



**Institut  
supérieur  
d'informatique**

---

# **PHP Web1909A**



**Amirhossein Ghasemi**

**May 2020**

255 Crémazie Est, bureau 100, Montréal (Québec) H2M 1M2  
Téléphone : (514) 842-2426, télécopieur : (514) 842-2084  
[www.isi-mtl.com](http://www.isi-mtl.com)

## Table of Contents

1	PHP Data Types .....	5
1.1	PHP String.....	5
1.2	PHP Integer .....	6
1.3	PHP Float.....	7
1.4	PHP Boolean.....	8
1.5	PHP Array .....	8
1.6	PHP Object .....	9
1.7	PHP NULL Value .....	10
2	PHP Strings.....	11
2.1	strlen() - Return the Length of a String .....	11
2.2	str_word_count() - Count Words in a String.....	12
2.3	strrev() - Reverse a String.....	12
2.4	strpos() - Search For a Text Within a String .....	13
2.5	str_replace() - Replace Text Within a String .....	14
3	PHP Numbers .....	14
3.1	PHP Integers.....	15
3.2	PHP Floats .....	16
3.3	PHP Infinity.....	17
3.4	PHP NaN.....	18
3.5	PHP Numerical Strings .....	19
3.6	PHP Casting Strings and Floats to Integers .....	20
4	PHP Constants.....	21
4.1	PHP Constant Arrays .....	23
4.2	Constants are Global.....	24
5	PHP Operators.....	25
5.1	PHP Arithmetic Operators.....	25
5.2	PHP Assignment Operators.....	29
5.3	PHP Comparison Operators .....	32
5.4	PHP Increment / Decrement Operators .....	38
5.5	PHP Logical Operators.....	40
5.6	PHP String Operators .....	44
5.7	PHP Array Operators.....	45

5.8	PHP Conditional Assignment Operators .....	49
6	PHP Conditional Statements .....	51
6.1	PHP - The if Statement .....	51
6.2	PHP - The if...else Statement .....	52
6.3	PHP - The if...elseif...else Statement .....	53
7	PHP switch Statement .....	56
8	PHP Loops .....	57
8.1	PHP while Loop .....	58
8.2	PHP do while Loop .....	61
8.3	PHP for Loop .....	63
8.4	PHP foreach Loop .....	66
8.5	The break statement .....	68
8.6	The continue statement .....	69
9	PHP Functions .....	70
9.1	PHP Built-in Functions .....	70
9.2	PHP User Defined Functions .....	70
9.3	PHP Function Arguments .....	71
9.4	PHP Default Argument Value .....	75
9.5	PHP Functions - Returning values .....	76
9.6	PHP Return Type Declarations .....	76
10	PHP Arrays .....	78
10.1	Create an Array in PHP .....	79
10.2	PHP Indexed Arrays .....	80
10.3	Loop Through an Indexed Array .....	81
10.4	PHP Associative Arrays .....	81
10.5	Loop Through an Associative Array .....	82
10.6	PHP Multidimensional Arrays .....	83
10.7	PHP - Two-dimensional Arrays .....	84
10.8	PHP - Sort Functions for Arrays .....	87
10.9	Sort Array in Ascending Order - sort() .....	88
10.10	Sort Array in Descending Order - rsort() .....	89
10.11	Sort Array (Ascending Order), According to Value - asort() .....	91
10.12	Sort Array (Ascending Order), According to Key - ksort() .....	92

10.13	Sort Array (Descending Order), According to Value - arsort() .....	92
10.14	Sort Array (Descending Order), According to Key - krsort().....	93

## 1 PHP Data Types

Variables can store data of different types, and different data types can do different things.

PHP supports the following data types:

- String
- Integer
- Float (floating point numbers - also called double)
- Boolean
- Array
- Object
- NULL
- Resource

### 1.1 PHP String

A string is a **sequence of characters**, like "Hello world!".

A string can be any text inside quotes. You can use **single or double quotes**:

**Example (E1):**

```

1  <!DOCTYPE html>
2  <html>
3      <body>
4          <?php
5              $x = "Hello world!";
6              $y = 'Hello world!';
7
8              echo $x;
9              echo "<br>";
10             echo $y;
11         ?>
12     </body>
13 </html>

```

## OUTPUT

Hello world!  
Hello world!

## 1.2 PHP Integer

An integer data type is a non-decimal number between -2,147,483,648 and 2,147,483,647.

### Rules for integers:

- An integer must have at least one digit
- An integer must not have a decimal point
- An integer can be either positive or negative
- Integers can be specified in decimal (base 10), hexadecimal (base 16), octal (base 8), or binary (base 2) notation
- In the following example \$x is an integer. The PHP **var\_dump()** function **returns the data type and value:**

## Example (E2):

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     $x = 5985;
7     var_dump($x);
8 >?>
9
10 </body>
11 </html>
```

## OUTPUT

```
int(5985)
```

## 1.3 PHP Float

A float (floating point number) is a number with a **decimal point** or a number in exponential form.

In the following example \$x is a float. The PHP **var\_dump()** function returns the data type and value:

## Example (E3):

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     $x = 5985.365;
7     var_dump($x);
8 >?>
9
10 </body>
11 </html>
```

## OUTPUT

float(5985.365)

### 1.4 PHP Boolean

A Boolean represents two possible states: TRUE or FALSE.

**\$x = true;**

**\$y = false;**

### 1.5 PHP Array

An array stores **multiple values in one single variable**.

In the following example \$cars is an array. The PHP var\_dump() function returns the **data type and value**:

Example (E4):

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6 $cars = array("Volvo", "BMW", "Toyota");
7 var_dump($cars);
8 ?>
9
10 </body>
11 </html>
```

## OUTPUT



```
array(3) { [0]=> string(5) "Volvo" [1]=> string(3) "BMW" [2]=> string(6) "Toyota" }
```

## 1.6 PHP Object

An object is a data type which **stores data and information on how to process that data.**

In PHP, an object must be **explicitly declared.**

**First, we must declare a class of object. For this, we use the class keyword. A class is a structure that can contain properties and methods:**

### Example (E5):

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6 class Car {
7     function Car() {
8         $this->model = "VW";
9     }
10 }
11 // create an object
12 $herbie = new Car();
13
14 // show object properties
15 echo $herbie->model;
16 ?>
17
18 </body>
19 </html>
```

### OUTPUT

VW

## 1.7 PHP NULL Value

Null is a special data type which can have only one value: NULL.

A variable of data type NULL is a variable that has no value assigned to it.

**Tip: If a variable is created without a value, it is automatically assigned a value of NULL.**

**Variables can also be emptied by setting the value to NULL:**

Example (E6):

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     $x = "Hello world!";
7     $x = null;
8     var_dump($x);
9 >
10
11 </body>
12 </html>
```

**OUTPUT**

**NULL**

## 2 PHP Strings

A string is a sequence of characters, like "Hello world!".

### PHP String Functions

we will look at some commonly used functions to manipulate strings.

#### 2.1 strlen() - Return the Length of a String

The PHP strlen() function returns the length of a string.

**Example (E7):**

**Return the length of the string "Hello world!":**



```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     echo strlen("Hello world!");
7 >?>
8
9 </body>
10 </html>
```

**OUTPUT**

12

## 2.2 str\_word\_count() - Count Words in a String

The PHP str\_word\_count() function counts the number of words in a string.

**Example (E8):**

**Count the number of words in the string "Hello world!":**

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     echo str_word_count("Hello world!");
7 >?>
8
9 </body>
10 </html>
```

**OUTPUT**

2

## 2.3 strrev() - Reverse a String

The PHP strrev() function reverses a string.

**Example (E9):**

**Reverse the string "Hello world!":**

```

1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     echo strrev("Hello world!");
7 >?>
8
9 </body>
10 </html>

```

## OUTPUT

!dlrow olleH

## 2.4 strpos() - Search For a Text Within a String

The PHP strpos() function searches for a specific text within a string. If a match is found, the function returns the character position of the first match. If no match is found, it will return FALSE.

### Example (E10):

**Search for the text "world" in the string "Hello world!":**

```

1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     echo strpos("Hello world!", "world");
7 >?>
8
9 </body>
10 </html>

```

## OUTPUT

## 6

**Tip: The first character position in a string is 0 (not 1).**

### 2.5 str\_replace() - Replace Text Within a String

The PHP str\_replace() function replaces some characters with some other characters in a string.

