

Array Functions

Disclaimer: вы смотрите просто запись лекции,
это НЕ специально подготовленный видеокурс!



Intro

In PHP we deal with arrays on regular basis. So, a lot of operations with arrays are already automated with a lot of inbuilt functions.

Full list: <https://www.php.net/manual/en/ref.array.php>

Warning! When dealing with multi-dimensional arrays, most of array functions work only with one level of an array, i.e. to process the whole array we have to implement a recursion.

Just a quick recap: important facts about arrays in PHP

Always dynamic.

There are no static arrays in PHP at all.

Dimension-dynamic.

Any dimension may be modified instantly.

Keys may be either **integers** or **strings**.

Detecting the size of an array

One of the most frequent operation with an array is the detection of its size.
The solution is that simple:

```
<?php

$someArray = array(array('A', 'B'), array('C', 'D'));
echo count($someArray) . "\n"; // 2
echo count($someArray, COUNT_RECURSIVE) . "\n"; // 6
```

Searching for an element in an array

The second most frequent operation with an array is the search for an element. It has some unusual side-effect due to types juggling:

```
<?php

// Search by key
$someArray = array('name' => 'Ivan', 'age' => 40);
if (isset($someArray['age'])) {
    echo $someArray['age'];
}

// Search by value (mind the 3rd parameter!!!)
$someArray = array('A', 99, true);
$x = in_array(99, $someArray); // true
$x = in_array(99999, $someArray); // true (!?!?!?!?)
$x = in_array(99999, $someArray, true); // false

// Search for a key by value (mind the 3rd parameter!!!)
$x = array_search(99, $someArray); // 1
$x = array_search(99999, $someArray, true); // false
```

Getting keys list and values list (“resetting” keys)

Sometimes we need to get the list of keys of an array as another array. Or we need to “reset” all keys (make them look like 0, 1, 2, 3, etc.)

```
<?php

$someArray = array('name' => 'Ivan', 'age' => 40, 'has_car' => true);
$keysArray = array_keys($someArray); // array(name, age, has_car)

$someArray = array(100 => 'A', 123 => 99, 555 => true);
$newArray = array_values($someArray); // array(0 => 'A', 1 => 99, 2 => true)
```

Sorting array data

There are a lot of inbuilt functions to sort array data almost any way one can imagine...

Ascending

sort

usort

rsort

asort

uasort

arsort

krsort

uksort

ksort

Descending

Do not preserve keys

Preserve keys

By keys values

Sorting array data

Now let's look at several usage samples. Pay attention: all these functions receive the analyzed array by reference.

```
<?php
```

```
$someArray = array('A' => 95, 'Z' => 17, 'N' => 35);  
sort($someArray); // array (0 => 17, 1 => 35, 2 => 95)  
rsort($someArray); // array (0 => 95, 1 => 35, 2 => 17)
```

```
$someArray = array('A' => 95, 'Z' => 17, 'N' => 35);  
asort($someArray); // array ('Z' => 17, 'N' => 35, 'A' => 95)  
arsort($someArray); // array ('A' => 95, 'N' => 35, 'Z' => 17)
```

```
$someArray = array('A' => 95, 'Z' => 17, 'N' => 35);  
ksort($someArray); // array ('A' => 95, 'N' => 35, 'Z' => 17)  
krsort($someArray); // array ('Z' => 17, 'N' => 35, 'A' => 95)
```


Sorting array data

But what if we need some custom approach? Imagine, we have to sort some orders based on the total price...

```
<?php
$orders = array
(
    array(10, 34, 34, 67),
    array(1, 2, 6),
    array(2000, 90),
    array(15, 67),
    array(34, 87, 34)
);
```

```
function ordersComparator($a, $b) : int
{
    $sum_a = 0;
    $sum_b = 0;

    foreach ($a as $v) {
        $sum_a += $v;
    }

    foreach ($b as $v) {
        $sum_b += $v;
    }

    return $sum_a <=> $sum_b;
}

print_r($orders);
usort($orders, 'ordersComparator');
print_r($orders);
```

```
Array
(
    [0] => Array
        (
            [0] => 1
            [1] => 2
            [2] => 6
        )
    [1] => Array
        (
            [0] => 15
            [1] => 67
        )
    [2] => Array
        (
            [0] => 10
            [1] => 34
            [2] => 34
            [3] => 67
        )
    [3] => Array
        (
            [0] => 34
            [1] => 87
            [2] => 34
        )
    [4] => Array
        (
            [0] => 2000
            [1] => 90
        )
)
```

