

# Course: PHP from scratch

by Sergey Podgornyy

Arrays and Loops



# About me



**Sergey Podgornyy**

Full-Stack Web Developer



and



# Overview

- Arrays
- Indexed arrays, Associative arrays and Multidimensional arrays
- Outputting arrays
- Sorting arrays
- Main array functions
- Superglobal arrays
- Loops: **for**, **foreach**, **while**, **do...while**
- Constructions **break**, **continue**, **die**, **exit**

# Arrays

- An array stores multiple values in one single variable:

```
1 <?php
2
3 $cars = array("Volvo", "BMW", "Toyota");
4 echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . ".";
5
```

- An array is a special variable, which can hold more than one value at a time
- In PHP, there are three types of arrays:
  - **Indexed arrays** - Arrays with a numeric index
  - **Associative arrays** - Arrays with named keys
  - **Multidimensional arrays** - Arrays containing one or more arrays



# Multidimensional Arrays

A multidimensional array is an array containing one or more arrays

**The dimension of an array indicates the number of indices you need to select an element.**

- For a two-dimensional array you need two indices to select an element
- For a three-dimensional array you need three indices to select an element

```
1 <?php
2
3 $cars = [
4     ["Volvo", 22, 18],
5     ["BMW", 15, 13],
6     ["Saab", 5, 2],
7     ["Land Rover", 17, 15],
8 ];
9
```



# Outputting arrays

- `print_r` — prints human-readable information about a variable
- `var_dump` — dumps information about a variable. This function displays structured information about one or more expressions that includes its type and value

# Sorting Arrays

Sorting function attributes				
Function name	Sorts by	Maintains key association	Order of sort	Related functions
<a href="#"><u>array_multisort()</u></a>	value	associative yes, numeric no	first array or sort options	<a href="#"><u>array_walk()</u></a>
<a href="#"><u>asort()</u></a>	value	yes	low to high	<a href="#"><u>arsort()</u></a>
<a href="#"><u>arsort()</u></a>	value	yes	high to low	<a href="#"><u>asort()</u></a>
<a href="#"><u>krsort()</u></a>	key	yes	high to low	<a href="#"><u>ksort()</u></a>
<a href="#"><u>ksort()</u></a>	key	yes	low to high	<a href="#"><u>asort()</u></a>
<a href="#"><u>natcasesort()</u></a>	value	yes	natural, case insensitive	<a href="#"><u>natsort()</u></a>
<a href="#"><u>natsort()</u></a>	value	yes	natural	<a href="#"><u>natcasesort()</u></a>
<a href="#"><u>rsort()</u></a>	value	no	high to low	<a href="#"><u>sort()</u></a>
<a href="#"><u>shuffle()</u></a>	value	no	random	<a href="#"><u>array_rand()</u></a>
<a href="#"><u>sort()</u></a>	value	no	low to high	<a href="#"><u>rsort()</u></a>
<a href="#"><u>uasort()</u></a>	value	yes	user defined	<a href="#"><u>uksort()</u></a>
<a href="#"><u>uksort()</u></a>	key	yes	user defined	<a href="#"><u>uasort()</u></a>
<a href="#"><u>usort()</u></a>	value	no	user defined	<a href="#"><u>uasort()</u></a>

# Sorting Arrays

- **sort()** - sort arrays in ascending order
- **rsort()** - sort arrays in descending order
- **asort()** - sort associative arrays in ascending order, according to the value
- **ksort()** - sort associative arrays in ascending order, according to the key
- **arsort()** - sort associative arrays in descending order, according to the value
- **krsort()** - sort associative arrays in descending order, according to the key





# Array functions

## Stack

- **array\_push()** - Inserts one or more elements to the end of an array
- **array\_pop()** - Deletes the last element of an array

## Queue

- **array\_unshift()** - Adds one or more elements to the beginning of an array
- **array\_shift()** - Removes the first element from an array, and returns the value of the removed element

# Array functions

- **array\_merge()** - Merges one or more arrays into one array
- **array\_key\_exists()** - Checks if the specified key exists in the array
- **count()** - Returns the number of elements in an array
- **in\_array()** - Checks if a specified value exists in an array
- **list()** - Assigns variables as if they were an array



# Global Variables - Superglobals

Several predefined variables in PHP are "superglobals", which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special

The PHP superglobal variables are:

- `$GLOBALS`
- `$_SERVER`
- `$_REQUEST`
- `$_POST`
- `$_GET`
- `$_FILES`
- `$_ENV`
- `$_COOKIE`
- `$_SESSION`



# Global Variables - Superglobals

- **`$GLOBALS`** is a PHP super global variable which is used to access global variables from anywhere in the PHP script
- **`$_SERVER`** is a PHP super global variable which holds information about headers, paths, and script locations
- **`$_REQUEST`** is used to collect data after submitting an HTML form
- **`$_POST`** is widely used to collect form data after submitting an HTML form with method="post"
- **`$_GET`** can also be used to collect form data after submitting an HTML form with method="get"

# Loops

In PHP, we have the following looping statements:

- **while** - loops through a block of code as long as the specified condition is true
- **do...while** - loops through a block of code once, and then repeats the loop as long as the specified condition is true
- **for** - loops through a block of code a specified number of times
- **foreach** - loops through a block of code for each element in an array

# while loop

The **while** loop executes a block of code as long as the specified condition is **true**

```
1 <?php
2
3 while (condition is true) {
4     // code to be executed;
5 }
6
```

# do...while loop

The **do...while** loop will always execute the block of code **once**, it will **then check** the condition, and repeat the loop while the specified condition is **true**

```
1 <?php
2
3 do {
4     // code to be executed;
5 } while (condition is true);
6
```



# for loop

The for loop is used when you know in advance how many times the script should run

```
1 <?php
2
3 for (init counter; test counter; increment counter) {
4     //code to be executed;
5 }
6
```

- **init counter**: Initialize the loop counter value
- **test counter**: Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends
- **increment counter**: Increases the loop counter value



# foreach loop

The foreach loop works only on arrays, and is used to loop through each key/value pair in an array

```
1 <?php
2
3 foreach ($array as /* $key => */ $value) {
4     // code to be executed;
5 }
6
```

- For **every loop iteration**, the value of the current array element is assigned to \$value and the array pointer is moved by one, **until it reaches the last** array element



# Useful resources

- [PHP array functions](#)
- [Learn PHP array functions](#)

# Thanks for your attention

Q & A