**Example (E11):**

**Replace the text "world" with "Dolly":**

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     echo str_replace("world", "Dolly", "Hello world!");
7 >?>
8
9 </body>
10 </html>
```

### OUTPUT

Hello Dolly!

## 3 PHP Numbers

we will look in depth into Integers, Floats, and Number Strings.

PHP Numbers

One thing to notice about PHP is that it provides automatic data type conversion.

So, if you assign an integer value to a variable, the type of that variable will automatically be an integer. Then, if you assign a string to the same variable, the type will change to a string.

This automatic conversion can sometimes break your code.

### 3.1 PHP Integers

- An integer is a number without any decimal part.
- 2, 256, -256, 10358, -179567 are all integers. While 7.56, 10.0, 150.67 are floats.
- So, an integer data type is a non-decimal number between -2147483648 and 2147483647. A value greater (or lower) than this, will be stored as float, because it exceeds the limit of an integer.
- Another important thing to know is that even if  $4 * 2.5$  is 10, the result is stored as float, because one of the operands is a float (2.5).

Here are some rules for integers:

- An integer must have at least one digit
- An integer must not have a decimal point
- An integer can be either positive or negative
- Integers can be specified in three formats: decimal (10-based), hexadecimal (16-based - prefixed with 0x) or octal (8-based - prefixed with 0)

**PHP has the following functions to check if the type of a variable is integer:**

- `is_int()`
- `is_integer()` - alias of `is_int()`
- `is_long()` - alias of `is_int()`

**Example (E12):**

**Check if the type of a variable is integer:**

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6 // Check if the type of a variable is integer
7 $x = 5985;
8 var_dump(is_int($x));
9
10 echo "<br>";
11
12 // Check again...
13 $x = 59.85;
14 var_dump(is_int($x));
15 ?>
16
17 </body>
18 </html>
```

## OUTPUT

```
bool(true)
bool(false)
```

**Tip: The `var_dump()` function is used to dump information about a variable.**

## 3.2 PHP Floats



- A float is a number with a decimal point or a number in exponential form.
- 2.0, 256.4, 10.358, 7.64E+5, 5.56E-5 are all floats.
- The float data type can commonly store a value up to 1.7976931348623E+308 (platform dependent) and have a maximum precision of 14 digits.

PHP has the following functions to check if the type of a variable is float:

- is\_float()
- is\_double() - alias of is\_float()

### Example (E13):

#### Check if the type of a variable is float:

```

1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6 // Check if the type of a variable is float
7 $x = 10.365;
8 var_dump(is_float($x));
9 ?>
10
11 </body>
12 </html>

```

### OUTPUT

bool(true)

## 3.3 PHP Infinity

**A numeric value that is larger than PHP\_FLOAT\_MAX is considered infinite.**

PHP has the following functions to check if a numeric value is finite or infinite:

- **is\_finite()**
- **is\_infinite()**

However, the PHP var\_dump() function returns the data type and value:

**Example (E14):**

**Check if a numeric value is finite or infinite:**

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6 // Check if a numeric value is finite or infinite
7 $x = 1.9e411;
8 var_dump($x);
9 ?>
10
11 </body>
12 </html>
```

**OUTPUT**

float(INF)

## 3.4 PHP NaN

**NaN stands for Not a Number.**

- NaN is used for impossible mathematical operations.

- PHP has the following functions to check if a value is not a number:  
`is_nan()`

However, the PHP `var_dump()` function returns the data type and value:

### Example (E15):

**Invalid calculation will return a NaN value:**

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6 // Invalid calculation will return a NaN value
7 $x = acos(8);
8 var_dump($x);
9 ?>
10
11 </body>
12 </html>
```

### OUTPUT

```
float(NAN)
```

## 3.5 PHP Numerical Strings

The PHP `is_numeric()` function can be used to find whether a variable is numeric. The function returns true if the variable is a number or a numeric string, false otherwise.

### Example(E16):

**Check if the variable is numeric:**

```

1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6 // Check if the variable is numeric
7 $x = 5985;
8 var_dump(is_numeric($x));
9
10 echo "<br>";
11
12 $y = "5985";
13 var_dump(is_numeric($x));
14
15 echo "<br>";
16
17 $z = "59.85" + 100;
18 var_dump(is_numeric($x));
19
20 echo "<br>";
21
22 $w = "Hello";
23 var_dump(is_numeric($x));
24 ?>
25
26 </body>
27 </html>

```

## OUTPUT

```

bool(true)
bool(true)
bool(true)
bool(false)

```

## 3.6 PHP Casting Strings and Floats to Integers

Sometimes you need to cast a numerical value into another data type.

The (int), (integer), or intval() function are often used to convert a value to an integer.

## Example (E17):

### Cast float and string to integer:

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6 // Cast float to int
7 $x = 23465.768;
8 $int_cast = (int)$x;
9 echo $int_cast;
10
11 echo "<br>";
12
13 // Cast string to int
14 $x = "23465.768";
15 $int_cast = (int)$x;
16 echo $int_cast;
17 ?>
18
19 </body>
20 </html>
```

### OUTPUT

23465  
23465

## 4 PHP Constants

- Constants are like variables except that once they are defined, they cannot be changed or undefined.
- A constant is an identifier (name) for a simple value. The value cannot be changed during the script.
- A valid constant name starts with a letter or underscore (no \$ sign before the constant name).

**Note:** Unlike variables, constants are automatically global across the entire script.

**To create a constant, use the `define()` function.**

## Syntax

`define(name, value, case-insensitive)`

Parameters:

name: Specifies the name of the constant

value: Specifies the value of the constant

case-insensitive: Specifies whether the constant name should be case-insensitive. Default is false

## Example (E18):

### Create a constant with a case-sensitive name:

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6 // case-sensitive constant name
7 define("GREETING", "Welcome to www.isi-international.com!");
8 echo GREETING;
9 ?>
10
11 </body>
12 </html>
```

## OUTPUT

Welcome to www.isi-international.com!

## Example (E19):

## Create a constant with a case-insensitive name:

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     // case-insensitive constant name
7     define("GREETING", "Welcome to www.isi-international.com!", true);
8     echo greeting;
9 >
10
11 </body>
12 </html>
```

### OUTPUT

Welcome to www.isi-international.com!

## 4.1 PHP Constant Arrays

In PHP7, you can create an Array constant using the define() function.

### Example (E20):

Create an Array constant:

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6 define("cars", [
7     "Alfa Romeo",
8     "BMW",
9     "Toyota"
10  ]);
11 echo cars[0];
12 ?>
13
14 </body>
15 </html>
```

## OUTPUT

Alfa Romeo

## 4.2 Constants are Global

Constants are automatically global and can be used across the entire script.

### Example (E21):

**This example uses a constant inside a function, even if it is defined outside the function:**



```

1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     define("GREETING", "Welcome to www.isi-international.com!");
7
8     function myTest() {
9         echo GREETING;
10    }
11
12    myTest();
13 ?>
14
15 </body>
16 </html>

```

## OUTPUT

Welcome to www.isi-international.com!

## 5 PHP Operators

Operators are used to perform operations on variables and values.

PHP divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Increment/Decrement operators
- Logical operators
- String operators
- Array operators
- Conditional assignment operators

### 5.1 PHP Arithmetic Operators

The PHP arithmetic operators are used with numeric values to perform common arithmetical operations, such as addition, subtraction, multiplication etc.

Operator	Name	Example	Result
+	Addition	$\$x + \$y$	Sum of $\$x$ and $\$y$
-	Subtraction	$\$x - \$y$	Difference of $\$x$ and $\$y$
*	Multiplication	$\$x * \$y$	Product of $\$x$ and $\$y$
/	Division	$\$x / \$y$	Quotient of $\$x$ and $\$y$
%	Modulus	$\$x \% \$y$	Remainder of $\$x$ divided by $\$y$
**	Exponentiation	$\$x ** \$y$	Result of raising $\$x$ to the $\$y$ 'th power

### Example (E22): Addition

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     $x = 10;
7     $y = 6;
8
9     echo $x + $y;
10 ?>
11
12 </body>
13 </html>
```

### OUTPUT

16

### Example (E23): Subtraction

```

1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     $x = 10;
7     $y = 6;
8
9     echo $x - $y;
10 ?>
11
12 </body>
13 </html>

```

## OUTPUT

4

## Example (E24): Multiplication

```

1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     $x = 10;
7     $y = 6;
8
9     echo $x * $y;
10 ?>
11
12 </body>
13 </html>

```

## OUTPUT

60

## Example (E25): Division

```

1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     $x = 10;
7     $y = 6;
8
9     echo $x / $y;
10 ?>
11
12 </body>
13 </html>

```

## OUTPUT

1.6666666666667

## Example (E26): Modulus

```

1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     $x = 10;
7     $y = 6;
8
9     echo $x % $y;
10 ?>
11
12 </body>
13 </html>

```

## OUTPUT

4

## Example (E27): Exponentiation

