



PHP – Arrays



Ciclo: DAW
Módulo: DWES
Curso: 2020-2021
Autor: César Guijarro Rosaleny



Introducción.....	3
Declaración.....	4
Acceso.....	9
Recorrer un array.....	10
Operadores para array.....	12
Unión.....	12
Comparadores de array.....	14
Funciones de los arrays.....	16
count().....	16
array_push().....	16
array_values().....	17
array_keys().....	18
in_array().....	19
array_search().....	19
array_key_exists().....	20
Sort().....	20
array_merge().....	21
array_diff().....	22
Bibliografía.....	23

Introducción

Un array en PHP es en realidad un mapa ordenado. Un mapa es un tipo de datos que asocia *valores* con *claves*. Este mapa se puede emplear como array, lista, tabla asociativa, diccionario, colección, pila, cola....

Como los valores de un array pueden ser otro arrays, podemos crear árboles y arrays multidimensionales.

Los valores no tienen porqué ser del mismo tipo de datos. Podemos crear un array donde se mezclen valores enteros con cadenas de caracteres. Incluso las claves pueden ser de diferentes tipos, como veremos más adelante.

Existen multitud de funciones en PHP para trabajar con arrays. Más adelante veremos algunas de las más útiles.

Declaración

Para declarar una variable de tipo array utilizamos el constructor **array()**. A continuación podemos escribir directamente los valores separados por coma, o definir las claves y sus valores. Para definir pares de clave – valor, usamos la notación **clave => valor**.

```
<?php
$a = array(9,12,34,56,78);

$b = array(
    'a' => 9,
    'b' => 12,
    'c' => 34,
    'd' => 56,
    'e' => 78
);
```

Si utilizamos la primera forma (no definimos las claves), PHP asigna una clave de forma automática a cada elemento que será un entero dependiendo de la posición que ocupe el valor en el array, **empezando por cero**.

A partir de PHP 5.4 también se puede usar la sintaxis de array corta, la cual reemplazará **array()** por **[]**.

```
<?php
$c = [
    'a' => 9,
    'b' => 12,
    'c' => 34,
    'd' => 56,
    'e' => 78
];
```

Las claves pueden ser un *integer* o un *string*. El valor puede ser de cualquier tipo (incluso otro array).

Hay que tener en cuenta los siguientes puntos para las claves:

- Un *string* que contenga un *integer* válido será convertido a *integer*. Por ejemplo, la clave “8” se transformará en 8. Sin embargo, la clave “08” no se convertirá en 8, ya que no es un *integer* válido.
- Un *float* eliminará la parte fraccionaria y se almacenará como la parte entera del número. Por ejemplo, la clave 7,3 se almacenará como 7
- Un booleano también se almacenará como *integer*. La clave `true` = 1, y `false` = 0.
- Un clave *null* será almacenada como la cadena “”

Si varios elementos en la declaración del array usan la misma clave, sólo se utilizará la última, eliminando los demás elementos.

```
<?php
$d = [
    1 => 'Pepe',
    1.5 => 'María',
    true => 'Juan',
    '1' => 'Amparo'
];
```

El array anterior se convertirá en:

```
<?php
$d = [
    1 => 'Amparo'
];
```

Los array en PHP pueden contener claves *integer* y *string* al mismo tiempo, ya que PHP no distingue entre arrays indexados y asociativos (son todos asociativos).

```
<?php
$e = [
    100 => 'Pepe',
    101 => 'María',
    'foo' => 'Juan',
    'bar' => 'Amparo'
];
```

La clave es opcional. Si no se utiliza, PHP usará el incremento de la clave de tipo *integer* mayor utilizada anteriormente (si no hemos utilizado ninguna empezará por 0).

Por ejemplo, si definimos el siguiente array:

```
<?php
$f = array('Pepe', 'María', 'Juan', 'Amparo');
```

PHP lo convertirá en:

```
<?php
$f = [
    0 => 'Pepe',
    1 => 'María',
    2 => 'Juan',
    3 => 'Amparo'
];
```

Es posible especificar las claves para algunos elementos y excluir a los demás.

```
<?php
    $f = [
        'Pepe',
        'María',
        6 => 'Juan',
        'Amparo'
    ];
```

El anterior array será equivalente a:

```
<?php
    $f = [
        0 => 'Pepe',
        1 => 'María',
        6 => 'Juan',
        7 => 'Amparo'
    ];
```

NOTA: La última clave es 7 y no 3 porque, como hemos dicho antes, PHP utiliza el mayor entero utilizado anteriormente para las claves. Al ser 6 el mayor entero, PHP incrementa en 1 esa clave, dando como resultado 7.