Getting random array elements

Another frequent operation with an array is the extraction of several random elements.

```
<?php

$someArray = array(1, 5, 8, 15, 21, 34);
$randomElements = array_rand($someArray, 3);
print_r($randomElements);

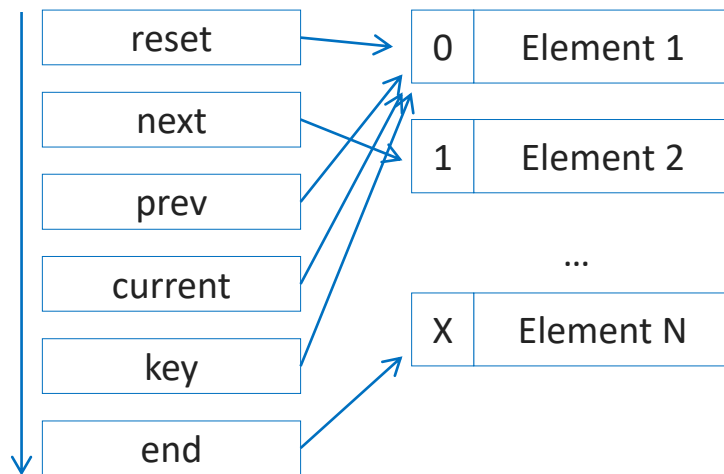
/*
Pay attention: here we have KEYS, not values!
Array
(
    [0] => 0
    [1] => 1
    [2] => 4
)
*/
```

Navigating an array step-by-step

Some algorithms require step-by-step array navigation. There are a lot of special functions for this case.

```
<?php
```

```
$someArray = [15, 19, 31, 85, 123];  
echo reset($someArray) . "\n";    // 15  
echo next($someArray) . "\n";     // 19  
echo prev($someArray) . "\n";     // 15  
echo current($someArray) . "\n";  // 15  
echo key($someArray) . "\n";      // 0  
echo end($someArray) . "\n";      // 123
```



... and so on!

Please visit this page to at least know the list of array functions, there are **80+** of them: <https://www.php.net/manual/en/ref.array.php>

The main idea here is not to reinvent the wheel, as almost everything you need with arrays is already implemented.

Superglobal arrays and variables

Several predefined variables in PHP are “superglobals”, which means they are available in all scopes throughout a script.

Details: <https://www.php.net/manual/en/language.variables.superglobals>

Superglobals: \$GLOBALS

This is an associative array containing references to all variables which are currently defined in the global scope of the script.

```
<?php
```

```
// Try running this script with web server  
// and from the command line!
```

```
$someVariable = 'Test';  
print_r($GLOBALS);
```

```
Array  
(  
    [_GET] => Array ()  
    [_POST] => Array ()  
    [_COOKIE] => Array ()  
    [_FILES] => Array ()  
    [_ENV] => Array ()  
    [_REQUEST] => Array ()  
    [_SERVER] => Array  
        (  
            [ALLUSERSPROFILE] => C:\ProgramData  
            ...  
        )  
    [someVariable] => Test  
)
```

```
Array  
(  
    [_GET] => Array ()  
    [_POST] => Array ()  
    [_COOKIE] => Array ()  
    [_FILES] => Array ()  
    [_ENV] => Array ()  
    [_REQUEST] => Array ()  
    [_SERVER] => Array  
        (  
            [HTTP_HOST] => 127.0.0.1  
            ...  
        )  
    [someVariable] => Test  
)
```

Superglobals: \$_SERVER

This is an associative array containing information such as headers, paths, and script locations.

```
<?php
```

```
// Try running this script with web server  
// and from the command line!
```

```
print_r($_SERVER);
```

```
Array  
(  
    [ALLUSERSPROFILE] => C:\ProgramData  
    [APPDATA] => C:\Users\VMUser\AppData\Roaming  
    [CommonProgramFiles] => C:\Program Files\Common Files  
    [CommonProgramFiles(x86)] => C:\Program Files (x86)\Common Files  
    [CommonProgramW6432] => C:\Program Files\Common Files  
    [COMPUTERNAME] => DESKTOP-CQBFSEE  
    [ComSpec] => C:\WINDOWS\system32\cmd.exe  
    ...  
    [argc] => 1  
)
```

```
Array  
(  
    [HTTP_HOST] => 127.0.0.1  
    [HTTP_USER_AGENT] => Mozilla/5.0 (Windows NT 10.0; Win64; x64;  
rv:96.0) Gecko/20100101 Firefox/96.0  
    [HTTP_ACCEPT] =>  
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w  
ebp,*/*;q=0.8  
    [HTTP_ACCEPT_LANGUAGE] => en-US,en;q=0.5  
    [HTTP_ACCEPT_ENCODING] => gzip, deflate  
    [HTTP_REFERER] => http://127.0.0.1/  
    ...  
)
```

Superglobals: \$_GET and \$_POST

These are associative arrays containing variables passed to the current script via the URL parameters (aka. query string) and via the HTTP POST method.