```

1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     $x = 10;
7     $y = 3;
8
9     echo $x ** $y;
10 ?>
11
12 </body>
13 </html>

```

## OUTPUT

1000

## 5.2 PHP Assignment Operators

The PHP assignment operators are used with numeric values to write a value to a variable.

The basic assignment operator in PHP is "=". It means that the left operand gets set to the value of the assignment expression on the right.

Assignment	Same as...	Description
<code>x = y</code>	<code>x = y</code>	The left operand gets set to the value of the expression on the right
<code>x += y</code>	<code>x = x + y</code>	Addition
<code>x -= y</code>	<code>x = x - y</code>	Subtraction
<code>x *= y</code>	<code>x = x * y</code>	Multiplication
<code>x /= y</code>	<code>x = x / y</code>	Division
<code>x %= y</code>	<code>x = x % y</code>	Modulus

**Example (E28): The left operand gets set to the value of the expression on the right**

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     $x = 10;
7     echo $x;
8 >?>
9
10 </body>
11 </html>
```

**OUTPUT**

10

**Example (E29): Addition**

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     $x = 20;
7     $x += 100;
8
9     echo $x;
10 >?>
11
12 </body>
13 </html>
```

**OUTPUT**

120

**Example (E30): Subtraction**

```

1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     $x = 50;
7     $x -= 30;
8
9     echo $x;
10 ?>
11
12 </body>
13 </html>

```

## OUTPUT

20

## Example (E31): Multiplication

```

1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     $x = 10;
7     $y = 6;
8
9     echo $x * $y;
10 ?>
11
12 </body>
13 </html>

```

## OUTPUT

60

## Example (E32): Division

```

1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     $x = 10;
7     $x /= 5;
8
9     echo $x;
10 ?>
11
12 </body>
13 </html>

```

## OUTPUT

2

## Example (E33): Modulus

```

1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     $x = 15;
7     $x %= 4;
8
9     echo $x;
10 ?>
11
12 </body>
13 </html>

```

## OUTPUT

3

## 5.3 PHP Comparison Operators

The PHP comparison operators are used to compare two values (number or string):



Operator	Name	Example	Result
==	Equal	<code>\$x == \$y</code>	Returns true if <code>\$x</code> is equal to <code>\$y</code>
===	Identical	<code>\$x === \$y</code>	Returns true if <code>\$x</code> is equal to <code>\$y</code> , and they are of the same type
!=	Not equal	<code>\$x != \$y</code>	Returns true if <code>\$x</code> is not equal to <code>\$y</code>
<>	Not equal	<code>\$x &lt;&gt; \$y</code>	Returns true if <code>\$x</code> is not equal to <code>\$y</code>
!==	Not identical	<code>\$x !== \$y</code>	Returns true if <code>\$x</code> is not equal to <code>\$y</code> , or they are not of the same type
>	Greater than	<code>\$x &gt; \$y</code>	Returns true if <code>\$x</code> is greater than <code>\$y</code>
<	Less than	<code>\$x &lt; \$y</code>	Returns true if <code>\$x</code> is less than <code>\$y</code>
>=	Greater than or equal to	<code>\$x &gt;= \$y</code>	Returns true if <code>\$x</code> is greater than or equal to <code>\$y</code>
<=	Less than or equal to	<code>\$x &lt;= \$y</code>	Returns true if <code>\$x</code> is less than or equal to <code>\$y</code>
<=>	Spaceship	<code>\$x &lt;=&gt; \$y</code>	Returns an integer less than, equal to, or greater than zero, depending on if <code>\$x</code> is less than, equal to, or greater than <code>\$y</code> . Introduced in PHP 7.

## Example (E34): Equal

```

1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     $x = 100;
7     $y = "100";
8
9     var_dump($x == $y); // returns true because values are equal
10 >?
11
12 </body>
13 </html>

```

## OUTPUT

bool(true)

## Example (E35): Identical

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     $x = 100;
7     $y = "100";
8
9     var_dump($x === $y); // returns false because types are not equal
10 ?>
11
12 </body>
13 </html>
```

## OUTPUT

bool(false)

## Example (E36): Not equal

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     $x = 100;
7     $y = "100";
8
9     var_dump($x != $y); // returns false because values are equal
10 ?>
11
12 </body>
13 </html>
```

## OUTPUT

bool(false)

## Example (E37): Not equal

```

1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     $x = 100;
7     $y = "100";
8
9     var_dump($x <> $y); // returns false because values are equal
10 >?>
11
12 </body>
13 </html>

```

## OUTPUT

bool(false)

## Example (E38): Not identical

```

1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     $x = 100;
7     $y = "100";
8
9     var_dump($x !== $y); // returns true because types are not equal
10 >?>
11
12 </body>
13 </html>

```

## OUTPUT

bool(true)

## Example (E39): Greater than

```

1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     $x = 100;
7     $y = 50;
8
9     var_dump($x > $y); // returns true because $x is greater than $y
10    ?>
11
12 </body>
13 </html>

```

## OUTPUT

bool(true)

## Example (E40): Less than

```

1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     $x = 10;
7     $y = 50;
8
9     var_dump($x < $y); // returns true because $x is less than $y
10    ?>
11
12 </body>
13 </html>

```

## OUTPUT

bool(true)

## Example (E41): Greater than or equal to

```

1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     $x = 50;
7     $y = 50;
8
9     var_dump($x >= $y); // returns true because $x is greater than or equal to $y
10    ?>
11
12 </body>
13 </html>

```

## OUTPUT

bool(true)

## Example (E42): Less than or equal to

```

1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     $x = 50;
7     $y = 50;
8
9     var_dump($x <= $y); // returns true because $x is less than or equal to $y
10    ?>
11
12 </body>
13 </html>

```

## OUTPUT

bool(true)

## Example (E43): Spaceship

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     $x = 5;
7     $y = 10;
8
9     echo ($x <=> $y); // returns -1 because $x is less than $y
10    echo "<br>";
11
12    $x = 10;
13    $y = 10;
14
15    echo ($x <=> $y); // returns 0 because values are equal
16    echo "<br>";
17
18    $x = 15;
19    $y = 10;
20
21    echo ($x <=> $y); // returns +1 because $x is greater than $y
22    ?>
23
24 </body>
25 </html>
```

## OUTPUT

```
-1
0
1
```

## 5.4 PHP Increment / Decrement Operators

The PHP increment operators are used to increment a variable's value.

The PHP decrement operators are used to decrement a variable's value

Operator	Name	Description
++\$x	Pre-increment	Increments \$x by one, then returns \$x
\$x++	Post-increment	Returns \$x, then increments \$x by one
--\$x	Pre-decrement	Decrements \$x by one, then returns \$x
\$x--	Post-decrement	Returns \$x, then decrements \$x by one

### Example (E44): Pre-increment

```

1  <!DOCTYPE html>
2  <html>
3  <body>
4
5  <?php
6      $x = 10;
7      echo ++$x;
8  ?>
9
10 </body>
11 </html>

```

### OUTPUT

11

### Example (E45): Post-increment

```

1  <!DOCTYPE html>
2  <html>
3  <body>
4
5  <?php
6      $x = 10;
7      echo $x++;
8  ?>
9
10 </body>
11 </html>

```

### OUTPUT

10

### Example (E46): Pre-decrement

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     $x = 10;
7     echo --$x;
8 >?>
9
10 </body>
11 </html>
```

### OUTPUT

9

### Example (E47): Post-decrement

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     $x = 10;
7     echo $x--;
8 >?>
9
10 </body>
11 </html>
```

### OUTPUT

10

## 5.5 PHP Logical Operators

The PHP logical operators are used to combine conditional statements.



Operator	Name	Example	Result
and	And	\$x and \$y	True if both \$x and \$y are true
or	Or	\$x or \$y	True if either \$x or \$y is true
xor	Xor	\$x xor \$y	True if either \$x or \$y is true, but not both
&&	And	\$x && \$y	True if both \$x and \$y are true
	Or	\$x    \$y	True if either \$x or \$y is true
!	Not	!\$x	True if \$x is not true

## Example (E48): And

```

1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     $x = 100;
7     $y = 50;
8
9     if ($x == 100 and $y == 50) {
10         echo "Hello world!";
11     }
12     ?>
13
14 </body>
15 </html>

```

## OUTPUT

Hello world!

## Example (E49): Or

```

1  <!DOCTYPE html>
2  <html>
3  <body>
4
5  <?php
6      $x = 100;
7      $y = 50;
8
9  if ($x == 100 or $y == 80) {
10     echo "Hello world!";
11 }
12 ?>
13
14 </body>
15 </html>

```

## OUTPUT

Hello world!

## Example (E50): Xor

```

1  <!DOCTYPE html>
2  <html>
3  <body>
4
5  <?php
6      $x = 100;
7      $y = 50;
8
9  if ($x == 100 Xor $y == 80) {
10     echo "Hello world!";
11 }
12 ?>
13
14 </body>
15 </html>

```

## OUTPUT

Hello world!

## Example (E51): And

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     $x = 100;
7     $y = 50;
8
9     if ($x == 100 && $y == 50) {
10         echo "Hello world!";
11     }
12 >?>
13
14 </body>
15 </html>
```

### OUTPUT

Hello world!

## Example (E52): Or

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     $x = 100;
7     $y = 50;
8
9     if ($x == 100 || $y == 80) {
10         echo "Hello world!";
11     }
12 >?>
13
14 </body>
15 </html>
```

### OUTPUT

Hello world!

