Tema 6
Arrays

Supongamos que queremos construir un algoritmo que almacene información sobre la cantidad de agua de lluvia recogida en nuestra localidad durante el último año, para posteriormente realizar una serie de cálculos estadísticos. Como vemos, nuestra intención es la de almacenar y procesar una colección de valores relacionados entre sí y que además serán del mismo tipo. ¿Cómo nos enfrentaremos a este problema? Como cada dato es diferente al anterior, con lo aprendido hasta ahora y utilizando tipos de datos simples, no tendríamos más remedio que definir diferentes variables que almacenarán los valores correspondientes a cada mes. De esta forma, el algoritmo resultante puede ser excesivamente complejo y tedioso. Para resolver las complicaciones derivadas de este tipo de situaciones, la mayoría de los lenguajes de programación cuentan con diferentes construcciones de datos que podremos utilizar para obtener soluciones elegantes y eficientes.

Un **array** es un tipo de datos estructurado constituido por una cantidad fija de valores de un mismo tipo. Básicamente, podemos definir un **array** como una colección de datos que, visualmente puede verse como un conjunto de celdas contiguas y ordenadas, que contienen valores, accesibles directamente y de forma individual.

20	3	-8	4	...	50

Un array no sólo queda definido por el **tipo base** de los elementos que puede contener, sino también por un nombre o **identificador** que hará referencia a la colección completa.

Tipos e inicialización de arrays

Aunque PHP no distingue entre ambos, podemos clasificar los arrays como **escalares** y **asociativos**, permitiéndonos utilizar unos u otros indistintamente.

Arrays escalares

Utilizados tradicionalmente en la mayoría de los lenguajes, un **array escalar** utiliza un índice numérico para referirse a cada uno de los elementos de la colección. Pero, ¿cómo definimos un *array*? Existen diferentes alternativas.

```php
$a = array(20, 3, -8, 4, 50) ;
```

En el ejemplo anterior hemos utilizado el constructor **array()** del lenguaje para crear un array escalar llamado **$a** y que contiene los valores 20, 3, -8, 4 y 50. A partir de ahora podremos acceder a los valores del array a través del nombre de la variable y del índice que indica la posición que ocupa cada elemento. Recordemos que el primer elemento ocupará la posición **0** del *array*. De esta manera, si queremos mostrar el tercer valor tendremos que hacerlo como sigue:

```php
echo $a[2] ;
```

A partir de la versión 5.4 de PHP podemos también definir un *array* como sigue:

```php
$a = [20, 3, -8, 4, 50] ;
```

Otra alternativa nos permitirá inicializar un array mediante asignación directa. Esto es, supongamos que queremos crear un *array* **$b** que contiene diferentes valores numéricos de tipo real:

```php
$b[0] = 2.25 ;
$b[1] = 9.5 ;
$b[2] = 7.75 ;
```

Aunque también podremos recurrir a la asignación automática.

```php
$b[] = 54 ;
```

De esta forma se asignan de forma automática los índices, asignando a cada elemento el valor numérico siguiente al último existente. Si tenemos en cuenta la definición que hicimos previamente de **$b** en la que habíamos asignado tres valores, el *array* final contará con los siguientes valores:

0	1	2	3
2.25	9.5	7.75	54

Aunque pueda resultarnos útil para añadir elementos al final del *array*, este método **no es aconsejable** y, preferiblemente se recomienda utilizar cualquiera de los que se han expuesto anteriormente. Pero, ¿cuál es realmente el motivo? Es debido a que basta que la clave máxima haya existido en algún momento en el array, para tomar ese valor como punto de partida para determinar el índice del siguiente elemento. Veámoslo con un ejemplo, para el cual será necesario introducir la función **unset()** de PHP, que utilizaremos para destruir una determinada variable. Retomemos el ejemplo anterior.

```
$a = [20, 3, -8, 4, 50] ;
```