 <http://someurl.com/somepage.php?a=99&b=ABD&c=1>

```
<?php
```

```
echo $_GET['a'] . ' ' . $_GET['b'] . ' ' . $_GET['c'];  
echo $_POST['a'] . ' ' . $_POST['b'] . ' ' . $_POST['c'];
```

A: 99

B: ABD

C: 1

Go!

```
<form method="post" action="page.php">  
  A: <input type="text" name="a" size="5" /><br />  
  B: <input type="text" name="b" size="5" /><br />  
  C: <input type="text" name="c" size="5" /><br />  
    <input type="submit" value="Go!" />  
</form>
```


Superglobals: \$_GET and \$_POST

This is how you may explore PHP behavior related to \$_GET and \$_POST superglobal arrays:

```
<form method="post" action="09_GET_and_POST.php">
  A: <input type="text" name="a" /><br />
  B: <input type="text" name="b" /><br />
  C: <input type="text" name="c" /><br />
  <input type="submit" value="Go!" />
</form>

<?php

// Request this page like
// http://127.0.0.1/09_GET_and_POST.php?a=99&b=ABD&c=1
// or just submit the form data.

print_r($_GET);
print_r($_POST);
```

Superglobals: \$_FILES

This is an associative array of items **uploaded** to the current script via the HTTP POST method.

```
<form method="post" action="10_FILES.php" enctype="multipart/form-data">
  File 1: <input type="file" name="f1" /><br />
  File 2: <input type="file" name="f2" /><br />
  <input type="submit" value="Go!" />
</form>
```

```
<?php
```

```
// Just submit the form data.
```

```
print_r($_FILES);
```

File 1:	C:\1.txt	<input type="button" value="Browse..."/>
File 2:	C:\2.html	<input type="button" value="Browse..."/>
<input type="button" value="Go!"/>		

```
/*
Array
(
    [f1] => Array
        (
            [name] => 1.txt
            [type] => text/plain
            [tmp_name] => C:\Windows\temp\php132.tmp
            [error] => 0
            [size] => 258
        )
    ...
)
*/
```

Superglobals: \$_ENV (and getenv() function)

This is an associative array of variables passed to the current script via the environment method.

This array is empty until you set variables_order = "GPCSE" in php.ini.

```
<?php  
  
print_r($_ENV); // Usually just: Array()  
  
echo getenv('PATH');  
// C:\Program Files\Common Files\Oracle\Java\javapath;C:\WINDOWS\system32; ...
```

Superglobals: \$_REQUEST

This is an associative array of variables provided to the script via the GET, POST, and COOKIE input mechanisms and therefore could be modified by the remote user and cannot be trusted.

The presence and order of variables listed in this array is defined according to the PHP **request_order**, and **variables_order** configuration directives..

```
<?php  
  
print_r($_REQUEST);
```

It is advised to use \$_GET, \$_POST, \$_COOKIE arrays directly.

Superglobals: \$argc and \$argv

The \$argc superglobal variable contains the number of arguments passed to script, the \$argv superglobal variable contains an array of all the arguments passed to the script.

```
<?php
```

```
// Run this script from command line:
```

```
// php 13_argc_and_argv.php A B C D E
```

```
echo $argc . "\n"; // 6
```

```
print_r($argv);
```

```
/*
```

```
Array
```

```
(
```

```
    [0] => 13_argc_and_argv.php
```

```
    [1] => A
```

```
    [2] => B
```

```
    [3] => C
```

```
    [4] => D
```

```
    [5] => E
```

```
)
```

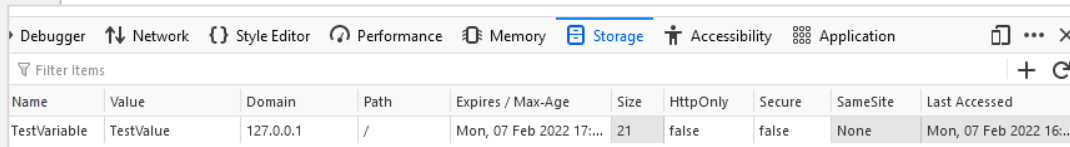
```
*/
```

Superglobals: \$_COOKIE

This is an associative array of variables passed to the current script via HTTP Cookies.

```
<?php  
  
setcookie("TestVariable", "TestValue", time()+3600);
```

```
<?php  
  
print_r($_COOKIE);  
  
/*  
Array  
(  
    [TestVariable] => TestValue  
)  
*/
```