## Example (E53): Not

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6 $x = 100;
7
8 if (!$x != 90) {
9     echo "Hello world!";
10 }
11 ?>
12
13 </body>
14 </html>
```

## OUTPUT

Hello world!

## 5.6 PHP String Operators

PHP has two operators that are specially designed for strings.

Operator	Name	Example	Result
.	Concatenation	\$txt1 . \$txt2	Concatenation of \$txt1 and \$txt2
.=	Concatenation assignment	\$txt1 .= \$txt2	Appends \$txt2 to \$txt1

## Example (E54): Concatenation

```

1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     $txt1 = "Hello";
7     $txt2 = " world!";
8     echo $txt1 . $txt2;
9     ?>
10
11 </body>
12 </html>

```

## OUTPUT

Hello world!

### Example (E55): Concatenation assignment

```

1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     $txt1 = "Hello";
7     $txt2 = " world!";
8     $txt1 .= $txt2;
9     echo $txt1;
10    ?>
11
12 </body>
13 </html>

```

## OUTPUT

Hello world!

## 5.7 PHP Array Operators

The PHP array operators are used to compare arrays.

Operator	Name	Example	Result
+	Union	<code>\$x + \$y</code>	Union of <code>\$x</code> and <code>\$y</code>
<code>==</code>	Equality	<code>\$x == \$y</code>	Returns true if <code>\$x</code> and <code>\$y</code> have the same key/value pairs
<code>===</code>	Identity	<code>\$x === \$y</code>	Returns true if <code>\$x</code> and <code>\$y</code> have the same key/value pairs in the same order and of the same types
<code>!=</code>	Inequality	<code>\$x != \$y</code>	Returns true if <code>\$x</code> is not equal to <code>\$y</code>
<code>&lt;&gt;</code>	Inequality	<code>\$x &lt;&gt; \$y</code>	Returns true if <code>\$x</code> is not equal to <code>\$y</code>
<code>!==</code>	Non-identity	<code>\$x !== \$y</code>	Returns true if <code>\$x</code> is not identical to <code>\$y</code>

## Example (E56): Union

```

1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6 $x = array("a" => "red", "b" => "green");
7 $y = array("c" => "blue", "d" => "yellow");
8
9 print_r($x + $y); // union of $x and $y
10 ?>
11
12 </body>
13 </html>

```

## OUTPUT

```
Array ( [a] => red [b] => green [c] => blue [d] => yellow )
```

## Example (E57): Equality

```

1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6 $x = array("a" => "red", "b" => "green");
7 $y = array("c" => "blue", "d" => "yellow");
8
9 var_dump($x == $y);
10 ?>
11
12 </body>
13 </html>

```

## OUTPUT

bool(false)

## Example (E58): Identity

```

1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6 $x = array("a" => "red", "b" => "green");
7 $y = array("c" => "blue", "d" => "yellow");
8
9 var_dump($x === $y);
10 ?>
11
12 </body>
13 </html>

```

## OUTPUT

bool(false)

## Example (E59): Inequality

```

1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6 $x = array("a" => "red", "b" => "green");
7 $y = array("c" => "blue", "d" => "yellow");
8
9 var_dump($x != $y);
10 ?>
11
12 </body>
13 </html>

```

## OUTPUT

bool(true)

## Example (E60): Inequality

```

1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6 $x = array("a" => "red", "b" => "green");
7 $y = array("c" => "blue", "d" => "yellow");
8
9 var_dump($x <> $y);
10 ?>
11
12 </body>
13 </html>

```

## OUTPUT

bool(true)

## Example (E61): Non-identity



```

1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6 $x = array("a" => "red", "b" => "green");
7 $y = array("c" => "blue", "d" => "yellow");
8
9 var_dump($x !== $y);
10 ?>
11
12 </body>
13 </html>

```

## OUTPUT

bool(true)

## 5.8 PHP Conditional Assignment Operators

The PHP conditional assignment operators are used to set a value depending on conditions:

Operator	Name	Example	Result
?:	Ternary	<pre>\$x = expr1 ? expr2 : expr3</pre>	<p>Returns the value of \$x.</p> <p>The value of \$x is <i>expr2</i> if <i>expr1</i> = TRUE.</p> <p>The value of \$x is <i>expr3</i> if <i>expr1</i> = FALSE</p>
??	Null coalescing	<pre>\$x = expr1 ?? expr2</pre>	<p>Returns the value of \$x.</p> <p>The value of \$x is <i>expr1</i> if <i>expr1</i> exists, and is not NULL.</p> <p>If <i>expr1</i> does not exist, or is NULL, the value of \$x is <i>expr2</i>.</p> <p>Introduced in PHP 7</p>

## Example (E62): Ternary

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     // if empty($user) = TRUE, set $status = "anonymous"
7     echo $status = (empty($user)) ? "anonymous" : "logged in";
8     echo("<br>");
9
10    $user = "John Doe";
11    // if empty($user) = FALSE, set $status = "logged in"
12    echo $status = (empty($user)) ? "anonymous" : "logged in";
13 ?>
14
15 </body>
16 </html>
```

## OUTPUT

anonymous  
logged in

## Example (E63): Null coalescing

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     // variable $user is the value of $_GET['user']
7     // and 'anonymous' if it does not exist
8     echo $user = $_GET["user"] ?? "anonymous";
9     echo("<br>");
10
11    // variable $color is "red" if $color does not exist or is null
12    echo $color = $color ?? "red";
13 ?>
14
15 </body>
16 </html>
```

## OUTPUT

anonymous  
red

## 6 PHP Conditional Statements

Conditional statements are used to perform different actions based on different conditions.

Very often when you write code, you want to perform different actions for different conditions. You can use conditional statements in your code to do this.

In PHP we have the following conditional statements:

- if statement - executes some code if one condition is true
- if...else statement - executes some code if a condition is true and another code if that condition is false
- if...elseif...else statement - executes different codes for more than two conditions
- switch statement - selects one of many blocks of code to be executed

### 6.1 PHP - The if Statement

The if statement executes some code if one condition is true.

Syntax

```
• if (condition) {  
    code to be executed if condition is true;  
}
```

**Example (E64):**

Output "Have a good day!" if the current time (HOUR) is less than 20:

```

1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     $t = date("H");
7
8     if ($t < "20") {
9         echo "Have a good day!";
10    }
11    ?>
12
13 </body>
14 </html>

```

## OUTPUT

Have a good day!

**Note: OUTPUT arises when code runs (Depends on the time)**

## 6.2 PHP - The if...else Statement

The if...else statement executes some code if a condition is true and another code if that condition is false.

### Syntax

```

if (condition) {
    code to be executed if condition is true;
} else {
    code to be executed if condition is false;
}

```

### Example

Output "Have a good day!" if the current time is less than 20, and "Have a good night!" otherwise:

### Example (E65):

```

1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     $t = date("H");
7
8     if ($t < "20") {
9         echo "Have a good day!";
10    } else {
11        echo "Have a good night!";
12    }
13 ?>
14
15 </body>
16 </html>

```

## OUTPUT

Have a good day!

**Note: OUTPUT arises when code runs (Depends on the time)**

## 6.3 PHP - The if...elseif...else Statement

The if...elseif...else statement executes different codes for more than two conditions.

### Syntax

```

if (condition) {
    code to be executed if this condition is true;
} elseif (condition) {
    code to be executed if first condition is false and this condition is true;
} else {
    code to be executed if all conditions are false;
}

```

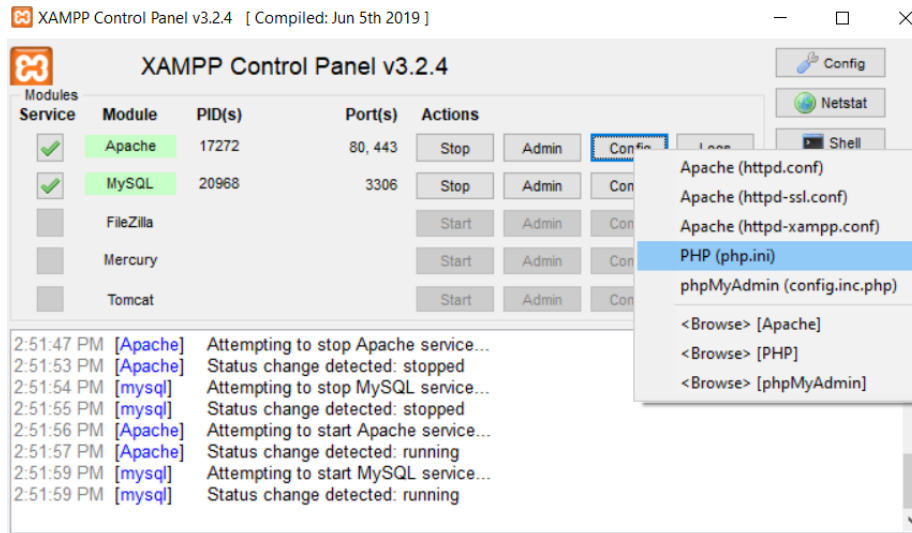
### Example (E66):

Output "Have a good morning!" if the current time is less than 10, and "Have a good day!" if the current time is less than 20. Otherwise it will output "Have a good night!":

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6 // set the default timezone to use. Available since PHP 5.1
7
8 $t = date("H");
9 echo "<p>The hour (of the server) is " . $t;
10 echo ", and will give the following message:</p>";
11
12 if ($t < "10") {
13     echo "Have a good morning!";
14 } elseif ($t < "20") {
15     echo "Have a good day!";
16 } else {
17     echo "Have a good night!";
18 }
19 ?>
20
21 </body>
22 </html>
```

To set the time on your XAMPP:

Open the file shown below:



Change the line shown below and save.

```
; Local Variables:
; tab-width: 4
; End:
[Syslog]
define_syslog_variables=Off
[Session]
define_syslog_variables=Off
[Date]
date.timezone= "America/Toronto"
[MySQL]
mysql.allow_local_infile=On
mysql.allow_persistent=On
```

After changing the line, restart your server (Xampp).