Podemos crear arrays multidimensionales definiendo a su vez los valores como arrays.

```
<?php
$h = [
    [
        "nombre" => 'Pepe',
        "edad" => 25
    ],
    [
        "nombre" => 'María',
        "edad" => 36
    ]
];
```

Podríamos pensar en poner claves a los arrays del segundo nivel para reducir el tamaño de la siguiente forma:

```
<?php
$h = [
    "Pepe" => [
        "edad" => 25
    ],
    "María" => [
        "edad" => 36
    ]
];
```

Aunque ésto tiene un problema evidente. Ya que no se pueden repetir las claves de los arrays, si añadiésemos una nueva persona llamada “María”, machacaríamos los datos anteriores, perdiendo datos en el proceso.

Por regla general, lo mejor es crear los arrays multidimensionales de la primera forma, simulando tablas donde las claves serían enteros que nos indicaría el número de registro.

Acceso

Para acceder a los elementos de un array lo hacemos mediante `$array[clave]`.

Dependiendo de si hemos definido las claves o no, podremos utilizar números secuenciales (empezando por 0) o nuestras propias claves:

```
<?php
$miArray = [3, 5, 67, 9, 48];
echo $miArray[0]; # imprimirá 3
echo $miArray[0]; # imprimirá 67

$miArray = [
    "elemento1" => 2,
    "elemento2" => 5,
    "elemento3" => 67,
    "elemento4" => 9,
    "elemento5" => 48,
];
echo $miArray["elemento2"]; #imprimirá 5
echo $miArray["elemento5"]; #imprimirá 48
```

Para acceder a los elementos de los array multidimensionales tendremos que anidar tantas claves como niveles tenga el elemento al que queramos acceder.

```
<?php
$personas = [
[
    "nombre" => "Pepe",
    "edad" => 34
]
];
echo $personas[0]["nombre"]; #imprimirá Pepe
```

Recorrer un array

Aunque se puede recorrer un array con casi cualquier bucle, PHP nos ofrece un modo sencillo para iterar sobre ellos: **foreach**.

Existen dos sintaxis:

```
<?php
    foreach ($miArray as $value) {
        # code...
    }

    foreach ($miArray as $key => $value) {
        # code...
    }
```

La diferencia es que en la primera obtenemos sólo los valores, mientras que con la segunda forma podemos acceder también a las claves del array.

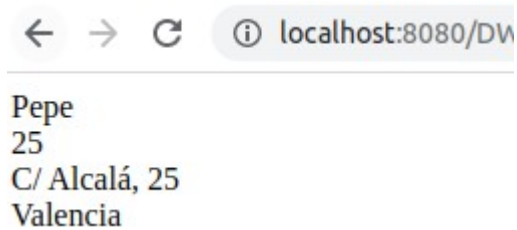
Por ejemplo, si tenemos el siguiente array:

```
<?php
    $contacto = [
        "nombre" => "Pepe",
        "edad" => 25,
        "direccion" => "C/ Alcalá, 25",
        "ciudad" => "Valencia"
    ];
```

Si recorremos el array con la primera forma:

```
<?php
foreach ($contacto as $value) {
    echo "$value<br>";
};
```

La salida será:



← → ↻ ⓘ localhost:8080/DV

Pepe
25
C/ Alcalá, 25
Valencia

Mientras que si lo recorremos de la segunda forma:

```
<?php
foreach ($contacto as $key => value) {
    echo "$value<br>";
};
```

Podríamos sacar también las claves del array por pantalla:



← → ↻ ⓘ localhost:8080/DV

nombre: Pepe
edad: 25
direccion: C/ Alcalá, 25
ciudad: Valencia

Operadores para array

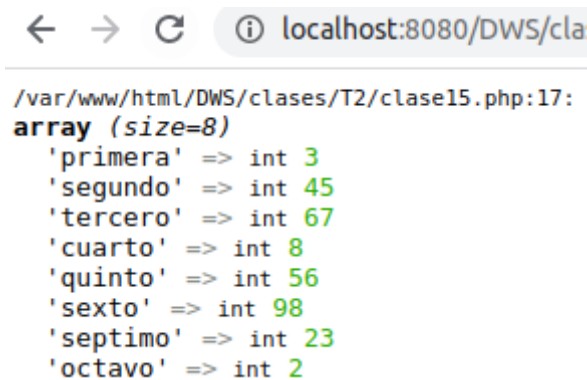
Unión

Para unir dos arrays utilizamos el operador `+`.

```
<?php
$arrayA = [
    "primero" => 3,
    "segundo" => 45,
    "tercero" => 67,
    "cuarto" => 8
];
$arrayB = [
    "quinto" => 56,
    "sexto" => 98,
    "septimo" => 23,
    "octavo" => 2
];

$resultado = $arrayA + $arrayB;
var_dump($resultado);
```