The screenshot shows the 'Storage' tab in a web browser's developer tools. It displays a single cookie entry in a table. The table has columns for Name, Value, Domain, Path, Expires / Max-Age, Size, HttpOnly, Secure, SameSite, and Last Accessed. The data row shows a cookie named 'TestVariable' with the value 'TestValue', domain '127.0.0.1', and path '/'. The expires time is 'Mon, 07 Feb 2022 17:...' and the size is '21'. The 'HttpOnly' and 'Secure' flags are false, and 'SameSite' is 'None'. The last accessed time is 'Mon, 07 Feb 2022 16:...'.

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Last Accessed
TestVariable	TestValue	127.0.0.1	/	Mon, 07 Feb 2022 17:...	21	false	false	None	Mon, 07 Feb 2022 16:...

Superglobals: \$_COOKIE

An HTTP cookie (web cookie, browser cookie) is a small piece of data that a server sends to a user's web browser. The browser may store the cookie and send it back to the same server with later requests. Typically, an HTTP cookie is used to tell if two requests come from the same browser — keeping a user logged in, for example. It remembers stateful information for the stateless HTTP protocol.

Cookies are mainly used for three purposes:

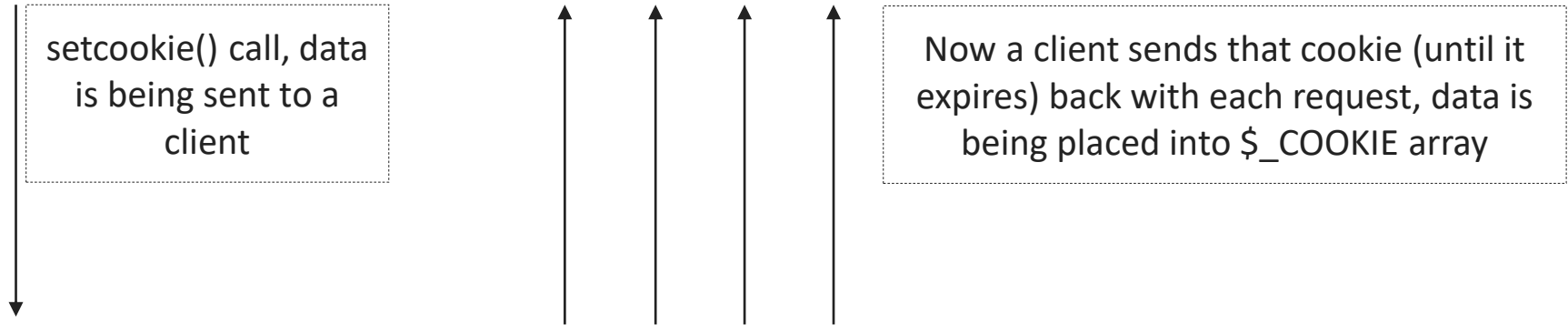
- Session management: logins, shopping carts, game scores, or anything else the server should remember.
- Personalization: user preferences, themes, and other settings.
- Tracking: recording and analyzing user behavior.

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies>

Superglobals: \$_COOKIE

This is critical to understand: any data appears in \$_COOKIE array only **after** being **received from a browser**, not immediately after setcookie() call.

SERVER [does not store any data]



CLIENT (browser) [stores 100% of all cookie data]

Superglobals: \$_SESSION

This is an associative array containing session variables available to the current script.

```
<?php

// Must be en each separately
// started script!
session_start();

$_SESSION['someVariable'] = 999;
```

```
<?php

// Must be en each separately
// started script!
session_start();

print_r($_SESSION);

/*
Array
(
    [someVariable] => 999
)
*/
```

Debugger										
Filter Items										
Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Last Accessed	
PHPSESSID	on82tabh9tribfktfg6hn4rh5	127.0.0.1	/	Session	35	false	false	None	Mon, 07 Feb 2022 1...	