## OUTPUT

The hour (of the server) is 14, and will give the following message:

Have a good day!

**Note: OUTPUT arises when code runs (Depends on the time)**

## 7 PHP switch Statement

The switch statement is used to perform different actions based on different conditions.

Use the switch statement to select one of many blocks of code to be executed.

### Syntax

```
switch (n) {  
    case label1:  
        code to be executed if n=label1;  
        break;  
    case label2:  
        code to be executed if n=label2;  
        break;  
    case label3:  
        code to be executed if n=label3;  
        break;  
    ...  
    default:  
        code to be executed if n is different from all labels;  
}
```

This is how it works: First we have a single expression *n* (most often a variable), that is evaluated once. The value of the expression is then compared with the values for each case in the structure. If there is a match, the block of code associated with that case is executed. Use `break` to prevent the code from running into the next case automatically. The default statement is used if no match is found.

### Example (E67):



```

1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     $favcolor = "red";
7
8     switch ($favcolor) {
9         case "red":
10             echo "Your favorite color is red!";
11             break;
12         case "blue":
13             echo "Your favorite color is blue!";
14             break;
15         case "green":
16             echo "Your favorite color is green!";
17             break;
18         default:
19             echo "Your favorite color is neither red, blue, nor green!";
20     }
21 ?>
22
23 </body>
24 </html>

```

## OUTPUT

Your favorite color is red!

## 8 PHP Loops

In the following chapters you will learn how to repeat code by using loops in PHP.

Often when you write code, you want the same block of code to run repeatedly a certain number of times. So, instead of adding several almost equal code-lines in a script, we can use loops.

Loops are used to execute the same block of code again and again, if a certain condition is true.

In PHP, we have the following loop types:

while - loops through a block of code if the specified condition is true

do...while - loops through a block of code once, and then repeats the loop if the specified condition is true

for - loops through a block of code a specified number of times

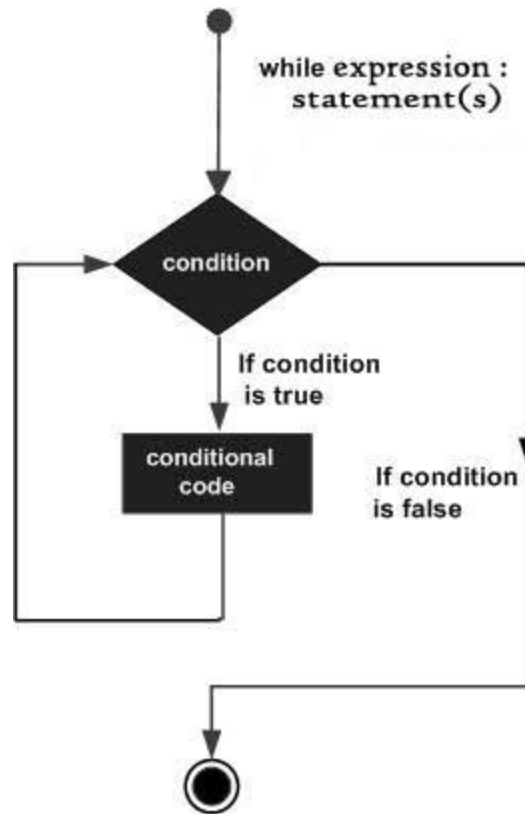
foreach - loops through a block of code for each element in an array

The following chapters will explain and give examples of each loop type.

## **8.1 PHP while Loop**

The while loop - Loops through a block of code if the specified condition is true.

if the test expression is true then the code block will be executed. After the code has executed the test expression will again be evaluated and the loop will continue until the test expression is found to be false.



The while loop executes a block of code if the specified condition is true.

## Syntax

```
while (condition is true) {  
    code to be executed;  
}
```

Example (E68):

The example below displays the numbers from 1 to 5:

```

1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     $x = 1;
7
8     while($x <= 5) {
9         echo "The number is: $x <br>";
10        $x++;
11    }
12    ?>
13
14 </body>
15 </html>

```

## OUTPUT

The number is: 1  
 The number is: 2  
 The number is: 3  
 The number is: 4  
 The number is: 5

## Example Explained

$\$x = 1$ ; - Initialize the loop counter ( $\$x$ ), and set the start value to 1

$\$x \leq 5$  - Continue the loop if  $\$x$  is less than or equal to 5

$\$x++$ ; - Increase the loop counter value by 1 for each iteration

This example counts to 100 by tens (E69):

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     $x = 0;
7
8     while($x <= 100) {
9         echo "The number is: $x <br>";
10        $x+=10;
11    }
12    ?>
13
14 </body>
15 </html>
```

## OUTPUT

The number is: 0  
The number is: 10  
The number is: 20  
The number is: 30  
The number is: 40  
The number is: 50  
The number is: 60  
The number is: 70  
The number is: 80  
The number is: 90  
The number is: 100

## 8.2 PHP do while Loop

The do...while loop - Loops through a block of code once, and then repeats the loop if the specified condition is true.

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.

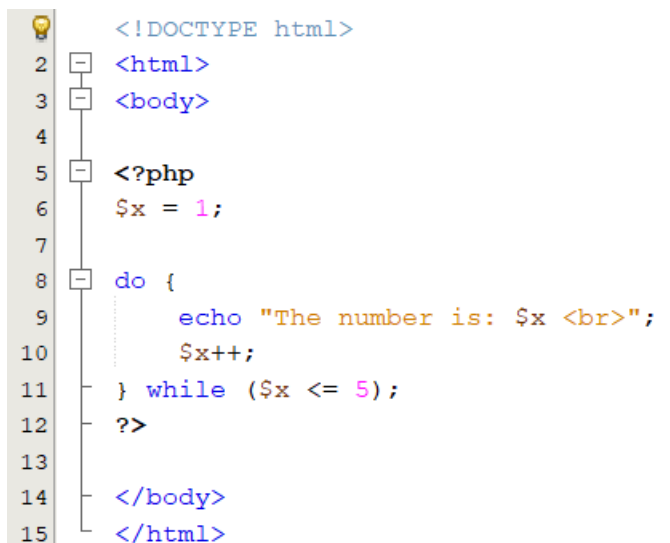
### Syntax

```
do {  
    code to be executed;  
} while (condition is true);
```

## Examples

The example below first sets a variable `$x` to 1 (`$x = 1`). Then, the do while loop will write some output, and then increment the variable `$x` with 1. Then the condition is checked (is `$x` less than, or equal to 5?), and the loop will continue to run as long as `$x` is less than, or equal to 5:

### Example (E70):



```
<!DOCTYPE html>  
<html>  
<body>  
  
<?php  
    $x = 1;  
  
    do {  
        echo "The number is: $x <br>";  
        $x++;  
    } while ($x <= 5);  
    ?>  
  