El array *\$resultado* sería:



```
← → ↻ ⓘ localhost:8080/DWS/cla
/var/www/html/DWS/clases/T2/clase15.php:17:
array (size=8)
  'primera' => int 3
  'segundo' => int 45
  'tercero' => int 67
  'cuarto' => int 8
  'quinto' => int 56
  'sexto' => int 98
  'septimo' => int 23
  'octavo' => int 2
```

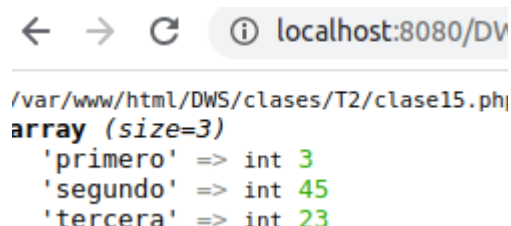
Hay que tener cuidado con el operador de unión para arrays. **Para las claves que existan en ambos arrays, serán utilizados los elementos del array de la izquierda y serán ignorados los elementos correspondientes del array de la derecha.**

Por ejemplo, si tenemos estos dos arrays:

```
<?php
$arrayA = [
    "primero" => 3,
    "segundo" => 45,
];
$arrayB = [
    "primero" => 56,
    "segundo" => 98,
    "tercer" => 23,
];

$resultado = $arrayA + $arrayB;
var_dump($resultado);
```

El array `$resultado` no contendrá 5 elementos, como podríamos pensar a priori, si no que tendría sólo 3 elementos:



```
← → ↻ ⓘ localhost:8080/DV
/var/www/html/DWS/clases/T2/clase15.php
array (size=3)
  'primero' => int 3
  'segundo' => int 45
  'tercera' => int 23
```

NOTA: Como hemos visto antes, si no utilizamos claves en nuestros arrays, PHP les asigna claves numéricas consecutivas empezando por 0, con lo que los primeros elementos de los dos arrays tendrán la misma clave y no obtendremos el resultado esperado.



Comparadores de array

Para comparar arrays podemos utilizar los mismos operadores de comparación que para el resto de tipo de datos con alguna particularidad.

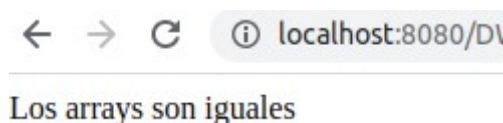
Si queremos saber si dos arrays son iguales, podemos usar `==` (igual que podemos utilizar `!=` para saber si son distintos), aunque puede haber alguna situación donde parezca que la comparación no funciona como debe.

Supongamos que tenemos el siguiente código:

```
<?php
$arrayA = [
    "primero" => 3,
    "segundo" => 45,
];
$arrayB = [
    "segundo" => 45,
    "primero" => 3,
];

if($arrayA == $arrayB) {
    echo "Los arrays son iguales";
} else {
    echo "Los arrays son diferentes";
}
```

Si vemos la salida que nos muestra el navegador, veremos que, al contrario de lo que podríamos pensar, los dos arrays son iguales:



← → ↻ ⓘ localhost:8080/D...
Los arrays son iguales

Esto ocurre porque el comparador `==` en arrays comprueba que tienen los mismos pares claves – valor, pero no el orden en el que están en cada array.

Si queremos comparar los arrays para saber si son exactamente idénticos, deberemos utilizar el operador `===`.

```
<?php
$arrayA = [
    "primero" => 3,
    "segundo" => 45,
];
$arrayB = [
    "segundo" => 45,
    "primero" => 3,
];

if($arrayA === $arrayB) {
    echo "Los arrays son iguales";
} else {
    echo "Los arrays son diferentes";
}
```

Cuya salida, ahora sí, es la esperada:



The screenshot shows a web browser window with the address bar displaying 'localhost:8080/DWS'. Below the address bar, the text 'Los arrays son diferentes' is displayed, which is the output of the PHP code provided in the previous block.

De todas formas, la mayoría de veces es más útil trabajar con funciones propias de los arrays, como veremos en el apartado siguiente.