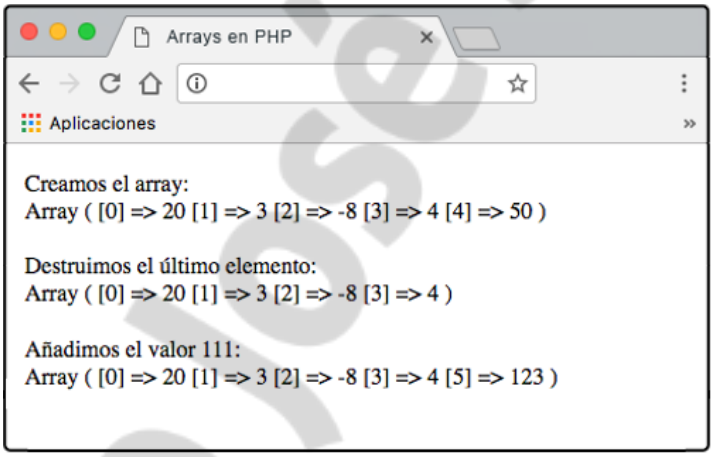
Como vemos, tenemos cinco elementos, siendo el **4** el valor del índice del último elemento. Supongamos que ahora eliminamos este elemento.

```
unset($a[4]) ;
```

Nuestro array pasará a tener ahora cuatro elementos. A continuación hacemos:

```
$a[] = 111 ;
```

Todo cabe suponer que el elemento se añadirá en la posición con índice 4 del array, pero nada más lejos de la realidad, ya que dicho elemento existió para PHP. De esta manera, el nuevo valor se añadirá al *array* bajo el índice 5, tal y como vemos:



Observamos que tenemos un resultado que, según qué situaciones, puede resultar poco apropiado o no. Téngase en cuenta que si hacemos **unset($a)** estaremos destruyendo el array completo. Recuerda que, si intentas acceder a una posición del *array* que no exista se obtendrá el error **Undefined offset**.

Arrays asociativos

Un **array asociativo** es aquel que tiene una cadena como valor del índice. De esta forma, podemos realizar asociaciones entre valores. Si has utilizado diccionarios en lenguajes como Python, seguramente estarás familiarizado con este concepto.

Utilizamos los métodos conocidos para definir este tipo de arrays, teniendo en cuenta que ahora debemos especificar como índice una cadena de caracteres. Utilizando el constructor **array()** del lenguaje utilizamos la siguiente sintaxis.

```
array(
```

```
  clave1 => valor1,
  clave2 => valor2,
  ...
) ;
```

Donde la clave será una cadena de caracteres y el valor puede ser de cualquier tipo. Igual que sucedía anteriormente, también podemos hacer uso de la sintaxis abreviada, sustituyendo por corchetes el constructor del lenguaje.

```
$array = [
  clave1 => valor1,
  clave2 => valor2,
  ...
];
```

Utilizamos el método tradicional para mostrar cualquier valor del *array*. Sin embargo, cuando tengamos que recorrer el *array* tendremos que hacerlo utilizando la construcción iterativa más apropiada.

Arrays con diferentes índices

La versatilidad de PHP nos permite combinar arrays escalares y asociativos, de manera que en un mismo array convivan índices de diferentes tipos. Como será necesario especificar cada par (clave, valor), utilizaremos cualquiera de los métodos expuestos en el apartado anterior.

```
$array = [
  1    => "a",
  "1"  => "b",
  1.25 => true,
  true => "d",
] ;
```

Es importante recordar que los únicos índices permitidos son valores de tipo entero y cadenas de caracteres. Entonces, ¿por qué en el ejemplo anterior hemos utilizado también valores de tipo **real** y **boolean**? PHP nos permite utilizar también valores de tipo **float**, **boolean** y **null** para definir el índice de cada elemento del array, aunque serán convertidos automáticamente según la siguiente tabla.

Tipo	Conversión
string	Una cadena que contenga un entero será almacenada como un índice de tipo escalar.

float	Entero.
bool	Entero
null	Cadena vacía.