</body>  
</html>
```

## OUTPUT

```
The number is: 1  
The number is: 2  
The number is: 3  
The number is: 4  
The number is: 5
```

**Note:** In a do...while loop the condition is tested **AFTER** executing the statements within the loop. This means that the

**do...while loop will execute its statements at least once, even if the condition is false. See example below.**

This example sets the \$x variable to 6, then it runs the loop, and then the condition is checked:

### **Example (E71):**

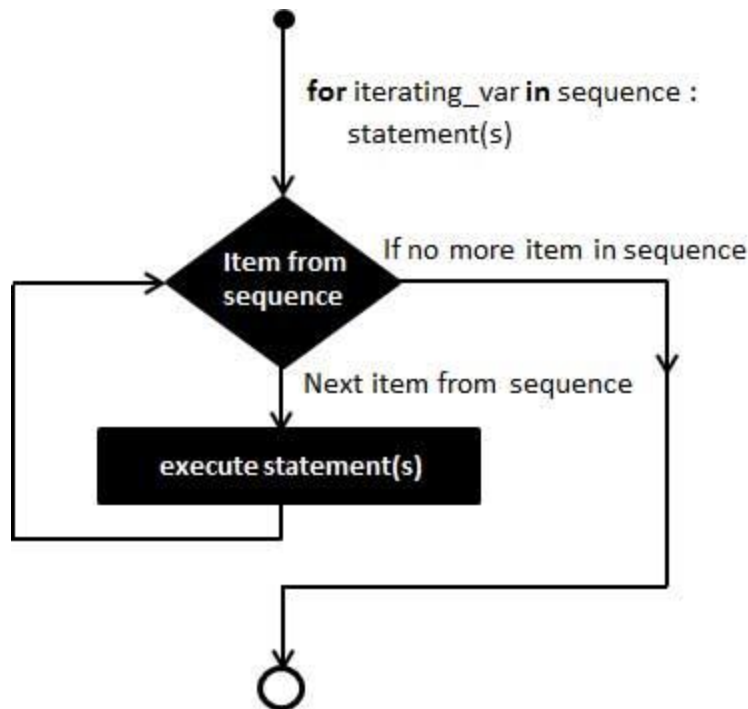
```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6     $x = 6;
7
8     do {
9         echo "The number is: $x <br>";
10        $x++;
11    } while ($x <= 5);
12    ?>
13
14 </body>
15 </html>
```

### **OUTPUT**

The number is: 6

## **8.3 PHP for Loop**

The for loop - Loops through a block of code a specified number of times.



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

## Syntax

```
for (init counter; test counter; increment counter) {  
    code to be executed for each iteration;  
}
```

## Parameters:

- `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

## Example (E72):

The example below displays the numbers from 0 to 10:



```

1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6   for ($x = 0; $x <= 10; $x++) {
7       echo "The number is: $x <br>";
8   }
9   ?>
10
11 </body>
12 </html>

```

## OUTPUT

The number is: 0  
 The number is: 1  
 The number is: 2  
 The number is: 3  
 The number is: 4  
 The number is: 5  
 The number is: 6  
 The number is: 7  
 The number is: 8  
 The number is: 9  
 The number is: 10

## Example Explained

`$x = 0;` - Initialize the loop counter (`$x`), and set the start value to 0

`$x <= 10;` - Continue the loop as long as `$x` is less than or equal to 10

`$x++` - Increase the loop counter value by 1 for each iteration

This example counts to 100 by tens:

## Example (E73)

```

1  <!DOCTYPE html>
2  <html>
3  <body>
4
5  <?php
6  for ($x = 0; $x <= 100; $x+=10) {
7      echo "The number is: $x <br>";
8  }
9  ?>
10
11 </body>
12 </html>

```

## OUTPUT

The number is: 0  
 The number is: 10  
 The number is: 20  
 The number is: 30  
 The number is: 40  
 The number is: 50  
 The number is: 60  
 The number is: 70  
 The number is: 80  
 The number is: 90  
 The number is: 100

## Example Explained

`$x = 0;` - Initialize the loop counter (`$x`), and set the start value to 0

`$x <= 100;` - Continue the loop as long as `$x` is less than or equal to 100

`$x+=10` - Increase the loop counter value by 10 for each iteration

## 8.4 PHP foreach Loop

The foreach loop - Loops through a block of code for each element in an array.

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

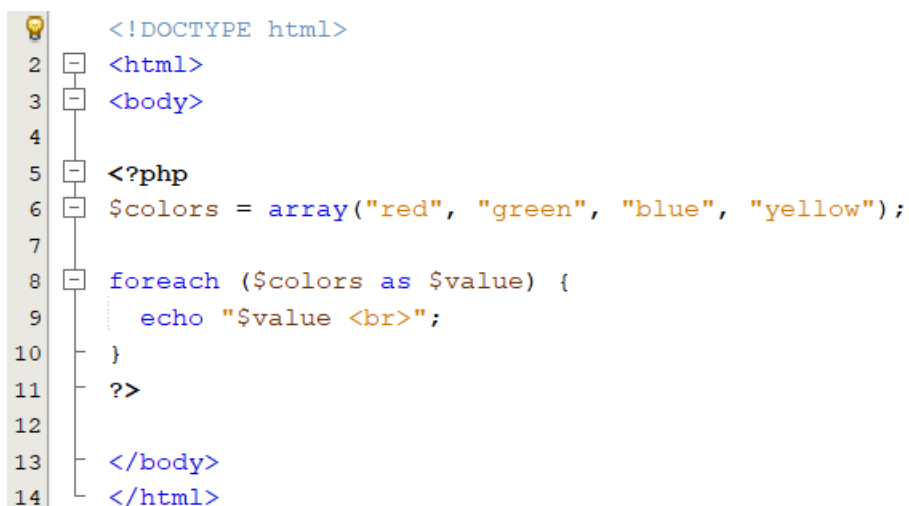
## Syntax

```
foreach ($array as $value) {  
    code to be executed;  
}
```

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.

Example (E74):

The following example will output the values of the given array (\$colors):

A screenshot of a code editor with a light blue background. On the left side, there is a vertical line with numbers 1 through 14. To the right of the numbers, the code is written in a monospaced font. The code starts with a light blue icon of a lightbulb at line 1. The code is as follows:

```
1 <!DOCTYPE html>  
2 <html>  
3 <body>  
4  
5 <?php  
6 $colors = array("red", "green", "blue", "yellow");  
7  
8 foreach ($colors as $value) {  
9     echo "$value <br>";  
10 }  
11 ?>  
12  
13 </body>  
14 </html>
```

## OUTPUT

red  
green  
blue  
yellow

The following example will output both the keys and the values of the given array (\$age):

**Example (E75):**

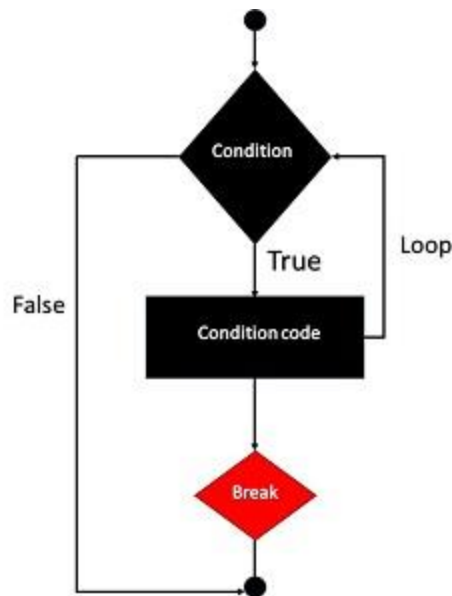
```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6 $age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
7
8 foreach($age as $x => $val) {
9     echo "$x = $val<br>";
10 }
11 ?>
12
13 </body>
14 </html>
```

## OUTPUT

Peter = 35  
Ben = 37  
Joe = 43

## 8.5 The break statement

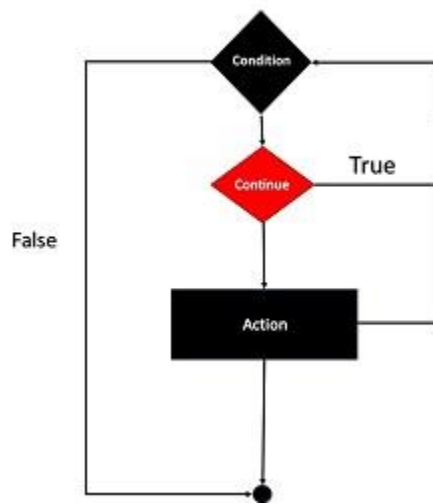
The PHP break keyword is used to terminate the execution of a loop prematurely.



The break statement is situated inside the statement block. It gives you full control and whenever you want to exit from the loop you can come out. After coming out of a loop immediate statement to the loop will be executed.

## 8.6 The continue statement

The PHP continue keyword is used to halt the current iteration of a loop, but it does not terminate the loop.



Just like the break statement the continue statement is situated inside the statement block containing the code that the loop executes, preceded by a conditional test. For the pass encountering continue statement, rest of the loop code is skipped, and next pass starts.

## **9 PHP Functions**

The real power of PHP comes from its functions.

PHP has more than 1000 built-in functions, and in addition you can create your own custom functions.

### **9.1 PHP Built-in Functions**

PHP has over 1000 built-in functions that can be called directly, from within a script, to perform a specific task.

Please check out our PHP reference for a complete overview of the PHP built-in functions.

### **9.2 PHP User Defined Functions**

Besides the built-in PHP functions, it is possible to create your own functions.

- A function is a block of statements that can be used repeatedly in a program.
- A function will not execute automatically when a page load.
- A function will be executed by a call to the function.

A user-defined function declaration starts with the word function:

```
function function_name ( [ parameters ] ) {  
    // function body  
}
```

## Syntax