Funciones de los arrays

Como hemos dicho, existen muchas funciones para trabajar con arrays. En este punto veremos algunas de ellas (podéis ver todas las que hay en el [manual de php](#)).

count()

Cuenta todos los elementos de un array. Podemos usar **sizeof()** como alias de **count()**.

```
<?php
$miArray = [34, 3, 65, 7, 87, 99, 3];
$longitud = count($miArray);
echo "El array tiene $longitud elementos";
```

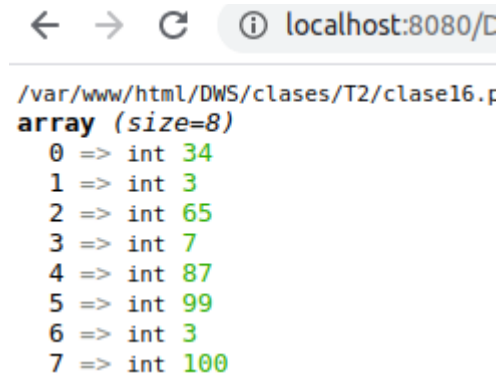


The screenshot shows a web browser address bar with "localhost:8080/D..." and a navigation bar with back, forward, and refresh buttons. Below the address bar, the text "El array tiene 7 elementos" is displayed, which is the output of the PHP code shown in the previous block.

array_push()

Añade un elemento al final de un array.

```
<?php
$miArray = [34, 3, 65, 7, 87, 99, 3];
array_push($miArray, 100);
var_dump($miArray);
```

```

<  >  ↻  ⓘ localhost:8080/
/var/www/html/DWS/clases/T2/clase16.p
array (size=8)
  0 => int 34
  1 => int 3
  2 => int 65
  3 => int 7
  4 => int 87
  5 => int 99
  6 => int 3
  7 => int 100

```

Podemos conseguir los mismo con **array[] = nuevo_elemento.**

```

<?php
    $miArray = [34, 3, 65, 7, 87, 99, 3];
    $miArray[] = 100;

```

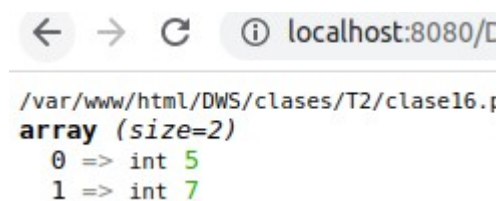
array_values()

Devuelve todos los valores del array y lo indexa numéricamente.

```

<?php
    $miArray = [
        "foo" => 5,
        "bar" => 7
    ];
    var_dump(array_values($miArray));

```



```


<  >  ↻  ⓘ localhost:8080/
/var/www/html/DWS/clases/T2/clase16.p
array (size=2)
  0 => int 5
  1 => int 7

```

array_keys()

Devuelve todas las claves de un array o un subconjunto de ellas.

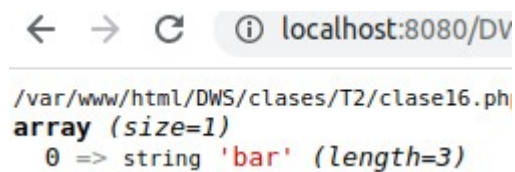
```
<?php
    $miArray = [
        "foo" => 5,
        "bar" => 7
    ];
    var_dump(array_keys($miArray));
```



```
localhost:8080/
/var/www/html/DWS/clases/T2/clase16
array (size=2)
  0 => string 'foo' (length=3)
  1 => string 'bar' (length=3)
```

Podemos sacar las claves de un valor si especificamos el segundo parámetro.

```
<?php
    $miArray = [
        "foo" => 5,
        "bar" => 7
    ];
    var_dump(array_keys($miArray, 7));
```



```
localhost:8080/DV
/var/www/html/DWS/clases/T2/clase16.php
array (size=1)
  0 => string 'bar' (length=3)
```

in_array()

Comprueba si un valor existe en un array.

```
<?php
$miArray = [34, 3, 65, 7, 87, 99, 3];
if (in_array(65, $miArray)) {
    echo "El valor existe en el array";
} else {
    echo "El valor no existe en el array";
}
```



A screenshot of a web browser window. The address bar shows 'localhost:8080'. The page content displays the text 'El valor existe en el array'.

array_search()

Busca el valor en un array y devuelve su clave.

```
<?php
$miArray = [
    "foo" => 5,
    "bar" => 7,
    "pre" => 18,
    "dig" => 56
];

$clave = array_search(7, $miArray);
echo "La clave del elemento 7 es $clave";
```



A screenshot of a web browser window. The address bar shows 'localhost:8080'. The page content displays the text 'La clave del elemento 7 es bar'.

array_key_exists()