Superglobals: \$_SESSION

HTTP sessions is an industry standard feature that allows Web servers to maintain user identity and to store user-specific data during multiple request/response interactions between a client application and a Web application. HTTP sessions preserves:

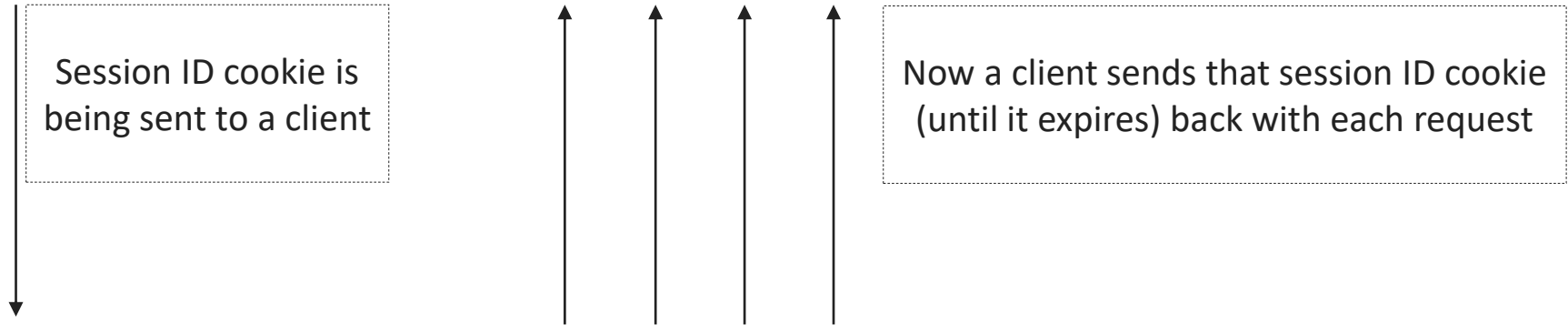
- Information about the session itself (session identifier, creation time, time last accessed, etc.)
- Contextual information about the user (client login state, for example, plus whatever else the Web application needs to save).

<https://docs.progress.com/bundle/pas-for-openedge-administration-117/page/Overview-of-HTTP-sessions.html>

Superglobals: \$_SESSION

\$_SESSION data is stored at server side, but in order to “recognize” a client, special cookie with session ID has to be set.

SERVER [stores serialized session data]



CLIENT (browser) [stores only session ID cookie]

Sessions vs Cookies

Session

Data is stored at server

Good for temporary storage

Easily handles lot of data

Session ID may be “stolen”

Cookie

Data is stored at client

Good for “long-term” storage

Best for small data chunks

100% of data may be “stolen”

Some final demo of arrays processing...

Imagine, we have to create a function that increments (by a given value) all numeric elements of an array...

Some final demo of arrays processing...

This function accepts an array by value:

```
<?php

// This function accepts an array by value:
function incrementNumericElementsBV(array $inputArray, int|float $incrementValue = 1) : array
{
    foreach ($inputArray as $key => $value) {
        if (is_array($value)) {
            $inputArray[$key] = incrementNumericElementsBV($value, $incrementValue);
        } elseif ((is_double($value)) || (is_integer($value))) {
            $inputArray[$key] = $value + $incrementValue;
        }
    }
    return $inputArray;
}

$someArray = [10, 'Test', '11', 98.5];
$newArray = incrementNumericElementsBV($someArray, 5);
print_r($newArray);
```

```
/*
Array
(
    [0] => 15
    [1] => Test
    [2] => '11'
    [3] => 103.5
)
*/
```

Some final demo of arrays processing...

This function accepts an array by reference:

```
<?php
```

```
// This function accepts an array by reference:
```

```
function incrementNumericElementsBR(array &$inputArray, int $incrementValue = 1)
```

```
{
```

```
    foreach ($inputArray as $key => $value) {
```

```
        if (is_array($value)) {
```

```
            incrementNumericElementsBR($value, $incrementValue);
```

```
            $inputArray[$key] = $value;
```

```
        } elseif ((is_double($value)) || (is_integer($value))) {
```

```
            $value += $incrementValue;
```

```
            $inputArray[$key] = $value;
```

```
        }
```

```
    }
```

```
}
```

```
$someArray = [10, 'Test', '11', 98.5];
```

```
incrementNumericElementsBR($someArray, 7);
```

```
print_r($someArray);
```

```
/*  
Array  
(  
    [0] => 17  
    [1] => Test  
    [2] => '11'  
    [3] => 105.5  
)  
*/
```

Array Functions

Disclaimer: вы смотрите просто запись лекции,
это НЕ специально подготовленный видеокурс!