```
function functionName() {  
    code to be executed;  
}
```

**Note:** A function name must start with a letter or an underscore. Function names are NOT case-sensitive.

**Tip:** Give the function a name that reflects what the function does!

In the example below, we create a function named "writeMsg()". The opening curly brace ( { ) indicates the beginning of the function code, and the closing curly brace ( } ) indicates the end of the function. The function outputs "Hello world!". To call the function, just write its name followed by brackets ( ):

### Example (E76):



```
1 <!DOCTYPE html>  
2 <html>  
3 <body>  
4  
5 <?php  
6 function writeMsg() {  
7     echo "Hello world!";  
8 }  
9  
10 writeMsg();  
11 ?>  
12  
13 </body>  
14 </html>
```

## OUTPUT

Hello world!

## 9.3 PHP Function Arguments

Information can be passed to functions through arguments. An argument is just like a variable. Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

The following example has a function with one argument (\$fname). When the familyName() function is called, we also pass along a name (e.g. Jani), and the name is used inside the function, which outputs several different first names, but an equal last name:

### Example (E77):

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6 function familyName($fname) {
7     echo "$fname Refsnes.<br>";
8 }
9
10 familyName("Jani");
11 familyName("Hege");
12 familyName("Stale");
13 familyName("Kai Jim");
14 familyName("Borge");
15 ?>
16
17 </body>
18 </html>
```

### OUTPUT

```
Jani Refsnes.
Hege Refsnes.
Stale Refsnes.
Kai Jim Refsnes.
Borge Refsnes.
```



The following example has a function with two arguments (\$fname and \$year):

### Example (E78):

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6 function familyName($fname, $year) {
7     echo "$fname Refsnes. Born in $year <br>";
8 }
9
10 familyName("Hege", "1975");
11 familyName("Stale", "1978");
12 familyName("Kai Jim", "1983");
13 ?>
14
15 </body>
16 </html>
```

### OUTPUT

```
Hege Refsnes. Born in 1975
Stale Refsnes. Born in 1978
Kai Jim Refsnes. Born in 1983
```

In the example above, notice that we did not have to tell PHP which data type the variable is.

PHP automatically associates a data type to the variable, depending on its value. Since the data types are not set in a strict sense, you can do things like adding a string to an integer without causing an error.

In PHP 7, type declarations were added. This gives us an option to specify the expected data type when declaring a function, and

by adding the strict declaration, it will throw a "Fatal Error" if the data type mismatch.

In the following example we try to send both a number and a string to the function without using strict:

### Example (E79):

```
<?php
2 function addNumbers(int $a, int $b) {
3     return $a + $b;
4 }
5 echo addNumbers(5, "5 days");
6 // since strict is NOT enabled "5 days" is changed to int(5), and it will return 10
?>
```

### OUTPUT

**Notice:** A non well formed numeric value encountered in C:\xampp\htdocs\Day2\E79.php on line 2  
10

To specify strict we need to set declare(strict\_types=1);. This must be on the very first line of the PHP file.

In the following example we try to send both a number and a string to the function, but here we have added the strict declaration:

### Example (E80):

```
<?php declare(strict_types=1); // strict requirement
2
3 function addNumbers(int $a, int $b) {
4     return $a + $b;
5 }
6 echo addNumbers(5, "5 days");
7 // since strict is enabled and "5 days" is not an integer, an error will be thrown
?>
```

## OUTPUT

**Fatal error:** Uncaught TypeError: Argument 2 passed to addNumbers() must be of the type int, string given, called in C:\xampp\htdocs\Day2\E80.php on line 6 and defined in C:\xampp\htdocs\Day2\E80.php:3  
Stack trace: #0 C:\xampp\htdocs\Day2\E80.php(6): addNumbers(5, '5 days') #1 {main} thrown in  
C:\xampp\htdocs\Day2\E80.php on line 3

The strict declaration forces things to be used in the intended way.

## 9.4 PHP Default Argument Value

The following example shows how to use a default parameter. If we call the function setHeight() without arguments it takes the default value as argument:

### Example (E81):

```
1 <?php declare(strict_types=1); // strict requirement ?>
2 <!DOCTYPE html>
3 <html>
4 <body>
5
6 <?php
7 function setHeight(int $minheight = 50) {
8     echo "The height is : $minheight <br>";
9 }
10
11 setHeight(350);
12 setHeight();
13 setHeight(135);
14 setHeight(80);
15 ?>
16
17 </body>
18 </html>
```

## OUTPUT

The height is : 350  
The height is : 50  
The height is : 135  
The height is : 80

## 9.5 PHP Functions - Returning values

To let a function, return a value, use the return statement:

### Example (E82):

```
1 <?php declare(strict_types=1); // strict requirement ?>
2 <!DOCTYPE html>
3 <html>
4 <body>
5
6 <?php
7 function sum(int $x, int $y) {
8     $z = $x + $y;
9     return $z;
10 }
11
12 echo "5 + 10 = " . sum(5,10) . "<br>";
13 echo "7 + 13 = " . sum(7,13) . "<br>";
14 echo "2 + 4 = " . sum(2,4) ;
15 ?>
16
17 </body>
18 </html>
```

### OUTPUT

5 + 10 = 15  
7 + 13 = 20  
2 + 4 = 6

## 9.6 PHP Return Type Declarations





PHP 7 also supports Type Declarations for the return statement. Like with the type declaration for function arguments, by

enabling the strict requirement, it will throw a "Fatal Error" on a type mismatch.

To declare a type for the function return, add a colon ( : ) and the type right before the opening curly ( { ) bracket when declaring the function.

In the following example we specify the return type for the function:

### Example (E83):





```
  <?php declare(strict_types=1); // strict requirement
2  function addNumbers(float $a, float $b) : float {
3     return $a + $b;
4 }
5 echo addNumbers(1.2, 5.2);
 ?>
```

### OUTPUT

6.4

You can specify a different return type, than the argument types, but make sure the return is the correct type:

### Example (E84):

```
  <?php declare(strict_types=1); // strict requirement
2  function addNumbers(float $a, float $b) : int {
3     return (int)($a + $b);
4 }
5 echo addNumbers(1.2, 5.2);
 ?>
```

### OUTPUT

6

## 10 PHP Arrays

An array stores multiple values in one single variable:

### Example (E85):

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6 $cars = array("Volvo", "BMW", "Toyota");
7 echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . ".";
8 ?>
9
10 </body>
11 </html>
```

### OUTPUT

I like Volvo, BMW and Toyota.

### What is an Array?

An array is a special variable, which can hold more than one value at a time.

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
$cars1 = "Volvo";
$cars2 = "BMW";
$cars3 = "Toyota";
```

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is to create an array!

An array can hold many values under a single name, and you can access the values by referring to an index number.

## 10.1 Create an Array in PHP

In PHP, the `array()` function is used to create an array:

```
array();
```

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

Get The Length of an Array - The `count()` Function

The `count()` function is used to return the length (the number of elements) of an array:

### Example (E86):

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6 $cars = array("Volvo", "BMW", "Toyota");
7 echo count($cars);
8 ?>
9
10 </body>
11 </html>
```

## OUTPUT

3

## 10.2 PHP Indexed Arrays

There are two ways to create indexed arrays:

The index can be assigned automatically (index always starts at 0), like this:

```
$cars = array("Volvo", "BMW", "Toyota");
```

or the index can be assigned manually:

```
$cars[0] = "Volvo";  
$cars[1] = "BMW";  
$cars[2] = "Toyota";
```

The following example creates an indexed array named \$cars, assigns three elements to it, and then prints a text containing the array values:

### Example (E87):

```
1 <!DOCTYPE html>  
2 <html>  
3 <body>  
4  
5 <?php  
6 $cars = array("Volvo", "BMW", "Toyota");  
7 echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . ".";  
8 ?>  
9  
10 </body>  
11 </html>
```

## OUTPUT



I like Volvo, BMW and Toyota.