Comprueba si la clave existe en el array. Podemos utilizar el alias **key_exists()**.

```
<?php
$miArray = [
    "foo" => 5,
    "bar" => 7,
    "pre" => 18,
    "dig" => 56
];

if (array_key_exists("dig", $miArray)) {
    echo "La clave dig existe en el array";
} else {
    echo "La clave dig no existe en el array";
}
```

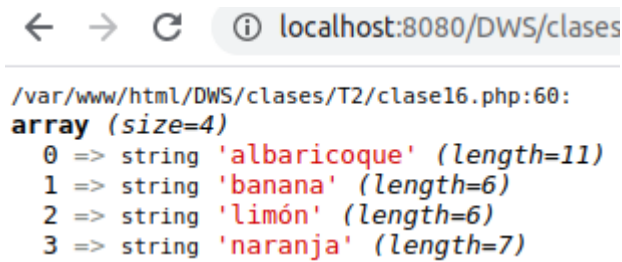
← → ↻ ⓘ localhost:8080/D

La clave dig existe en el array

Sort()

Ordena un array de menor a mayor.

```
<?php
$frutas = array("limón", "naranja", "banana", "albaricoque");
sort($frutas);
var_dump($frutas);
```



```

← → ↻ ⓘ localhost:8080/DWS/clases
/var/www/html/DWS/clases/T2/clase16.php:60:
array (size=4)
  0 => string 'albaricoque' (length=11)
  1 => string 'banana' (length=6)
  2 => string 'limón' (length=6)
  3 => string 'naranja' (length=7)

```

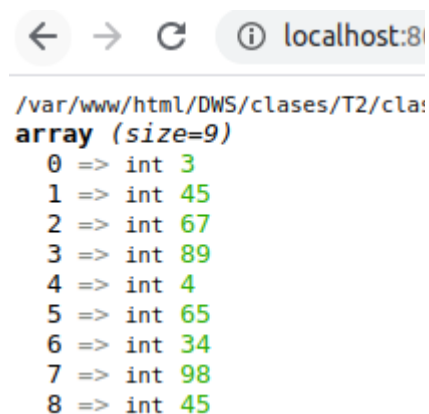
array_merge()

Combina los elementos de uno o más arrays juntándolos de modo que los valores de uno se anexan al final del anterior. Los valores del array de entrada con claves numéricas serán renumeradas con claves incrementales en el array resultante, comenzando desde cero.

```

<?php
$arrayA = [3, 45, 67, 89];
$arrayB = [4, 65, 34, 98, 45];
$resultado = array_merge($arrayA, $arrayB);
var_dump($resultado);

```



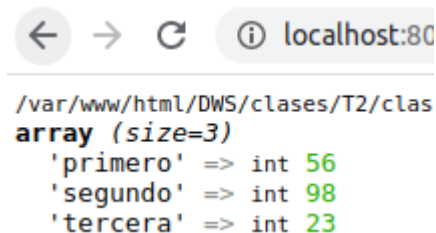
```

← → ↻ ⓘ localhost:8080
/var/www/html/DWS/clases/T2/clase16.php:60:
array (size=9)
  0 => int 3
  1 => int 45
  2 => int 67
  3 => int 89
  4 => int 4
  5 => int 65
  6 => int 34
  7 => int 98
  8 => int 45

```

Si los arrays de entrada tienen las mismas claves de tipo *string*, el último valor para esa clave sobrescribirá al anterior.

```
<?php
$arrayA = [
    "primero" => 3,
    "segundo" => 45,
];
$arrayB = [
    "primero" => 56,
    "segundo" => 98,
    "tercera" => 23,
];
$resultado = array_merge($arrayA, $arrayB);
var_dump($resultado);
```

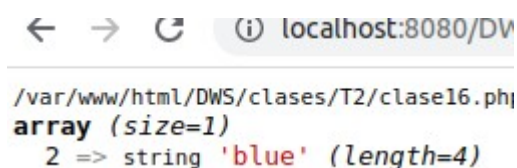


```
array (size=3)
  'primero' => int 56
  'segundo' => int 98
  'tercera' => int 23
```

array_diff()

Compara array1 con uno o más arrays y devuelve los valores de array1 que no estén presentes en ninguno de los otros arrays.

```
<?php
$arrayA = ["green", "red", "blue", "red"];
$arrayB = ["green", "yellow", "red"];
$resultado = array_diff($arrayA, $arrayB);
var_dump($resultado);
```



```
array (size=1)
  2 => string 'blue' (length=4)
```

Bibliografía

<http://php.net/manual/es/index.php>