale**. El formato puede incluir estos símbolos: %a Día de la semana (tres letras) %A Día de la semana (completo) %d El día del mes con dos dígitos (del 01 al 31) %e El día del mes (de 1 a 31) %j Día del año, 3 dígitos (del 001 al 366) %u Día de la semana (del 1 al 7) %w Día de la semana (del 0 al 6) %W Número de semana del año, comenzando con el primer Domingo como la primera semana

función	significado
	%b Nombre del mes con tres letras %B Nombre del mes completo %m Número de4l mes en dos cifras (del 01 al 12) %C Número de siglo (por ejemplo 21) %y Año en dos cifras %Y Año en cuatro cifras %H Hora en formato 24 horas (del 00 al 23) %k Hora en formato 24 horas (del 0 al 23) %I Hora en formato de 12 horas (01 al 12) %l La hora en formato de 12 horas (del 1 al 12) %M Minutos en dos cifras (del 01 al 59) %p 'AM' o 'PM' en MAYÚSCULAS basados en la hora dada %P 'am' o 'pm' en minúsculas basados en la hora dada %S Segundos en dos dígitos (del 00 al 59) %X Representación preferida de la hora basada en la configuración regional, sin la fecha
date(formato [,fecha])	Da formato a la fecha. Si no se indica segundo parámetro, se toma la fecha actual. El formato puede llevar estos símbolos (se indican los más importantes): d Día del mes (del 01 al 31) D Día de la semana con trEs letras (en inglés) j Día del mes (del 1 al 31) l Día de la semana (en inglés) w Día de la semana (del 1 al 7) z Día del año (del 1 al 365 ó 366) F Nombre del mes(en inglés) m Mes (del 01 al 12) M Mes con tres letras (en inglés) n Mes (del 1 al 12) t Número de días del mes dado (28,29,30 ó 31)

función	significado
	L Verdadero si el año es bisiesto y Año con dos cifras Y Año con cuatro cifras a Símbolo AM en minúsculas A AM en mayúsculas h Hora en formato 12 horas (del 01 al 12) H Hora en formato 24 horas (del 00 al 23) g Hora en formato 12 horas (del 1 al 12) G Hora en formato 24 horas (del 0 al 23) i Minutos (del 00 al 59) s Segundos (del 00 al 59) u Microsegundos
mktime([hora [,minuto [,segundo [,mes [,día [,año [época]]]]]]] **)**	Crea una fecha y hora a partir de los parámetros indicados. Los que no se indiquen se toman de la fecha y hora actuales. El último parámetro indica con un 1 que estamos en horario de verano y con un -1 en el de invierno.
strptime(fecha, formato**)**	Devuelve un array asociativo que contiene como valores los datos de la fecha indicada. El formato cumple los posibles valores del parámetro ***formato*** de la función ***strftime*** explicada anteriormente.
local_time([fecha [,asociativo]] **)**	Devuelve la fecha actual (o la que se indique como primer parámetro) en forma de array en el que cada elemento contiene cada parte de la fecha y hora. Se puede indicar una fecha concreta y un valor verdadero como segundo parámetro para indicar que deseamos un

función	significado
	array asociativo en lugar de escalar.

strtotime(texto)	Analiza el texto y devuelve la fecha correspondiente. Ejemplos de uso[2]:
	`<?php`
	`echo strtotime("now"), "\n";`
	`echo strtotime("10 September 2000"), "\n";`
	`echo strtotime("+1 day"), "\n";`
	`echo strtotime("+1 week"), "\n";`
	`echo strtotime("+1 week 2 days 4 hours 2 seconds"), "\n";`
	`echo strtotime("next Thursday"), "\n";`
	`echo strtotime("last Monday"), "\n";`
	`?>`

checkdate(mes, dia, año)	Devuelve verdadero si la fecha indicada de esta forma es válida.
date_default_timezone_get()	Devuelve la zona horaria en uso (por ejemplo *Europe/Berlin*)
date_default_timezone_set(zona)	Establece la nueva zona horaria. EL texto debe ser estándar, por ejemplo (*Europe/Berlin*).

[2] Ejemplo #1 de la documentación de php.net