## 10.3 Loop Through an Indexed Array

To loop through and print all the values of an indexed array, you could use a for loop, like this:

### Example (E88):

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6 $cars = array("Volvo", "BMW", "Toyota");
7 $arlength = count($cars);
8
9 for($x = 0; $x < $arlength; $x++) {
10     echo $cars[$x];
11     echo "<br>";
12 }
13 ?>
14
15 </body>
16 </html>
```

### OUTPUT

Volvo  
BMW  
Toyota

## 10.4 PHP Associative Arrays

Associative arrays are arrays that use named keys that you assign to them.

There are two ways to create an associative array:

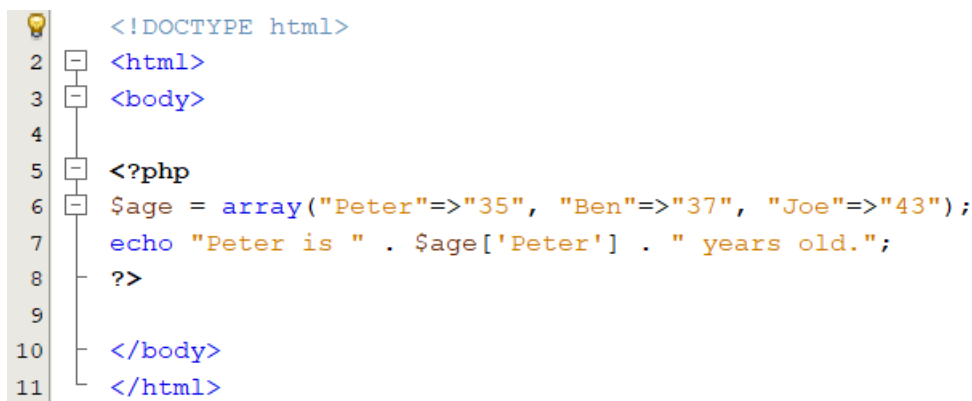
```
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
```

or:

```
$age['Peter'] = "35";  
$age['Ben'] = "37";  
$age['Joe'] = "43";
```

The named keys can then be used in a script:

### Example (E89):



```
<!DOCTYPE html>  
<html>  
<body>  
  
<?php  
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");  
echo "Peter is " . $age['Peter'] . " years old."  
?>  
  
</body>  
</html>
```

### OUTPUT

Peter is 35 years old.

## 10.5 Loop Through an Associative Array

To loop through and print all the values of an associative array, you could use a foreach loop, like this:

### Example (E90):

```

1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6 $age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
7
8 foreach($age as $x => $x_value) {
9     echo "Key=" . $x . ", Value=" . $x_value;
10    echo "<br>";
11 }
12 ?>
13
14 </body>
15 </html>

```

## OUTPUT

```

Key=Peter, Value=35
Key=Ben, Value=37
Key=Joe, Value=43

```

## 10.6 PHP Multidimensional Arrays

In the previous pages, we have described arrays that are a single list of key/value pairs.

However, sometimes you want to store values with more than one key. For this, we have multidimensional arrays.

### PHP - Multidimensional Arrays

A multidimensional array is an array containing one or more arrays.

PHP supports multidimensional arrays that are two, three, four, five, or more levels deep. However, arrays more than three levels deep are hard to manage for most people.

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

## 10.7 PHP - Two-dimensional Arrays

A two-dimensional array is an array of arrays (a three-dimensional array is an array of arrays of arrays).

First, look at the following table:

Name	Stock	Sold
Volvo	22	18
BMW	15	13
Saab	5	2
Land Rover	17	15

We can store the data from the table above in a two-dimensional array, like this:

```
$cars = array  
(  
    array("Volvo",22,18),  
    array("BMW",15,13),  
    array("Saab",5,2),  
    array("Land Rover",17,15)
```

```
array("Land Rover",17,15)
);
```

Now the two-dimensional \$cars array contains four arrays, and it has two indices: row and column.

To get access to the elements of the \$cars array we must point to the two indices (row and column):

### Example (E91):

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6 $cars = array
7 (
8     array("Volvo",22,18),
9     array("BMW",15,13),
10    array("Saab",5,2),
11    array("Land Rover",17,15)
12 );
13
14 echo $cars[0][0].": In stock: ".$cars[0][1].", sold: ".$cars[0][2].<br>;
15 echo $cars[1][0].": In stock: ".$cars[1][1].", sold: ".$cars[1][2].<br>;
16 echo $cars[2][0].": In stock: ".$cars[2][1].", sold: ".$cars[2][2].<br>;
17 echo $cars[3][0].": In stock: ".$cars[3][1].", sold: ".$cars[3][2].<br>;
18 ?>
19
20 </body>
21 </html>
```

### OUTPUT

```
Volvo: In stock: 22, sold: 18.
BMW: In stock: 15, sold: 13.
Saab: In stock: 5, sold: 2.
Land Rover: In stock: 17, sold: 15.
```

We can also put a for loop inside another for loop to get the elements of the \$cars array (we still must point to the two indices):

## Example (E92):

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6 $cars = array
7 (
8     array("Volvo",22,18),
9     array("BMW",15,13),
10    array("Saab",5,2),
11    array("Land Rover",17,15)
12 );
13
14 for ($row = 0; $row < 4; $row++) {
15     echo "<p><b>Row number $row</b></p>";
16     echo "<ul>";
17     for ($col = 0; $col < 3; $col++) {
18         echo "<li>".$cars[$row][$col]."</li>";
19     }
20     echo "</ul>";
21 }
22 ?>
23
24 </body>
25 </html>
```

## OUTPUT

**Row number 0**

- Volvo
- 22
- 18

**Row number 1**

- BMW
- 15
- 13

**Row number 2**

- Saab
- 5
- 2

**Row number 3**

- Land Rover
- 17
- 15

## 10.8 PHP - Sort Functions for Arrays

In this chapter, we will go through the following PHP array sort functions:

- `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

## 10.9 Sort Array in Ascending Order - sort()

The following example sorts the elements of the \$cars array in ascending alphabetical order:

### Example (E93):

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6 $cars = array("Volvo", "BMW", "Toyota");
7 sort($cars);
8
9 $clength = count($cars);
10 for($x = 0; $x < $clength; $x++) {
11     echo $cars[$x];
12     echo "<br>";
13 }
14 ?>
15
16 </body>
17 </html>
```

### OUTPUT

```
BMW
Toyota
Volvo
```

The following example sorts the elements of the \$numbers array in ascending numerical order:

### Example (E94):



```

1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6 $numbers = array(4, 6, 2, 22, 11);
7 sort($numbers);
8
9 $arrlength = count($numbers);
10 for($x = 0; $x < $arrlength; $x++) {
11     echo $numbers[$x];
12     echo "<br>";
13 }
14 ?>
15
16 </body>
17 </html>

```

## OUTPUT

```

2
4
6
11
22

```

### 10.10 Sort Array in Descending Order - rsort()

The following example sorts the elements of the \$cars array in descending alphabetical order:

#### Example (E95):

```

1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6 $cars = array("Volvo", "BMW", "Toyota");
7 rsort($cars);
8
9 $clength = count($cars);
10 for($x = 0; $x < $clength; $x++) {
11     echo $cars[$x];
12     echo "<br>";
13 }
14 ?>
15
16 </body>
17 </html>

```

## OUTPUT

Volvo  
Toyota  
BMW

The following example sorts the elements of the \$numbers array in descending numerical order:

### Example (E96):

```

1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6 $numbers = array(4, 6, 2, 22, 11);
7 rsort($numbers);
8
9 $arrlength = count($numbers);
10 for($x = 0; $x < $arrlength; $x++) {
11     echo $numbers[$x];
12     echo "<br>";
13 }
14 ?>
15
16 </body>
17 </html>

```

## OUTPUT

22  
11  
6  
4  
2

### 10.11 Sort Array (Ascending Order), According to Value - asort()

The following example sorts an associative array in ascending order, according to the value:

#### Example (E97):

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6 $age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
7 asort($age);
8
9 foreach($age as $x => $x_value) {
10     echo "Key=" . $x . ", Value=" . $x_value;
11     echo "<br>";
12 }
13 ?>
14
15 </body>
16 </html>
```

## OUTPUT

Key=Peter, Value=35  
Key=Ben, Value=37  
Key=Joe, Value=43

## 10.12 Sort Array (Ascending Order), According to Key - ksort()

The following example sorts an associative array in ascending order, according to the key:

### Example (E98):

```
1 <!-- ? --> <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6 $age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
7 ksort($age);
8
9 foreach($age as $x => $x_value) {
10     echo "Key=" . $x . ", Value=" . $x_value;
11     echo "<br>";
12 }
13 ?>
14
15 </body>
16 </html>
```

### OUTPUT

```
Key=Ben, Value=37
Key=Joe, Value=43
Key=Peter, Value=35
```

## 10.13 Sort Array (Descending Order), According to Value - arsort()

The following example sorts an associative array in descending order, according to the value:

## Example (E99):

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6 $age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
7 arsort($age);
8
9 foreach($age as $x => $x_value) {
10     echo "Key=" . $x . ", Value=" . $x_value;
11     echo "<br>";
12 }
13 ?>
14
15 </body>
16 </html>
```

## OUTPUT

```
Key=Joe, Value=43
Key=Ben, Value=37
Key=Peter, Value=35
```

## 10.14 Sort Array (Descending Order), According to Key - krsort()

The following example sorts an associative array in descending order, according to the key:

## Example (E100):

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <?php
6 $age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
7 krsort($age);
8
9 foreach($age as $x => $x_value) {
10     echo "Key=" . $x . ", Value=" . $x_value;
11     echo "<br>";
12 }
13 ?>
14
15 </body>
16 </html>
```

## OUTPUT

Key=Peter, Value=35  
Key=Joe, Value=43  
Key=Ben, Value=37