De esta manera, cuando accedemos a los elementos del *array* debemos hacerlo de forma apropiada, según el índice que se haya definido para cada uno de ellos. No obstante, cuando queremos recorrerlo mediante un bucle, ¿qué construcción utilizamos? Si recordamos lo expuesto en la sección anterior, dado que tenemos índices de tipo asociativo, no podremos utilizar una sentencia iterativa tradicional. Supongamos el siguiente *array*:

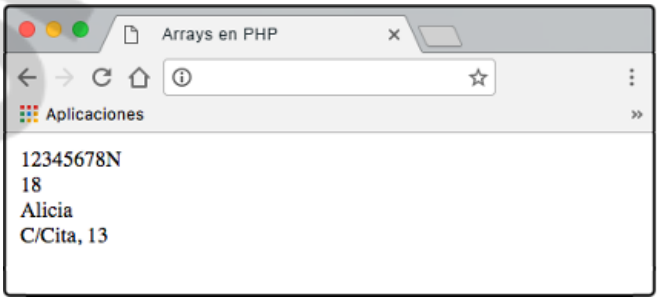
```
$array = [
  0        => "12345678A",
  "1"      => 18,
  "nombre" => "Alicia",
  3        => "C/Cita, 13",
] ;
```

Utilizamos un bucle **for** para recorrer el array completo y mostrar en pantalla su contenido:

```
for($i = 0; $i < 4; $i++)
echo "{$array[$i]}<br/>" ;
```

Si comprobamos ahora el fragmento de código anterior, ¿qué resultado obtendremos por pantalla? Básicamente tendremos un resultado a medias. Aparecerán los valores 12345678N, 18 y C/Cita,13. ¿Por qué? Precisamente porque son éstos los que se encuentran asociados a índices de tipo numérico. Sin embargo, obtendremos un error **Undefined offset 2** en el lugar donde debería haber aparecido el nombre Alicia. Sin embargo, si utilizamos la construcción **foreach** tendremos el resultado esperado

```
foreach($array as $valor) echo "$valor<br/>" ;
```



Finalizamos este apartado comentando algo que hemos omitido hasta ahora. Sea cual sea el tipo de array que estemos definiendo, incluso los escalares, no solo podremos especificar siempre un valor para la clave, sino que además podremos omitirlo.

```php
$array = [
       "12345678A",
       18,
  5 => "Alicia",
       "C/Cita, 13",
] ;
```

Si mostráramos el array en pantalla con la función **print_r** tendríamos el siguiente resultado.

```
Array( [0] => 12345678A [1] => 18 [5] => Alicia [6] => C/Cita, 13 )
```

Como vemos, el intérprete completa de forma apropiada los índices no especificados. Cuando definimos un *array* escalar obviamos siempre el valor de clave. No obstante, siempre podemos optar por especificarla, igual que hacemos cuando definimos un *array* asociativo o mixto. Ahora bien, también podemos optar por no especificar la clave en estos últimos, tal y como hacíamos con los escalares.

Arrays constantes

PHP7 incorporó como novedad la posibilidad de definir constantes de tipo array. De esta manera y utilizando la directiva **define** podemos hacer lo siguiente.

```php
define("ROBOTS", [ "C3PO", "R2D2", "BB8" ]) ;
```

Posteriormente utilizaremos el identificador **ROBOTS** como cualquier variable array para poder acceder a cada uno de sus elementos que, lógicamente no podrán ser modificados.

Operadores de arrays

A diferencia de lo que ocurre con otros lenguajes, PHP permite copiar un array en una variable utilizando directamente el **operador asignación** (=). Además de este, existen otros operadores que también podemos utilizar con arrays de forma análoga a como lo hacíamos con operadores escalares.

Operador	Se escribe...	Resultado
+	**$a + $b**	El **operador unión** se utiliza para concatenar dos *arrays*. No se añadirán aquellas claves de **$b** que también existan en **$a**.
==	**$a == $b**	Compara el contenido de ambos *arrays* y devuelve **true** si coinciden; **false** en otro caso..

===	$a === $b	Igual que el anterior. Compara ambos *arrays* y devuelve **true** si contienen los mismos elementos, del mismo tipo y en el mismo orden; **false** en otro caso.
!=	$a != $b	El operador distinto devuelve **true** si los *arrays* son diferentes; **false** en otro caso.

Arrays multidimensionales

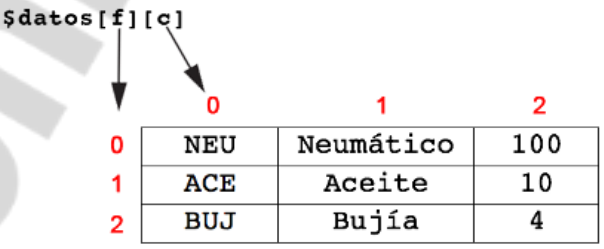
Hasta ahora hemos construido arrays que almacenan valores de tipo cadena, entero o real, pero asociado a una clave también podemos encontrar otro array. De esta manera, creamos lo que se conoce como **array bidimensional** o **matriz**, constituida por filas y columnas.

Código	Descripción	Precio
NEU	Neumático	100
ACE	Aceite	10
BUJ	Bujía	4

En PHP definimos un *array* bidimensional para almacenar la información anterior.

```
$datos = array ( array("NEU", "Neumático", 100),
                 array("ACE", "Aceite",     10),
                 array("BUJ", "Bujía",       4)) ;
```

Observamos como nuestro *array* **$datos** contiene otros tres arrays en su interior. Gráficamente podemos representarlo como vemos en la siguiente imagen



Aunque podemos interpretar la forma de almacenamiento como deseemos, generalmente se toma como convención usar el primer para señalar la fila, y el segundo para la columna. De esta manera, si queremos mostrar el valor **Aceite** bastará con escribir:

```
echo $datos[1][1] ;
```

Obviamente, podemos crear un *array* asociativo para almacenar la misma información, lo

cual puede resultar mucho más conveniente para recordar a qué corresponde cada uno de los valores guardados en nuestro *array*.

```
$datos = [
    ["codigo" => "NEU", "descripcion" => "Neumático", "precio" => 100],
    ["codigo" => "ACE", "descripcion" => "Aceite",    "precio" => 10],
    ["codigo" => "BUJ", "descripcion" => "Bujía",     "precio" => 4],
];
```

¿Cómo mostraremos ahora el valor **Aceite** en pantalla? Tal y como se ha definido el *array*, tendríamos que indicar en primer lugar el número de la fila y seguidamente el índice asociado al elemento.

```
echo $datos[1]["descripcion"] ;
```

Ahora cabría hacerse una nueva pregunta: ¿podríamos crear un *array* bidimensional asociativo completo? Esto es, cada fila quedaría identificada por una cadena de caracteres y no por un valor numérico. La respuesta es simplemente si.

```
$datos = [ "f1" => ["codigo" => "NEU", "descripcion" => "Neumático" ,  ... ],
           "f2" => ["codigo" => "ACE", "descripcion" => "Aceite" ,     ... ],
           "f3" => ["codigo" => "BUJ", "descripcion" => "Bujía" ,      ... ],
  ] ;
```

De igual manera a como hemos hecho con las matrices, podríamos crear *arrays* multidimensionales escalares o asociativos de otras dimensiones. Generalmente, el uso de *arrays* de más de tres dimensiones no suele utilizarse con frecuencia debido a su falta de practicidad y a la cantidad de memoria que se requiere para almacenarlos.

Funciones para el manejo de arrays

PHP incorpora bastantes funciones para facilitarnos el trabajo a la hora de utilizar arrays. Aunque podemos obtener más información sobre todas ellas a través de la documentación oficial del lenguaje, resumimos de forma esquemática en la siguiente tabla alguna de estas funciones.

Función	Descripción
is_array($a)	Comprueba si la variable **$a** es un *array*.
count($a [, modo])	Cuenta el número de elementos incluídos en el *array* **$a**. Si establecemos el **modo** como **COUNT_RECURSIVE** nos será bastante útil para contar todos los elementos de un *array* multidimensional.

current($a)	Todo *array* cuenta con un puntero interno que señala la posición del elemento actual. La función devuelve el elemento apuntado. Si el *array* está vacío devuelve **false**.
end($a)	Hace que el puntero interno del *array* apunte al último elemento de la colección y devuelve su valor o **false** si el *array* está vacío.
next($a)	Desplaza la posición del puntero al siguiente elemento y devuelve el elemento apuntado.
prev($a)	Hace retroceder la posición del puntero a la posición anterior.
reset($a)	Restablece el puntero de **$a** al comienzo del *array* y devuelve su valor. Si el array está vacío devuelve **false**.
in_array($i, $a [, modo])	Busca el elemento **$i** en el *array* **$a** utilizando una comparación flexible. Si el **modo** se establece como **true** se comprobarán también los tipos.

Bibliografía

PHP.net
Guía oficial del lenguaje

PHP and MySQL Web Development
Luke Welling, Laura Thompson
Developer's Library

Learning PHP7
Antonio López
Packt Publishing